Universidad de las Ciencias Informáticas

Facultad 2

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Solución para la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del Centro de Informática Médica.

Autores:

Isbel López González.

Alejandro Ramón Haro Cabrera.

Tutores:

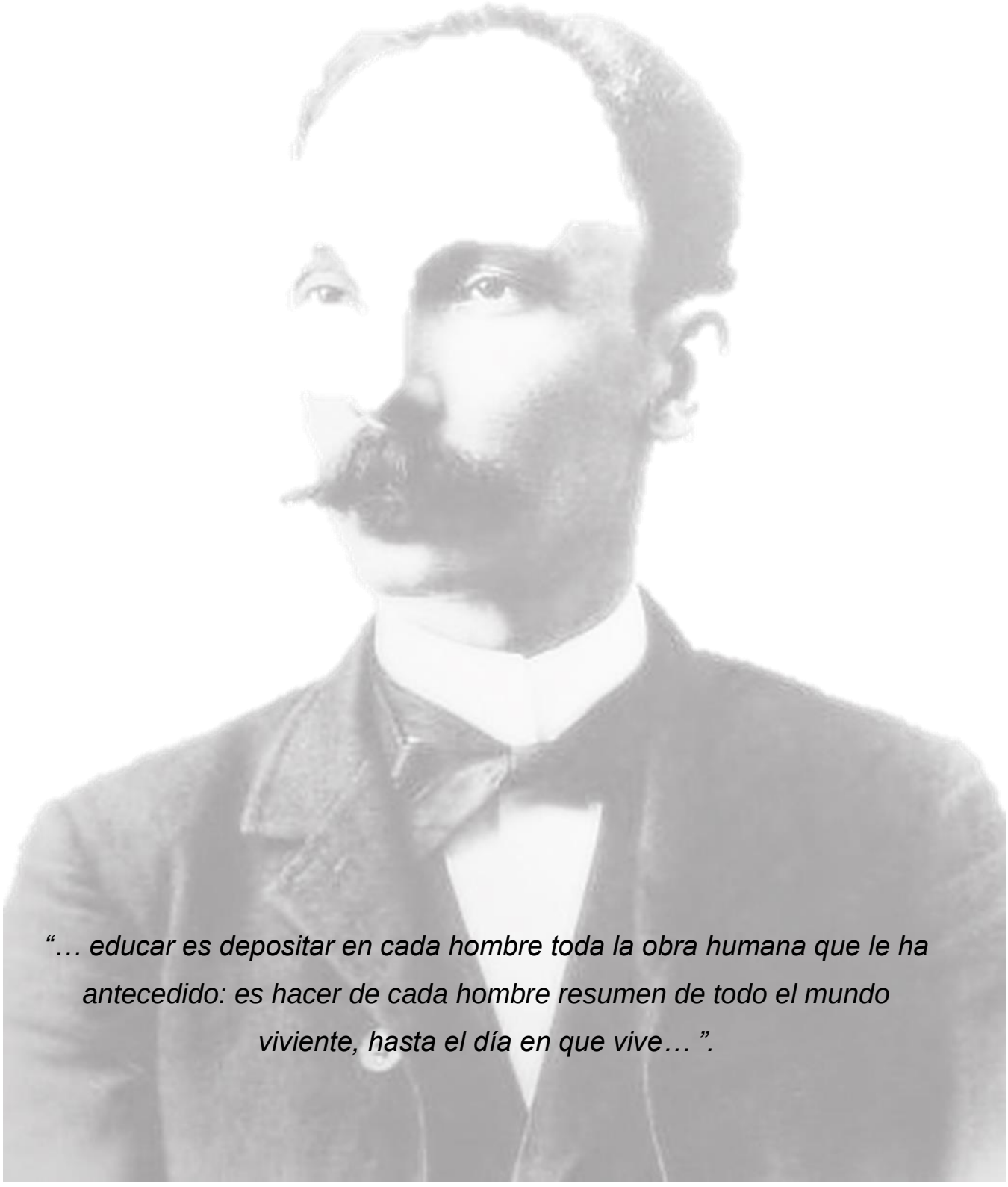
Ing. Isnel Leyva Herbella

Ing. Suleydis Suárez Serpa

Cotutora:

Ing. Diana Lázara Durán Dor

La Habana, Junio del 2014



“… educar es depositar en cada hombre toda la obra humana que le ha antecedido: es hacer de cada hombre resumen de todo el mundo viviente, hasta el día en que vive… ”.

Declaración de Autoría

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

Para que así conste firmo la presente a los____días del mes de_____del año_____.

Firma del autor *Alejandro Ramón Haro Cabrera*	*Firma del autor* *Isbel López González*
Firma del tutor *Ing. Isnel Leyva Herbella*	*Firma del tutor* *Ing. Suleydis Suárez Serpa*

Firma del cotutor
Ing. Diana Lázara Durán Dor

Datos de Contacto

Ing. Isnel Leyva Herbella

Graduado de ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas en el 2009. Posee la categoría docente de Instructor. Durante su trabajo como profesor ha impartido la asignatura Bases de Datos, Programación y cursos optativos del departamento. En la vinculación con la producción pertenece al departamento de Gestión Hospitalaria del Centro de Informática Médica (CESIM) y específicamente trabaja en el desarrollo del proyecto xaviaHIS donde se desempeña como jefe de desarrollo del Módulo de Farmacia.

Correo: iherbella@uci.cu

Ing. Suleydis Suárez Serpa

Instructor graduado en el año 2007 de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas. Ha impartido las asignaturas Matemática Discreta, Álgebra lineal y Metodología de la Investigación Científica. Se encuentra vinculado al desarrollo del Sistema de Información Hospitalaria xaviaHIS, que se desarrolla en el departamento Sistemas de Gestión Hospitalaria del Centro de Informática Médica (CESIM). Ha tutorado varios trabajos de diploma hasta el momento.

Correo: ssuarez@uci.cu

Ing. Diana Lázara Durán Dor

Ingeniero en Ciencias Informáticas, graduado en el año 2013 de la Universidad de las Ciencias Informáticas (UCI). Vinculado al departamento de Sistemas de Gestión Hospitalaria (GEHOS) del Centro de Informática Médica (CESIM), donde actualmente desempeña el rol de analista en el proyecto Sistema de Información Hospitalaria xaviaHIS

Correo: dlduran@uci.cu

Dedicatoria

De Alejandro

Dedico este trabajo de diploma a toda mi familia y en especial a mis padres Amarilis y Vidal que son la luz de mi vida y los encargados de que hoy en día me convierta en ingeniero. También se la quiero dedicar a mi padre biológico que siempre será mi guía a seguir,

De Isbel

Dedico este trabajo de diploma a las personas que desde el comienzo de esta aventura mantuvieron su confianza en mí, especialmente a mi mamá y mi papá que son mi razón de ser, ese estímulo me dio las fuerzas para convertirme hoy en ingeniero.

Agradecimientos

De Alejandro

Primero que todo agradecer a toda mi familia que desde un principio me apoyaron, a mis padres que son mi razón de ser, a mis hermanas que son lo más lindo que me han regalado mis padres, a mi abuela que siempre me apoyo.

A mis tutores que siempre me apoyaron en cada una de las tareas a desarrollar de todo corazón se los agradezco. A yoel que fue como otro tutor para mí.

A todos mis amigos, compañeros de aula, los del apartamento, los que por una razón u otra no están aquí conmigo, a los nuevos amigos que han entrado en mi vida y han formado parte de ella.

Agradecerle a Yariel, Tapia y Keili por estar siempre estuvieron en los momentos más difíciles de la carrera. A mi novia por ser siempre mi guía en estos últimos tiempos, de todo corazón se los agradezco.

Y agradecerle también a mi compañero de tesis y hermano Isbel que sin él no hubiese sido posible la realización de este trabajo.

De forma general quiero agradecerle a todos los que de una forma u otra colaboraron en mi formación como profesional y sobre todo a los que un dia pensaron que no podía cumplir esta meta.

De Isbel

Quiero agradecer a mis padres por convertir cada obstáculo que se ha presentado a lo largo de la vida en apoyo incondicional y depositar toda su confianza en mí para lograr hacer realidad el día de hoy lo que comenzó hace cinco años.

A toda mi familia por desear siempre lo mejor para mí, especialmente a los que me acogieron aquí en la Habana durante todo el tránsito de la carrera, además de agradecer a todos mis primos que a través de los años han sabido ser el ejemplo a seguir en el cual yo he moldeado mi carácter para conseguir sea cada día mejor persona.

A los tutores por facilitar la realización del trabajo de diploma, en especial a Isnel por ponerse en contacto con nosotros desde el principio.

A todos mis amigos por estar siempre conmigo, en especial a mi compañero de tesis el cual además fue durante todo este tiempo compañero de aventuras, de estudio para pruebas finales y extras, además de complementar el esfuerzo dedicado al éxito de este trabajo.

Resumen

Las funciones fundamentales de un centro de atención hospitalaria son la prevención, el diagnóstico y el tratamiento de enfermedades, sin embargo cada institución modela y define de manera diferente la forma en que se comportan los procesos médicos que se llevan a cabo en la atención al paciente y en la labor administrativa de la misma, para alcanzar mejor eficiencia. Este proceso es definido como gestión de las Reglas del Negocio, el cual permite controlar y conocer la manera de operar de una institución hospitalaria atendiendo a sus propias especificidades.

En un sistema informático se deben registrar esas Reglas del Negocio para poder consultarlas posteriormente, debido a que son dinámicas y pueden requerir modificaciones en su funcionamiento. La presente investigación tiene como objetivo el desarrollo de funcionalidades para el módulo de Configuración del Sistema de Información Hospitalaria del CESIM, las cuales le facilitarán al personal médico que interactúa con el mismo la gestión de dichas reglas sin necesidad de tener conocimientos avanzados en lenguajes de programación.

Para el desarrollo de las funcionalidades se realiza un estudio de los sistemas existentes que gestionan las Reglas del Negocio. Se obtienen los artefactos correspondientes al modelo de dominio, casos de uso, modelo de diseño, despliegue y diagrama de clases.

El desarrollo de la solución propuesta está guiado por el Proceso Unificado de Desarrollo y se basa en tecnologías libres, multiplataforma y en una arquitectura en capas implementando el patrón Modelo Vista Controlador, y utilizando Java como lenguaje de programación.

Palabras Claves: Reglas del Negocio, Sistema de Información Hospitalaria del CESIM.

Índice de Contenido

Índice de Figuras

Índica de Tablas

Introducción

En las últimas décadas es cada vez mayor la presencia de las Tecnologías de la Información y las Comunicaciones (TIC) en las actividades humanas, la constante evolución de las mismas ha propiciado que su empleo esté prácticamente generalizado. En estrecho vínculo con las TIC, la informática se ha manifestado en los diversos sectores sociales, aportando grandes beneficios en situaciones donde se hace necesario el almacenamiento y manipulación de considerables volúmenes de datos, una ejecución de cálculos matemáticos a grandes velocidades o un fácil y rápido acceso a determinada información.

Debido a que en las organizaciones hospitalarias existe la generación masiva de información médica, una de las aplicaciones sociales más frecuentes e importantes de las TIC desde hace varias décadas ha sido en la rama de la medicina. Cada día los procesos de registro, seguimiento y tratamiento del paciente deben mejorarse, innovarse y apoyarse en tecnologías para hacer más eficiente y eficaz las actividades rutinarias del hospital, centro de salud o clínica. No basta con tener datos e información, hay que procesarla, analizarla, interpretarla y utilizarla.

Aunque aún no es suficiente la infraestructura tecnológica que posee el país para responder a las demandas actuales en este sector; se han desarrollado soluciones para las instituciones de salud que tienen como objetivo informatizar y optimizar las actividades que se realizan en el proceso de atención al paciente, con el fin de incrementar su eficiencia; siendo una de estas, los Sistemas de Información Hospitalaria (HIS por sus siglas en inglés).

Los HIS son sistemas informáticos destinados a la gestión de la información hospitalaria de forma eficiente y robusta. Estos están diseñados con el objetivo de mejorar la calidad y la eficiencia del trabajo en los centros de atención médica. Los mismos se encargan de almacenar, procesar, recopilar, recuperar y comunicar los datos obtenidos en la atención al paciente y en la labor administrativa de estas instituciones. (1)

Sobreponiéndose a la falta de infraestructura informática, el Sistema Nacional de Salud (SNS) y el Ministerio de Salud Pública (MINSAP) han desarrollado proyectos para la informatización de sus servicios. La colaboración de vital importancia para alcanzar este objetivo ha sido la Universidad de las Ciencias Informáticas (UCI), la cual posee diversos centros de desarrollo de software, entre ellos el Centro de Informática Médica (CESIM), encargado del desarrollo de sistemas de gestión para el sector de la salud.

El Centro de Informática Médica de la Universidad de las Ciencias Informáticas desarrolla un Sistema de Información Hospitalaria, con el objetivo de gestionar los datos de los diferentes procesos de una institución de salud. Este sistema cuenta con diecisiete módulos que interconectan en la aplicación las distintas áreas de un centro asistencial tales como: Banco de Sangre, Epidemiología, Anatomía Patológica, Bloque Quirúrgico, Emergencia, Hospitalización, Consulta Externa, entre otros.

Cuando se decide desplegar el Sistema de Información Hospitalaria del CESIM en un nuevo hospital, centro de salud o clínica, se requiere como paso previo una clasificación de los diferentes centros ya que existe diversidad en cuanto a dimensión, tipología, complejidad de los servicios ofrecidos, entre otras. La estructura de un hospital está diseñada para cumplir las funciones de prevención, diagnóstico y tratamiento de enfermedades. Aun así, cada institución está compuesta por diferentes órganos en su forma de organización para alcanzar mejor eficiencia. Como resultado, existen diferentes tipos de hospitales atendiendo a sus propias especificidades.

Con el principio de facilitar la gestión, dentro de cada centro hospitalario también existen diferencias en cuanto a cómo se modelan, definen y controlan el comportamiento de las actividades rutinarias que se llevan a cabo en la atención al paciente, la labor administrativa y en la organización de la lógica del negocio de la entidad hospitalaria. Este comportamiento es conocido como regla del negocio, la cual permitirá representar y ejecutar la forma en que opera determinada institución.

Las Reglas del Negocio son condiciones, estándar o reglas que deben ser cumplidas y controladas por la organización durante el flujo del proceso. (2) Estas tienen carácter dinámico y variable en el tiempo. Por esta razón surge la necesidad de registrarlas para poder consultarlas posteriormente cuando necesiten ser modificadas. Esto es importante a la hora de modelar y definir el comportamiento de los procesos, pero más importante será a la hora de mantener una aplicación operativa dentro de muchos contextos diferentes.

La diferencia existente entre los centros de atención médica genera la necesidad de gestionar las Reglas del Negocio registradas en el Sistema de Información Hospitalaria del CESIM. De esta manera se permitirá cambiar las operaciones de las Reglas del Negocio a un contexto diferente en caso de ser necesario, evitando dificultades para desplegar el sistema en diferentes instituciones y permitiendo la adaptación del mismo a cualquier entorno.

Actualmente, los analistas y desarrolladores del sistema registran las Reglas del Negocio, a través del motor de reglas Drools, utilizando el lenguaje de reglas de Drools (DRL), para especificar las condiciones, acciones y funciones de las regla, las cuales son guardadas en archivos de texto con la extensión drl en un directorio físico que se encuentra desplegado en el servidor de aplicaciones. Luego, se ejecutan dentro del motor de reglas, y no sirven únicamente para representar la lógica de negocio, sino también para ejecutarla. De esta manera se codifica toda la lógica del negocio en reglas y se van aplicando en la toma de decisiones.

Debido al carácter dinámico y variable de las instituciones hospitalarias pueden ocurrir cambios en las operaciones que se realizan luego de haber superado la fase de despliegue del sistema provocando la necesidad de realizar modificaciones a una regla del negocio cuando se requiere un cambio en la estructura. La modificación se realiza expresando en lenguaje DRL nuevas especificaciones de las condiciones, acciones y funciones de las reglas. Debido a que la gestión de las Reglas del Negocio es de forma manual y no todos los usuarios pueden realizar la operación antes mencionada porque se debe contar con un conocimiento avanzado en el lenguaje de programación en el que fueron confeccionadas las mismas, el proceso de gestión pasa a ser engorroso y se necesita un largo periodo de tiempo para su confección.

El fichero de texto en el que son expresadas estas reglas está confeccionado en texto plano, es puramente técnico lo cual puede generar difícil entendimiento para el administrador del sistema que será el encargado de realizar la gestión de las Reglas del Negocio. Debido a que cada institución debe cumplir con el reglamento de seguridad informática se hace necesario poder gestionar las reglas sin necesidad de acceder al servidor de aplicaciones en pleno proceso de ejecución para no violar este reglamento.

Teniendo en cuenta los elementos mencionados anteriormente se plantea como **problema a resolver**: ¿Cómo facilitar la gestión de las Reglas del Negocio en el Sistema de Información Hospitalaria del Centro de Informática Médica?

Para dar solución a dicho problema se define como **objeto de estudio**: la estructura de las Reglas del Negocio en el motor de reglas Drools, enmarcándose en el **campo de acción**: la gestión de Reglas del Negocio del Sistema de Información Hospitalaria del Centro de Informática Médica.

Se propone como **objetivo general**: desarrollar las funcionalidades para la gestión de las Reglas del Negocio en lenguaje natural del Sistema de Información Hospitalaria del Centro de Informática Médica.

Para dar cumplimiento al objetivo anteriormente planteado se definen las siguientes **tareas de investigación**:

1. Describir la estructura de las Reglas del Negocio para el motor de reglas Drools.
2. Realizar el estudio del estado del arte de las herramientas de gestión de Reglas del Negocio en sistemas en producción.
3. Obtener mediante el Proceso Unificado de Desarrollo, los artefactos de trabajo de: Modelado de Negocio, Gestión de Requerimientos, Diseño e Implementación.
4. Asimilar la arquitectura definida por el departamento de Sistemas de Gestión Hospitalaria para el desarrollo de sus aplicaciones.
5. Implementar las funcionalidades que posibiliten la gestión de las Reglas del Negocio establecidas en el Sistema de Gestión Hospitalaria del CESIM.

Con el desarrollo de la herramienta para la gestión de las Reglas del Negocio se esperan obtener los siguientes beneficios:

1. Perfeccionar los procesos que se llevan a cabo en la gestión de las Reglas del Negocio del sistema de gestión hospitalaria del CESIM.
2. Disponer de una herramienta que satisfaga las necesidades reales y actuales de los profesionales de la salud en las actividades rutinarias del hospital, centro de salud o clínica, mejorando las condiciones de trabajo de los especialistas.
3. Permitir al usuario definir y mantener sus Reglas del Negocio, de forma que se puedan ir modificando manualmente en la toma de decisiones.

Los métodos teóricos utilizados para cumplir con las tareas a desarrollar son:

Análisis histórico-lógico: fue de gran importancia para elaborar la fundamentación teórica de la investigación, permitiendo estudiar lo más relevante en el plano teórico acerca la estructura de las Reglas del Negocio en el motor de reglas Drools, las metodologías, herramientas y tecnologías que se utilizan para el desarrollo de la aplicación y para el estudio del de software existentes a nivel internacional.

Analítico-sintético: este método fue utilizado en todo el proceso investigativo permitiendo descomponer todo el problema en varias partes que posibilitaron una mejor comprensión del mismo. Su empleo facilitó el análisis de la bibliografía encontrada referente al tema en cuestión y se sintetizan los aspectos más importantes para la investigación.

Inductivo-Deductivo: se utilizó para el planteamiento del objetivo y la extracción de las ideas fundamentales para la elaboración y fundamentación del trabajo de diploma.

Los métodos empíricos utilizados para obtener información sobre el objeto de estudio son:

Entrevista: a través de la realización de entrevistas a al jefe de desarrollo del módulo de farmacia del departamento de Gestión Hospitalaria del Centro de Informática Médica (CESIM) se recopiló toda la información necesaria para el desarrollo del trabajo con el objetivo de llegar a un acuerdo sobre lo que se quiere que tenga el producto.

Observación: se realizó un seguimiento sobre cómo se ejecutan las Reglas del Negocio en el Sistema de Información hospitalaria del Centro de Informática Médica para hacer un registro visual de los procesos que se llevan a cabo en la actualidad en el centro.

El documento se encuentra dividido en 4 capítulos, estructurado de la siguiente manera:

Capítulo 1: Fundamentación teórica de las soluciones informáticas relacionadas con el uso de las Reglas del Negocio en una institución hospitalaria: en este capítulo se exponen los conceptos relacionados al dominio del problema planteado, los cuales favorecerán la familiarización con el entorno en que se manifiesta la investigación. Se hace un estudio de los principales sistemas informáticos que gestionan reglas. Además son abordadas las metodologías, tecnologías y herramientas a utilizar.

Capítulo 2: Características de las funcionalidades para la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM: en este capítulo se presenta el modelo de dominio el cual contiene su descripción para un mayor entendimiento, conjuntamente con la especificación de los requerimientos funcionales, no funcionales y el Modelo de Casos de Usos de la solución propuesta.

Capítulo 3: Diseño de las funcionalidades para la gestión de las Reglas del Negocio del sistema de información hospitalaria del CESIM: se fundamenta la arquitectura empleada, así como las

estrategias de integración a tener en cuenta. Se realiza una descripción y análisis de la estructura de la solución que se propone para dar respuesta a la problemática planteada.

Capítulo 4: Implementación de las funcionalidades para la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM: se introduce el flujo de trabajo de implementación, partiendo de los resultados obtenidos en el diseño y se exponen aspectos referentes a la seguridad del sistema, las estrategias de codificación, así como la forma en que se tratarán los errores.

Capítulo 1. Fundamentación Teórica de las soluciones informáticas relacionadas con el uso de las Reglas del Negocio en una institución hospitalaria

En este capítulo se tratan los principales conceptos relacionados con la gestión de las Reglas del Negocio en el Sistema de Gestión Hospitalaria del CESIM, la descripción de otras herramientas para la gestión de las reglas, así como la selección de las herramientas, tecnologías, lenguajes y metodologías utilizadas para el desarrollo de la solución propuesta a partir de la arquitectura definida por el departamento.

1.1 Conceptos básicos relacionados con el dominio del problema

A menudo las Reglas del Negocio están focalizadas en el control, en la forma de realizar los cálculos, como facturar a los clientes, plazos, formas de pago, excepciones a las formas de pago, otras permiten establecer las políticas y así se tienen en cualquier actividad del negocio que requiera que el usuario actúe de una forma pre-establecida. También, pueden estar formalmente definidas en manuales de procedimiento, contratos o acuerdos, o bien pueden existir como conocimientos o experiencias. Las Reglas del Negocio son dinámicas, están sujetas a cambios en el tiempo y pueden encontrarse en todo tipo de aplicaciones.

Una "***Regla del Negocio***", literalmente, es lo que se usa para operar un negocio. Son las guías que determinan cómo se realizan las operaciones. Sin reglas se estaría en una situación en la que cada decisión se resuelve en el momento. Una Regla del Negocio define o limita un aspecto del negocio con el objetivo de establecer una estructura o un grado de influencia que condiciona el comportamiento de los actores del negocio. (3)

En el momento de diseñar el funcionamiento de la aplicación se deben registrar esas Reglas del Negocio para poder consultarlas posteriormente. Estas reglas bien documentadas darán la información necesaria de cómo y por qué tiene que funcionar una aplicación, esto es importante a la hora de diseñar y programar pero más importante será a la hora de mantener esa aplicación. Sin reglas se estaría en una situación en la que cada decisión se resuelve en el momento, eligiendo alternativas caso a cado o ad-hoc, el mismo, se utiliza para indicar que un determinado acontecimiento es temporal y es destinado a ese propósito específico y que define la creación de algo provisional, que sólo va a servir para un determinado propósito. Este modo de operar es muy lento, costoso y puede generar resultados inconsistentes. (4)

Al no estar presente el concepto de Reglas del Negocio en un proyecto de implementación tradicional se genera una dispersión de la información correspondiente a las Reglas del Negocio, y una dilución de la responsabilidad de su mantenimiento y documentación.

La dispersión de la información se genera porque al no considerar el concepto en cuestión, las reglas se tratan en distintos objetos como son: herramientas para la parametrización de un sistema, programas, estructuras de datos, reglamentos, manuales de procedimientos, conocimiento de los usuarios, entre otros. A su vez la dilución de las responsabilidades se origina por el proceso de generación; que no es sistemático en cuanto a la recopilación, identificación de los responsables y a la forma de documentar. En resumen, se mezclan responsabilidades propias del Negocio con las de Informática.

Las Reglas del Negocio se ejecutan dentro de un motor de reglas, este en su forma más simple, está constituido por tres elementos: un conjunto de reglas, la base de conocimientos (conocida como área de trabajo) y el procesador de reglas. El motor utilizará la base de conocimientos para decidir que reglas deben activarse. El criterio de decisión para la utilización de un motor de reglas podría ser:

- La complejidad moderada o alta de las Reglas del Negocio.
- Las Reglas del Negocio no son estáticas y se prevé que cambien continuamente.
- Como herramienta de simulación y pruebas de concepto.
- Como parte de la metodología para recopilar, documentar y mantener las reglas.

El "***Motor de Reglas***", es un sistema que se configura para dar servicio a las necesidades de negocio a través de la definición de objetos y Reglas del Negocio, el software se rige por flujos que derivan responsabilidades a los distintos cargos de la entidad repartiendo así el trabajo equitativamente y cuantitativamente, cuándo, quién y dónde tiene que desempeñar la tarea asignada. (5)

1.2 Sistemas existentes vinculados a la gestión de las Reglas del Negocio en una institución hospitalaria

Teniendo en cuenta la importancia que representa el proceso de gestión de las Reglas del Negocio para aplicaciones informáticas, es fundamental destacar que a nivel internacional existen diferentes sistemas que están compuestos por soluciones enfocadas a esta gestión de las reglas y son utilizados por varias empresas. El estudio de estas soluciones, aporta una guía para la presente investigación, ya

que permite establecer puntos de comparación entre las existentes y la que se quiere implementar. Entre estas se encuentran:

1.2.1 Drools Guvnor

Drools Guvnor es un repositorio centralizado para Bases de Conocimiento Drools, con interfaces gráficas de usuario basadas en web, editores y herramientas para ayudar en la gestión de un gran número de reglas. Drools le permite crear bases de conocimiento ejecutables. El componente de repositorio es donde se puede almacenar versiones de reglas, modelos, funciones y procesos, que están relacionadas con estas bases de conocimiento. El acceso es controlado, y es posible bloquear el acceso y restringir características para que los expertos del dominio (no programadores) puedan ver y editar las reglas sin estar expuestos a todas las funciones a la vez. (6)

1.2.2 RuleXpress

Es independiente de la tecnología de motores de reglas y ha sido diseñado para usuarios y analistas de negocio no tecnológicos. RuleXpress les permite documentar sus reglas utilizando sus propias palabras. Su uso evita tener que rehacer el trabajo: el conocimiento y el saber hacer se mantienen en una fuente única, manejable, e independiente de la implementación.

RuleXpress es la herramienta que proporciona las reglas adecuadas en un formato independiente de la tecnología. De esta manera, permite a las organizaciones que aplican sistemas de gestión de Reglas del Negocio y de la toma de decisiones, liderar ellas mismas la gestión de las reglas de las mismas. A diferencia de los sistemas de gestión de Reglas del Negocio centrados en la ejecución, RuleXpress las gestiona tal como las contempla el negocio, y no sólo de la manera en que las tecnologías de la información las necesitan.

Algunas de las principales ventajas que brinda RuleXpress son:

Proporciona una herramienta para capturar la lógica de decisión del negocio como requisito para sus proyectos de tecnologías de la información. Dado que gestiona el vocabulario (la terminología) que forma la base de todas sus reglas, asegurando que todos los directivos utilizan el lenguaje de la misma forma, define un modelo para los conceptos que sustentan su negocio, garantizando que todos los directivos trabajan a partir de las mismas definiciones.

RuleXpress ofrece:

- Una manera de gestionar sus reglas en su propio lenguaje, utilizando la terminología de negocio, no la terminología de las tecnologías de la información.
- Una interfaz de usuario (que no requiere conocimientos de leguajes de programación) de "*arrastrar y soltar*", para gestionar la terminología y las reglas.
- Unos controles de calidad integrados y configurables para garantizar que los términos y las reglas se definan de forma completa y precisa.
- Capacidad para distinguir entre las perspectivas y el vocabulario de negocio y el de las tecnologías de la información, permitiendo que ambas comunidades hablen de las mismas reglas desde diferentes puntos de vista.
- Amplias funciones de generación de informes, extensibilidad y fácil integración con las herramientas de terceras partes para garantizar una adopción sencilla y generalizada. (7)

1.2.3 Repcon Rules Server

Repcon Rules Server es un sistema experto que facilita la inferencia de decisiones y tiene un lenguaje propio de las reglas que operan sobre los procesos. Permite diferentes tipos de funcionalidades como son: modelización de procesos de negocio, además de posible integración con sistemas y lenguajes de programación heterogéneos. Este sistema es de fácil administración por usuarios que no son informáticos debido a que permite la generación automática de documentación, así como cálculos técnicos y de precios.

Repcon Rules Server se aplica en entornos de negocio cambiantes, dónde es preciso que las reglas que se aplican sobre los sistemas estén separadas del código de las aplicaciones de gestión. Este sistema también puede operar en aplicaciones informáticas con problemas de: tarificación, cálculos de incentivos y aprobación de riesgo, además de sectores de actividad: aseguradoras, entidades financieras, distribución, administración pública entre otras. (8) Estos beneficios permitirán a los usuarios de las áreas de negocio definir y mantener las reglas independientes del código de programación atribuyendo rapidez en la adaptación de las aplicaciones a los cambios en el negocio.

1.2.4 BRMS (Business Rules Management System)

Es un sistema que básicamente permite al usuario definir sus Reglas del Negocio, de forma que se van aplicando automáticamente en la toma de decisiones. Con un sistema BRMS, los analistas del negocio determinan y escriben la lógica del mismo, todo lo que necesitan es escribir una regla. Normalmente, una regla de negocio se expresa, con un "*Si*", es decir, si existen tales condiciones, "*Entonces*" deben

ejecutarse tales opciones y/o acciones o bien, en caso contrario, ejecutarse tales otras. La relación entre esas condiciones y las acciones a llevar a cabo crea una regla de negocio, la cual puede o no estar relacionada también con otras reglas. Un sistema BRMS permite que los analistas del negocio puedan ver y entender las reglas sin tener que depender del departamento de informática para realizar las modificaciones. (9)

Luego de un estudio de las principales herramientas existentes que permiten gestionar las Reglas del Negocio se concluye que las mismas no pueden ser escogidas como posible solución a la problemática planteada. El problema está dado por la necesidad de incorporarle al módulo de Configuración del Sistema de Información Hospitalaria del CESIM funcionalidades con características específicas para el mismo, desarrollado sobre la concepción arquitectónica, tecnologías, herramientas y lenguajes de programación definidos por el centro. Sin embargo este análisis permitió identificar ventajas y beneficios que serán incluidos en la solución.

1.3 Arquitectura

La Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema. Una Arquitectura de Software, también denominada Arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. No es más que la estructura de un sistema informático, por lo que define los principales componentes del mismo y sus relaciones.

En la realización de este sistema se define a utilizar la arquitectura Modelo Vista Controlador (MVC).

La arquitectura de Modelo Vista Controlador (MVC): es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El estilo de llamada y retorno MVC, se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista. (10)

1.4 Metodologías

1.4.1 Proceso unificado de desarrollo (RUP)

RUP (Proceso Unificado de Desarrollo) es un proceso para el desarrollo de un proyecto de software que define claramente quién, cómo, cuándo y qué debe hacerse en el proyecto. Permite el desarrollo de software a gran escala, mediante un proceso continuo de pruebas y retroalimentación, garantizando

el cumplimiento de ciertos estándares de calidad. Este proceso de desarrollo de software lo conforman el conjunto de actividades necesarias para trasformar los requisitos funcionales de un usuario en un producto de software. (11)

Los que sustentan el proceso de desarrollo de software son: el proyecto, las personas, el producto y el proceso, existe una estrecha relación entre ellas. Es conocido como las cuatro P en el desarrollo del software. RUP define para cada etapa: el flujo de trabajo, los trabajadores que intervienen, las actividades que realizan y los artefactos que se necesitan o producen. Su meta es asegurar la producción de software con la más alta calidad, que cumpla con las necesidades de los usuarios dentro del cronograma planeado y la inversión prevista.

En la actualidad constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Propone cómo deben ser ejecutadas las actividades para la obtención de los productos típicos de trabajo y la forma de documentar todas las acciones que se llevan a cabo en las mismas, soportando el ciclo completo de desarrollo de la aplicación.

1.4.2 Lenguaje Unificado de Modelado (UML)

UML es el lenguaje de modelado más conocido y utilizado en la actualidad. Es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo. Las estructuras que soporta tienen sus fundamentos en las tecnologías orientadas a objetos, tales como clases, componentes y nodos. Representa el comportamiento del sistema a través de casos de uso, diagramas de secuencia y de colaboración Los estereotipos son el mecanismo de extensibilidad incorporado más utilizado dentro de UML. Un estereotipo representa una distinción de uso. Puede ser aplicado a cualquier elemento de modelado, incluyendo clases, paquetes, relaciones de herencia, entre otros. (12)

Uno de los objetivos de este modelado visual es que sea independiente del lenguaje de implementación, de tal forma que los diseños realizados se puedan implementar en cualquier lenguaje de programación que soporte las posibilidades de UML (principalmente lenguajes orientados a objetos). Con este método formal de modelado se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa.

1.5 Herramientas

1.5.1 Eclipse

Eclipse es una plataforma de desarrollo de código abierto basada en Java. Por si misma, es simplemente un marco de trabajo y un conjunto de servicios para la construcción del entorno de desarrollo de los componentes de entrada. Afortunadamente, Eclipse tiene un conjunto de complementos, incluidas las Herramientas de Desarrollo de Java (JDT). Mientras que la mayoría de los usuarios están felices de usar Eclipse como un IDE de Java, sus ambiciones no se detienen ahí. También incluye el Entorno de Desarrollo de Complementos (PDE), que es de interés principalmente para los desarrolladores que quieren extender Eclipse, dado que les permite construir herramientas que se integran sin dificultades con el entorno de la plataforma. Dado que todo en Eclipse es un complemento, los desarrolladores de herramientas tienen un campo de juego de nivel para ofrecer extensiones a Eclipse y para proporcionar un entorno de desarrollo integrado y unificado para los usuarios. (13)

1.5.2 Jboss Seam

Jboss Seam es un framework que integra la capa de presentación Java Server Faces (JSF) con la capa de negocios y persistencia Enterprise Java Bean (EJB). Con Seam basta agregar anotaciones propias de éste a los objetos Entidad y Session de EJB, logrando con esto escribir menos código Java y XML. Otra característica importante es que puedes hacer validaciones en los Plain Object Java (POJO) como además manejar directamente la lógica de la aplicación y de negocios desde tus sesiones beans. También se integra perfectamente con otros frameworks como: RichFaces, ICEFaces (soportan Ajax) MyFaces, Hibernate y Spring. (14)

1.5.3 Jboss Tools

Conjunto de plug-ins diseñados para el entorno integrado de desarrollo Eclipse. Está constituido por varios módulos: RichFaces VE, Seam Tools, Hibernate Tools, Jboss AS Tools, Drools IDE, jBPM Tools y JBossWS Tools. (15)

Los módulos de JBoss Tools son:

RichFaces VE: el editor visual aportado por Exadel proporciona el apoyo para la edición visual de páginas HTML, JSF, JSP y Facelets. También incluye soporte visual para las librerías de componentes JSF incluyendo JBoss RichFaces.

Seam Tools: incluye soporte para Seam-gen, RichFaces VE.

Hibernate Tools: soporta el mapeo de archivos, anotaciones y JPA con la ingeniería inversa, completamiento de código, asistentes de proyecto, refactorización, ejecución interactiva de HQL/JPA-QL/Criteria.

JBoss AS Tools: fácil de iniciar, detener y debuguear al estar integrado con Eclipse. También incluye funciones para el despliegue eficaz de cualquier tipo de proyecto en el IDE.

Drools IDE: editor de ficheros de reglas, debugueo e inspección de reglas.

JBPM Tools: edición del flujo de trabajo del jBPM, motor de procesos BPM.

JBossWS Tools: desarrollo, invocación, inspección y pruebas de servicios web sobre http con la adición y soporte de características JBossWS.

1.5.4 Visual Paradigm para UML 8.0

Visual Paradigm para Lenguaje Unificado de Modelado (UML por sus siglas en inglés): es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. (16)

1.6 Tecnologías y Lenguajes

1.6.1 Java

Otra característica que distingue al Sistema de Información Hospitalaria del CESIM es que debe estar libre del costo relacionado con patentes de software, asociadas al servidor de aplicaciones, al sistema operativo huésped u otras herramientas o tecnologías utilizadas para su desarrollo. Para lograr este objetivo se propone el uso de un lenguaje de programación multiplataforma, como es el caso de Java.

Java es un lenguaje de programación con el que se puede realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras. (17)

En el mundo la tecnología Java se usa en los principales sectores de la industria informática y está presente en un gran número de dispositivos, equipos y redes. Este lenguaje trabaja con sus datos

como objetos y con interfaces a esos objetos y soporta las características propias del paradigma orientado a objetos como son: abstracción, encapsulación, herencia y polimorfismo.

1.6.2 Java Server Faces (JSF)

Es un framework de desarrollo basado en el patrón MVC (Modelo Vista Controlador) para aplicaciones java de tipo web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java Enterprise Edition (Java EE por sus siglas en inglés). Permite además crear interfaces de usuario (UI por sus siglas en inglés) para aplicaciones web, mediante componentes reutilizables. Permite el manejo de estados y eventos, así como la asociación entre los datos de la interfaz y los datos de la aplicación web. (18)

1.6.3 RichFaces

Es una biblioteca de componentes para JSF, constituye un avanzado framework para la integración de Ajax con facilidad en la capacidad de desarrollo de aplicaciones de negocio. Provee facilidades de validación y conversión de los datos proporcionados por el usuario y administración avanzada de recursos como imágenes, código java script y hojas de estilo en cascada (CSS).Se integra completamente dentro del ciclo de vida JSF. Permite crear interfaces de usuario modernas de manera eficiente y rápida, basadas en componentes listos para usar, altamente configurables en cuanto a temas y esquemas de colores predefinidos por el propio framework o desarrollados a conveniencia, lo que mejora la experiencia de usuario. (19)

1.6.4 Ajax4JSF

Ajax4jsf es una librería open source 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, control de cualquier evento de usuario, entre otras funcionalidades. En definitiva Ajax4jsf permite dotar a nuestra aplicación JSF de contenido mucho más profesional con muy poco esfuerzo. (20)

1.6.5 Hibernate

Framework que provee herramientas de mapeo objeto/relacional y permite reducir significativamente el tiempo de desarrollo. Facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos eXtensible Markup Language

(XML por sus siglas en inglés) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible. (21)

1.6.6 Enterprise Java Bean (EJB)

Enterprise JavaBeans (EJB) es una arquitectura de componentes de servidor que simplifica el proceso de construcción de aplicaciones de componentes empresariales distribuidos en Java. Con su utilización es posible escribir aplicaciones escalables, fiables y seguras sin escribir código de infraestructura. La existencia de infraestructura permite un desarrollo más rápido de la parte servidora. Dado que son componentes, permiten desarrollar aplicaciones portables entre distintas plataformas (son Java) y servidores de aplicaciones (especificación estándar). (22)

Un EJB es un componente software que se ejecuta del lado del servidor en una aplicación multicapa. Los clientes del EJB acceden a él por medio de una interfaz que esconde los detalles de implementación del componente. Esta interfaz debe cumplir la especificación EJB. La especificación fuerza la presencia de unos determinados métodos que permitirán al contenedor de EJBs manejar los componentes y su ciclo de vida.

1.6.7 Java Persistence API (JPA)

Java Persistence API (JPA) es un framework para el trabajo con bases de datos relacionales, compuestas por tres partes:

- El paquete: Javax.persistence.
- El lenguaje de consultas de Persistence.
- Los metadatos para los objetos y sus relaciones. (23)

JPA es un framework de persistencia, que se abstrae de las bases de datos y brinda un estándar para persistir los datos en Java. JPA viene a solucionar el vacío que hay entre utilizar objetos y persistirlos en una Base de Datos (DB) relacional. Este framework mapea automáticamente las clases en la base de datos de manera transparente y utilizando un estándar, lo cual entre otras cosas permite poder migrar de motor cuando se desee, y poder compartir código o trabajar en equipo sin ningún problema.

1.6.8 Facelets

Java Server Facelets es un framework para plantillas (templates) centrado en la tecnología JSF, por lo cual se integran de manera muy fácil. Provee templating, lo cual implica reutilización de código, simplificación de desarrollo y facilidad en el mantenimiento de grandes aplicaciones. El uso de Facelets reduce el tiempo y esfuerzo que se necesita para el desarrollo y despliegue de aplicaciones. (24)

1.6.9 XHTML

XHTML o Lenguaje Extensible de Marcado de Hipertexto es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web. XHTML es la versión XML de HTML, fue desarrollado con el objetivo de avanzar en el proyecto del World Wide Web Consortium de lograr una web semántica, donde la información, y la forma de presentarla estuvieran claramente separadas, de forma que, se centra en transmitir la información que contiene un documento, dejando para hojas de estilo su aspecto y diseño. (25)

1.6.10 Cascading Style Sheets (CSS)

Hojas de Estilo en Cascada (Cascading Style Sheets), es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. CSS se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación. Los Estilos definen la forma de mostrar los elementos HTML y XML. CSS permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas al mismo tiempo. (26)

1.6.11 JavaScript

JavaScript es un lenguaje de programación utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de una página web. Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación del lado del cliente más utilizado. Con JavaScript se puede crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador. (27)

1.6.12 Drools

Es un motor de reglas basado en una adaptación orientada a objetos, conocido como sistema de reglas de producción, usando una implementación avanzada del algoritmo Rete. Drools surgen ante la

necesidad de centralizar y gestionar la lógica de negocio. Para ello, se codifica la lógica de las operaciones en forma de Reglas del Negocio, que sirven para tomar decisiones dentro de un contexto. Estas reglas se ejecutan dentro del motor de reglas, y no sirven únicamente para representar la lógica de negocio, sino también para ejecutarla. Además, el hecho de que la lógica se encuentre codificada en una regla, hace más fácil su comprensión que cuando se encuentra en el código de una aplicación, sobre todo para el personal no técnico. También provee una forma de acceder a los objetos del sistema y por ende se puede modificar comportamiento en tiempo de ejecución, sin tener que compilar o desplegar la aplicación.

En la mayoría de las empresas la lógica de negocio se encuentra dispersa por diferentes sitios tales como: en el código de las aplicaciones, hojas de cálculo, o en las mentes de los expertos. Este hecho hace que, a la mayoría de las personas de la organización les sea complejo consultar y comprender las reglas que constituyen la base del negocio.

Para especificar las reglas Drools utiliza el lenguaje de reglas de drools (DRL) para especificar las condiciones, acciones y funciones de las mismas, las cuales se puede expresar con distintos lenguajes, como Java y MVEL, entonces las reglas son guardadas en archivos de texto con la extensión drl. (28)

A continuación se presenta un ejemplo de como se expresan las las condiciones en el lenguaje de reglas de drools:

```
rule "entidad1"

    when
        IntervaloValidezCronicismo(idEntidad=="1")
        $med: IntervaloValidezCronicismo()
    then
        $med.setCantDiasAntes(2);
        $med.setCantDiasDespues(5);
end

rule "entidad2"

    when
        IntervaloValidezCronicismo(idEntidad=="2")
        $med: IntervaloValidezCronicismo()
    then
        $med.setCantDiasAntes(2);
        $med.setCantDiasDespues(5);
end
```

1.6.13 Reflection

Reflection es un componente de la API Java que permite examinar o modificar el comportamiento de las aplicaciones que se ejecutan en la máquina virtual de java. Se usa para instanciar clases e invocar métodos usando sus nombres, además de obtener y usar clases, interfaces, métodos, campos, y constructores en tiempo de ejecución un concepto que permite la programación dinámica. Mediantes el uso de esta herramienta se puede obtener una diversidad de características de una determinada clase de la cual se desea obtener la información necesaria para dar solución a un determinado problema. (29)

Conclusiones

A través un estudio detallado de los conceptos básicos asociados al dominio de la problemática, como son: Reglas del Negocio y Motor de Reglas, este capítulo ha facilitado la comprensión de la investigación que se realiza. También, el estudio de los principales sistemas existentes, se utilizó como apoyo para detectar las principales funcionalidades del sistema a desarrollar y permitió identificar ventajas e incorporar buenas prácticas asociadas a sus características. Además se justifica, la elección

de los lenguajes de programación, las tecnologías y las herramientas a utilizar, analizando cada una de ellas por las características y ventajas que aportan facilidad para el desarrollo de la aplicación.

Capítulo 2. Características de las funcionalidades para la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM

En este capítulo se tiene como objetivo abordar los diferentes elementos que brindan la base teórica y conceptual para el desarrollo de las funcionalidades. Debido a que no existe una definición del negocio se desarrolla un Modelo de Dominio, donde se muestran las definiciones asociadas a la solución. Además, se presentan los requisitos funcionales agrupados en los diagramas de cada caso de uso correspondientes.

2.1 Modelo de Dominio

El modelo de dominio es una representación visual de los conceptos u objetos que se manejan en el dominio de la solución propuesta. Los objetos o conceptos incluidos en este no describen clases u objetos de software, sino entidades o conceptos del mundo real que están asociados al problema en cuestión. Cuando se realiza la programación orientada a objetos, el funcionamiento interno del software va a imitar en alguna medida a la realidad, por lo que el mapa de conceptos del modelo de dominio constituye una primera versión para el desarrollo de las funcionalidades. (30)

2.2 Conceptos fundamentales del dominio

Con el objetivo de una mejor comprensión del diagrama del modelo de dominio, a continuación se dará una breve descripción de los conceptos encontrados en el ámbito del problema.

Reglas: regla que el usuario desea modificar.

Operaciones: son las acciones que el usuario realizará.

Módulos: son las áreas en que está estructurada una institución hospitalaria.

Usuario: profesional de la salud que interactúa con el sistema durante la ejecución de las funcionalidades.

DRL: lenguaje en el que son descritas las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM.

Drools: motor que interpreta Reglas del Negocio.

2.2.1 Diagrama del Modelo de Dominio

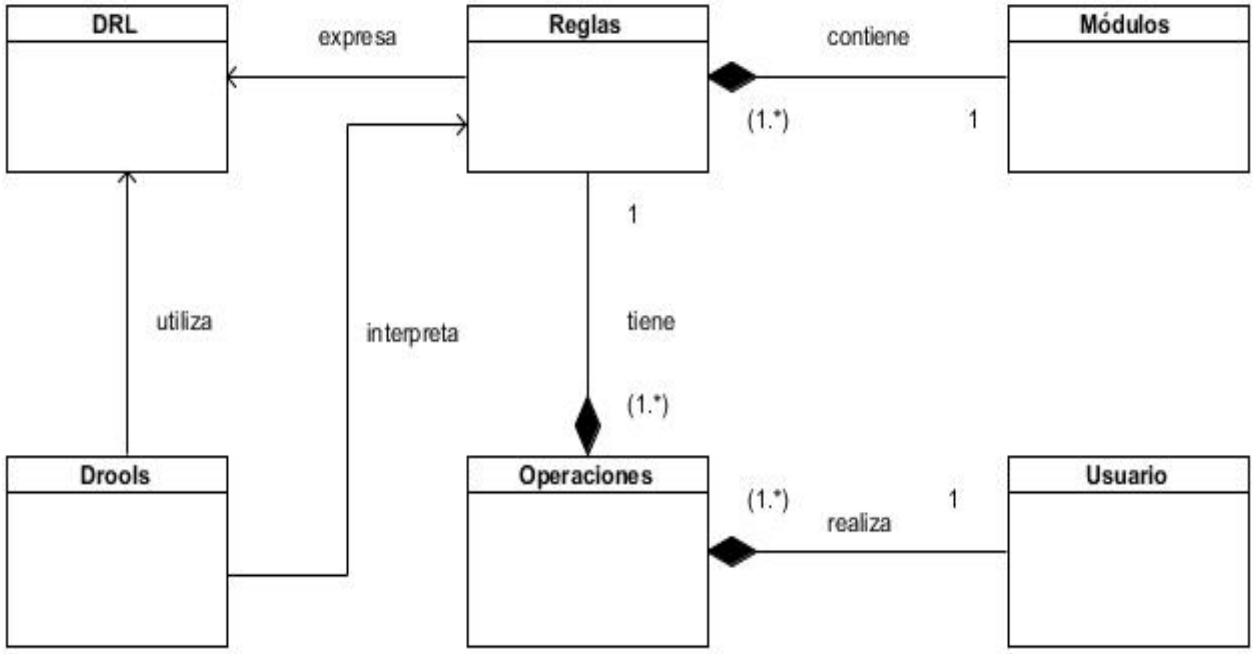


Figura 1 Diagrama de Modelo de Dominio

2.3 Especificación de los requerimientos de software

Un requerimiento es una condición que debe tener un sistema o un componente de este para satisfacer un contrato, norma, especificación u otro documento formal, facilitando el entendimiento entre clientes y desarrolladores. (31)

2.3.1 Requisitos funcionales del sistema

Los requisitos funcionales especifican acciones que debe poder realizar un software, sin tener en cuenta las restricciones físicas, además definen su comportamiento de entrada y salida. Seguidamente se exponen los requisitos funcionales, por los cuales se rige el desarrollo de la aplicación. (32)

Funcionalidad	Descripción
RF1 Configurar Regla	Permite listar todas las Reglas del Negocio.
RF3 Ver Regla	Permite observar una Regla del Negocio específica.
RF3 Modificar Regla	Permite al usuario modificar la Regla del Negocio escogida.

Tabla 1 Requisitos funcionales del sistema

2.3.2 Requisitos no funcionales del sistema

Los requisitos no funcionales describen aspectos del comportamiento de un sistema, capturando las propiedades y restricciones bajo las cuales deben operar. Son requerimientos que no se refieren específicamente a la funcionalidad de una aplicación. Se imponen restricciones sobre el producto que se está desarrollando y que especifican restricciones externas que el producto debe cumplir. (32)

2.3.2.1 Requerimientos de usabilidad

El sistema estará diseñado de manera que los usuarios adquieran las habilidades necesarias para explotarlo en un tiempo reducido. Las funcionalidades deben brindar, a través de una barra de navegación, un acceso fácil y rápido a todas las opciones de la aplicación. El significado de los íconos y los textos debe ser claro para la persona que interactúe con la aplicación. Las opciones que se proveen deben ser comprensibles, sin explicaciones excesivas sobre su uso, y deben admitir flujos alternativos, como cancelar la operación.

2.3.2.2 Restricciones de diseño

El sistema estará dividido en las siguientes capas:

2.3.2.2.1 Capas físicas

Cliente: computadora con cualquier tecnología o sistema operativo Firefox 4.x o superior y que cuente con el plugin de Flash Player 10.x o superior.

Servidor de Aplicaciones: servidor con cualquier tecnología o sistema operativo que soporte el Java Runtime Environment (JRE) 1.6.0_24 o superior y al JBoss AS 4.2.2.GA.

2.3.2.2.2 Capas lógicas

Presentación: contiene todas las vistas y la lógica de la presentación. El flujo web se maneja de forma declarativa y basándose en definiciones de procesos del negocio.

Negocio: mantiene el estado de las conversaciones y procesos del negocio que concurrentemente pueden estar siendo ejecutados por cada usuario. En los casos de que algún objeto del negocio tenga una interfaz externa, siendo accesible la misma desde sistemas legados o directamente del cliente, se garantiza la seguridad a nivel de objeto y métodos.

Acceso a Datos: contiene las entidades y los objetos de acceso a datos correspondientes a las mismas. El acceso a datos está basado en el estándar JPA y particularmente en la implementación del motor de persistencia Hibernate.

2.3.2.3 Interfaz

2.3.2.3.1 Interfaces de usuario

Las ventanas del sistema contendrán claro y bien estructurados los datos, además de permitir la interpretación correcta de la información.

La interfaz contará con teclas de función y menús desplegables que faciliten y aceleren su utilización.

La entrada de datos incorrecta será detectada claramente e informada al usuario.

Todos los textos y mensajes en pantalla aparecerán en el idioma de preferencia del usuario siempre que este sea soportado por la aplicación.

El diseño de la interfaz del sistema responderá a la ejecución de acciones de una manera rápida, minimizando los pasos a dar en cada proceso.

2.3.2.3.2 Interfaces hardware

Clientes:

Instalación de computadoras personales (computadoras personales con al menos 512 Mb de RAM).

Servidores:

Instalación de los terminal server (terminal server con al menos 16 Gb de RAM y 1 Tb de HDD).

2.3.2.4 Rendimiento

Los procesos generados por las nuevas funcionalidades en el momento de su ejecución deben ser lo suficientemente óptimo en cuanto al uso de memoria física y necesidad de procesamiento.

2.3.2.5 Software

Las funcionalidades estarán integradas al Sistema de Información Hospitalaria del CESIM, dicha aplicación debe poder ser desplegada en sistemas operativos Windows y Linux, utilizando la plataforma Java (Máquina Virtual de Java – Java Enterprise Edition), el servidor de aplicaciones JBoss AS. Los usuarios deberán disponer de un navegador web, este puede ser Firefox 3.6, Google Chrome 14 o versiones superiores de ellos y deben tener habilitado JavaScript.

2.3.2.6 Seguridad

Las funcionalidades deben mantener seguridad y control a nivel de usuarios, garantizando el acceso de los mismos sólo a los niveles establecidos de acuerdo a la función o rol que desempeñan. Las contraseñas podrán cambiarse únicamente por el propio usuario o por el administrador del sistema.

Toda entrada de información estará validada, de forma que la introducción de datos incorrectos se mostrará al usuario especificándole el tipo de error.

2.4 Modelo de casos de uso del sistema

El modelo de casos de uso del sistema documenta el comportamiento del sistema desde el punto de vista del usuario, permitiendo representar las funciones que se desean en el sistema, el entorno del sistema (actores), y las relaciones entre ellos. Aunque la parte más visible de dicho modelo son los diagramas de casos de uso, suele ir acompañado de una especificación textual de cada uno de estos casos.

Los actores del sistema no forman parte del mismo, sino que representan elementos que interactúan con él. Estos elementos son nombrados roles que pueden ser desempeñados por una o varias personas, un equipo o un sistema automatizado. Un actor puede introducir o recibir información del sistema. (33)

2.4.1 Definición de los actores del sistema

Actor	Descripción
Usuario	Se encarga de gestionar las Reglas del Negocio y es el que interactúa con el sistema durante la ejecución de las funcionalidades.

Tabla 2 Definición de los actores del sistema.

2.4.2 Diagrama de Actores

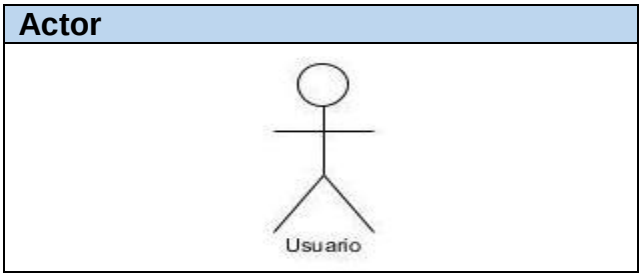


Figura 2 Diagrama de Actores del sistema

2.4.3 Diagrama de Casos de Uso del Sistema

Casos de Uso del Sistema

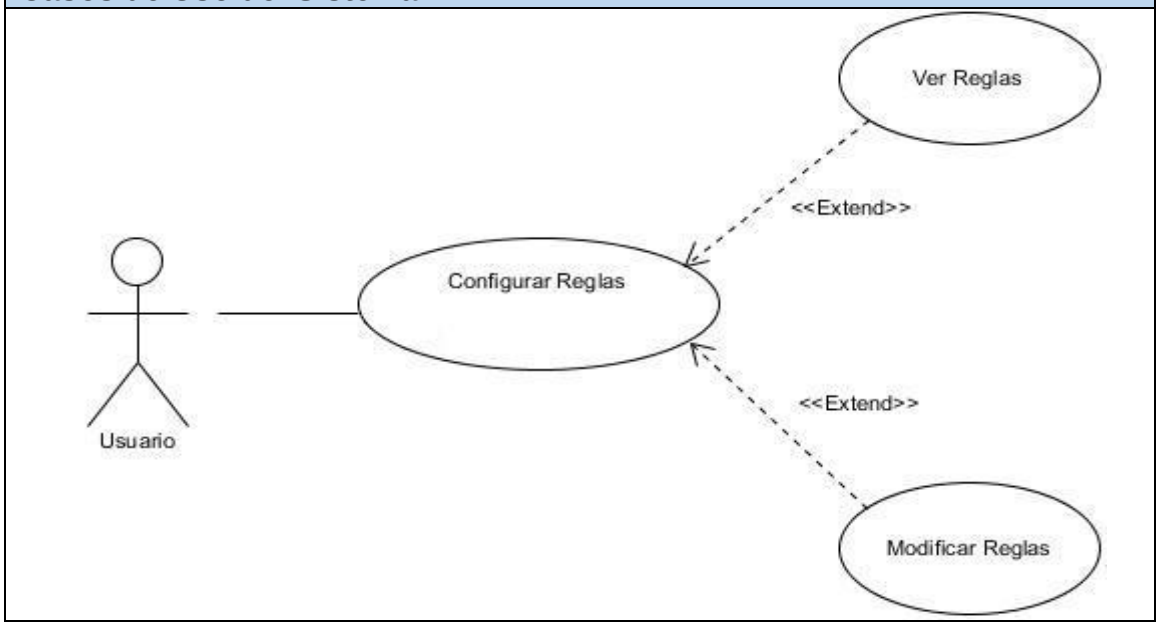


Figura 3 Diagrama de caso de uso del sistema

2.4.4 Descripción Textual de los Casos de Uso

Caso de Uso	**Configurar Reglas**
Propósito	Permitir listar todas las reglas del sistema.
Actores	Usuario
Resumen	El caso de uso inicia cuando el actor accede a la opción Configurar Reglas, luego de activada la funcionalidad se pueden obtener todas las reglas del sistema, acto seguido el caso de uso termina.
Referencias	
Actores	Usuario

Tabla 3 Descripción breve del caso de uso: Configurar Reglas

Caso de Uso	**Ver Regla**

Propósito	Permitir ver una regla en específico
Actores	Usuario
Resumen	El caso de uso inicia cuando el actor accede a la opción Ver Regla, luego de activada la funcionalidad el usuario puede acceder a una regla en específica, luego el caso de uso termina.
Referencias	
Actores	Usuario
Requisitos	RF1

Tabla 4 Descripción breve del caso de uso: Ver Regla

Caso de Uso	**Modificar Regla**
Propósito	Permite Modificar una regla en específico
Actores	Usuario
Resumen	El caso de uso inicia cuando el actor accede a la opción Modificar Regla, después de activada la funcionalidad se permite modificar una regla especificada, luego el caso de uso termina
Referencias	
Actores	Usuario
Requisitos	RF1

Tabla 5 Descripción breve del caso de uso: Modificar Regla

Conclusiones

El desarrollo de este capítulo ha permitido analizar los requisitos funcionales a través de los cuales se definieron las funcionalidades a implementar, lo cual ha hecho posible alcanzar un mejor entendimiento de la solución y las restricciones que deben existir para satisfacer las necesidades de los clientes. Se identificó al actor que interviene, además del diagrama de casos de uso del sistema, logrando una representación detallada de cada proceso. Teniendo en cuenta estos aspectos se posibilita un mejor

entendimiento de la solución propuesta y para que se logre su total funcionamiento se debe cumplir tanto con los requerimientos de software como los de hardware anteriormente planteados.

Capítulo 3. Diseño de las funcionalidades para la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM

En el presente capítulo se profundiza en los casos de usos detallándolos de manera que permitan reflejar una vista interna del sistema, descrito con el lenguaje de los desarrolladores. Trazando como objetivo esencial la transformación los requerimientos definidos a una propuesta de diseño que constituirá una guía para la implementación del sistema.

4.1 Descripción de la arquitectura propuesta

En la realización de la solución se define a utilizar la arquitectura Modelo Vista Controlador (MVC), para separar los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones

El patrón MVC se ve reflejado de la siguiente forma:

Vistas o Interfaz

Esta capa incluye todas las páginas interfaces XHTML que interactúan con el usuario, estas páginas están desarrolladas por componentes de Java Server Faces, componentes de la librería RichFaces 3.2.0 G, y ajax4jsf los cuales realizan solicitudes a la capa controladora mediante eventos que envían estos componentes cuando el cliente realiza alguna acción en las mismas, además de, Facelets como motor de plantillas, los cuales permiten el desarrollo de interfaces amigables en un periodo corto de tiempo. Para lograr la integración con las otras capas se utiliza el framework Seam.

Controlador o Lógica del negocio

En esta capa se encuentra lo referente a la lógica del negocio del sistema estando representada por clases controladoras las cuales son las encargadas de recibir y gestionar los eventos enviados por las interfaces y de acorde al carácter de la petición ejecutar una acción; ejemplo la de modificar los datos del modelo. Esta capa utiliza Seam como framework de integración entre los componentes visuales y los de acceso a datos y Drools para definir las Reglas del Negocio.

Modelo de datos

En esta capa se encuentran las entidades persistentes, que no son clases Java o Bean que con el uso de la implementación de JPA de Hibernate 3.3 se realiza mapeo de objeto relacional que constituye la

implementación para Enterprise Java Bean, logrando aislar la implementación del gestor de base de datos además de lograr una forma de conectar mediante Seam estas entidades con la capa lógica del negocio.

4.2 Modelo de Diseño

El Modelo de Diseño es un modelo de objetos que describe la realización de los casos de uso, es abarcador y está compuesto por artefactos que engloban todas las clases del diseño, subsistemas, paquetes, colaboraciones, y las relaciones entre ellos. Se centra principalmente en cómo los requisitos funcionales y no funcionales, junto a otras restricciones relacionadas con el entorno de implementación, tienen impacto en las funcionalidades a considerar. Es usado como una entrada inicial en las actividades de implementación y prueba.

3.2.1 Realización de casos de uso del diseño

La realización de los casos de uso está organizada en paquetes. A continuación se explica la organización por paquetes del sistema y se muestra el diagrama de paquetes.

Para elaborar el modelo de diseño y con el objetivo de dividir el sistema en fragmentos manejables para su futura implementación, se define una estructura de paquetes, empleando el criterio de empaquetamiento por proceso, siguiendo la estructura de procesos definidos en el sistema. Cada uno de ellos estará compuesto por diversos subpaquetes, los cuales a su vez contienen los diagramas de clases del diseño para cada caso de uso.

Todos estos paquetes están graficados en dependencia de la relación que guardan entre sí, utilizando el paquete repositorio de clases para su funcionamiento.

El paquete repositorio de clases está compuesto por 3 subpaquetes, sesiones, vistas y Reglas del Negocio. El de las sesiones, estará conformado por las clases controladoras personalizadas.

El paquete de las vistas, en cambio, estará compuesto por contenidos web referentes a las páginas clientes y los formularios que las componen, además de contener las vistas que interactúan con el usuario.

El paquete Reglas del Negocio Contiene las clases java en las que se expresan las Reglas del Negocio.

Diagrama de Paquetes

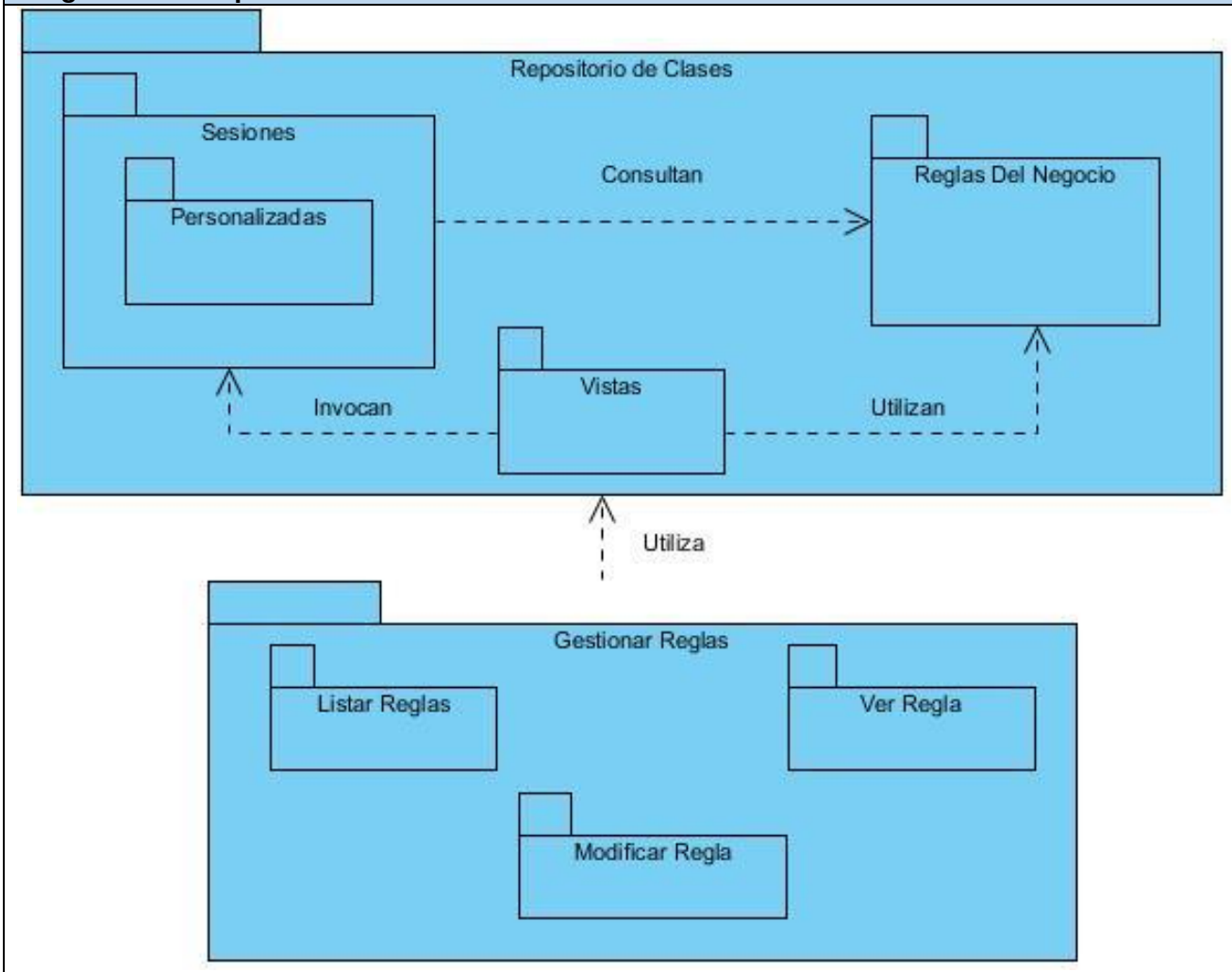


Figura 4 Diagrama de Paquetes

3.2.2 Patrones de diseño

GRASP (Patrones Generales de Software para Asignar Responsabilidades)

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. (34) Entre los patrones de diseño más utilizados se encuentran los patrones GRASP que son patrones de software para la asignación general de responsabilidades y describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. En esta investigación se pueden destacar como patrones fundamentales:

Experto: es un patrón que se usa para asignar responsabilidades; es un principio básico que suele ser útil en el diseño orientado a objetos. Tiene como beneficio que se conserva el encapsulamiento, ya que

los objetos se valen de su propia información para hacer lo que se les pide. Soporta un bajo acoplamiento, lo cual favorece tener sistemas más robustos y de fácil mantenimiento. Además el comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y mantener. (35) En la figura 5 se representa la clase controladora gestionarFichero que cumple con este patrón, ya que maneja toda la información que contiene la clase objetoRegla y a su vez las clases Definicion y Resultado.

Creador: el patrón creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. (35) Este patrón se evidencia en la figura 5 donde la clase controladora es la encargada de crear la lista de objetoRegla. Se evidencia en las clases controladoras quienes tienen la información necesaria para crear nuevas instancias de objetos y así acceder a sus métodos.

Bajo acoplamiento: asignar una responsabilidad para mantener bajo acoplamiento. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. (35) Todas las clases definidas en el diseño de la propuesta de solución no están sobrecargadas de métodos y realizan todas las funciones necesarias para su buen funcionamiento.

Patrones Experto, Creador y Bajo Acoplamiento

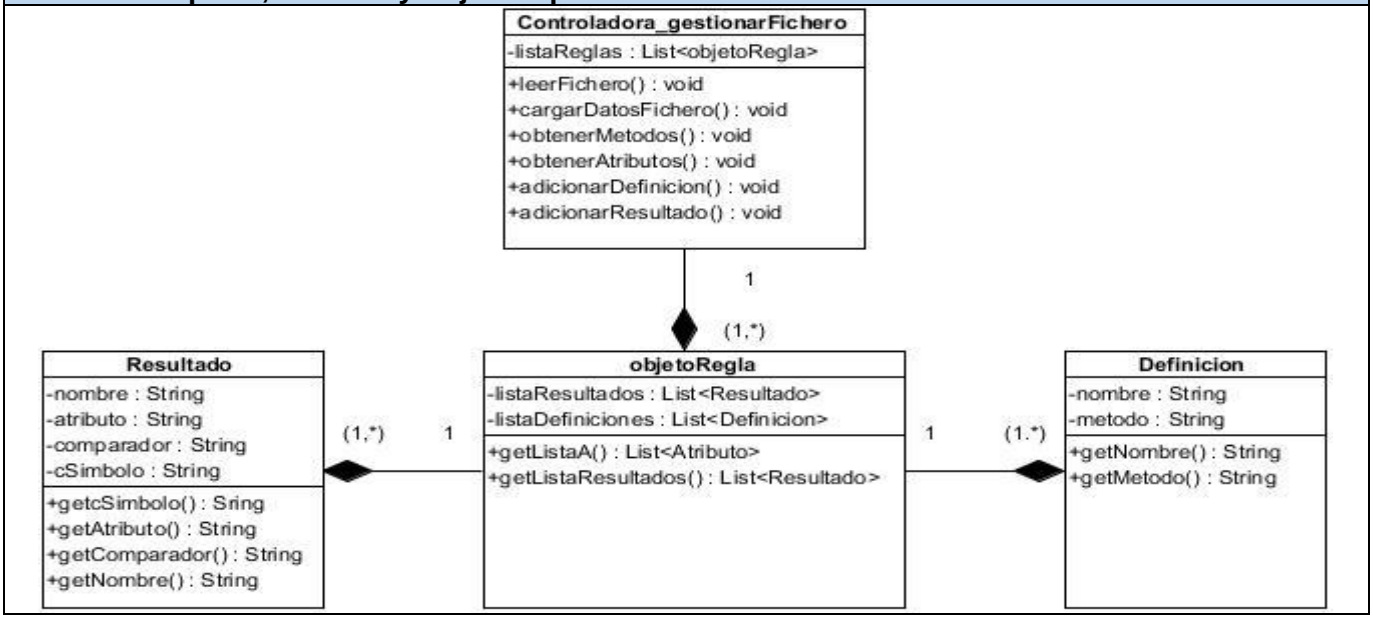


Figura 5 Ejemplo que evidencia los Patrones Experto, Creador y Bajo Acoplamiento

Alta cohesión: asigna una responsabilidad de modo que la cohesión siga siendo alta. En la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. (35) Las clases se reparten las responsabilidades según las tareas que se realizan. Si el usuario desea configurar las reglas, la clase configurarReglas es la encargada de realizarlo. Por otro lado si el investigador desea gestionar las reglas, la clase gestionarFichero se encarga de todo el proceso y si desea modificar las reglas, la clase modificarRegla es la responsable.

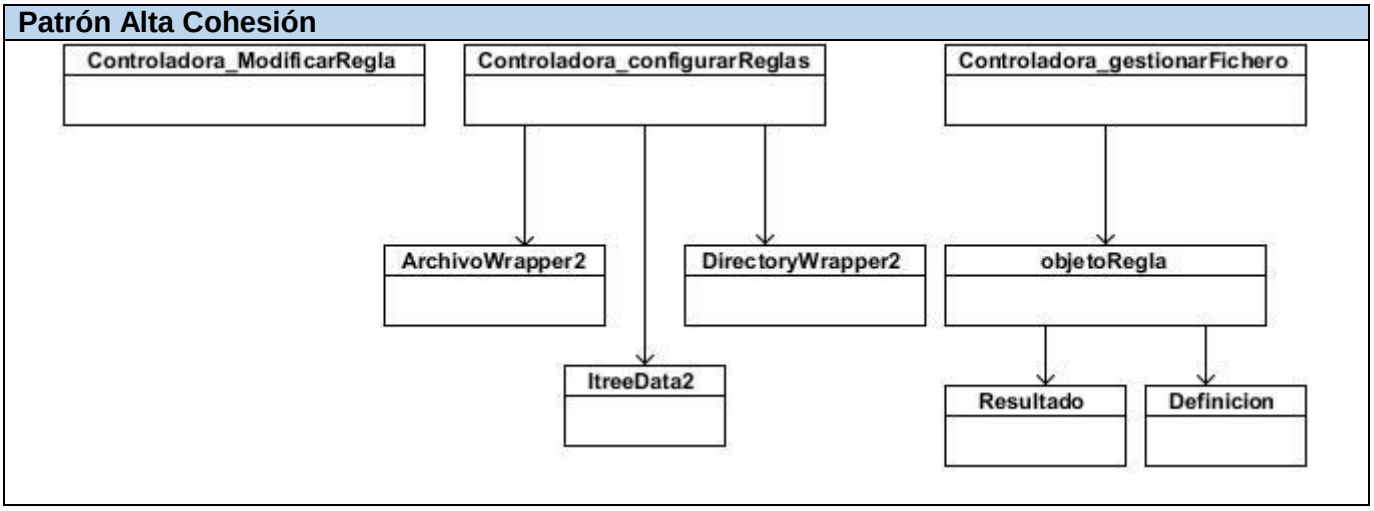


Figura 6 Ejemplo que evidencia el Patrón Alta Cohesión

4.3 Diagrama de clases del diseño

Un diagrama de clases muestra un conjunto de clases que componen el sistema, interfaces y colaboraciones, así como las relaciones que existen entre ellos. Este diagrama se utiliza para modelar la vista de diseño estática y estructural de un sistema o de una parte del modelo. Son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa. (36)

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 encargaran del funcionamiento y la relación entre uno y otro.

Los diagramas de clases del diseño que se expondrán a continuación fueron modelados utilizando estereotipos web.

Diagrama de clases del diseño: configurarReglas

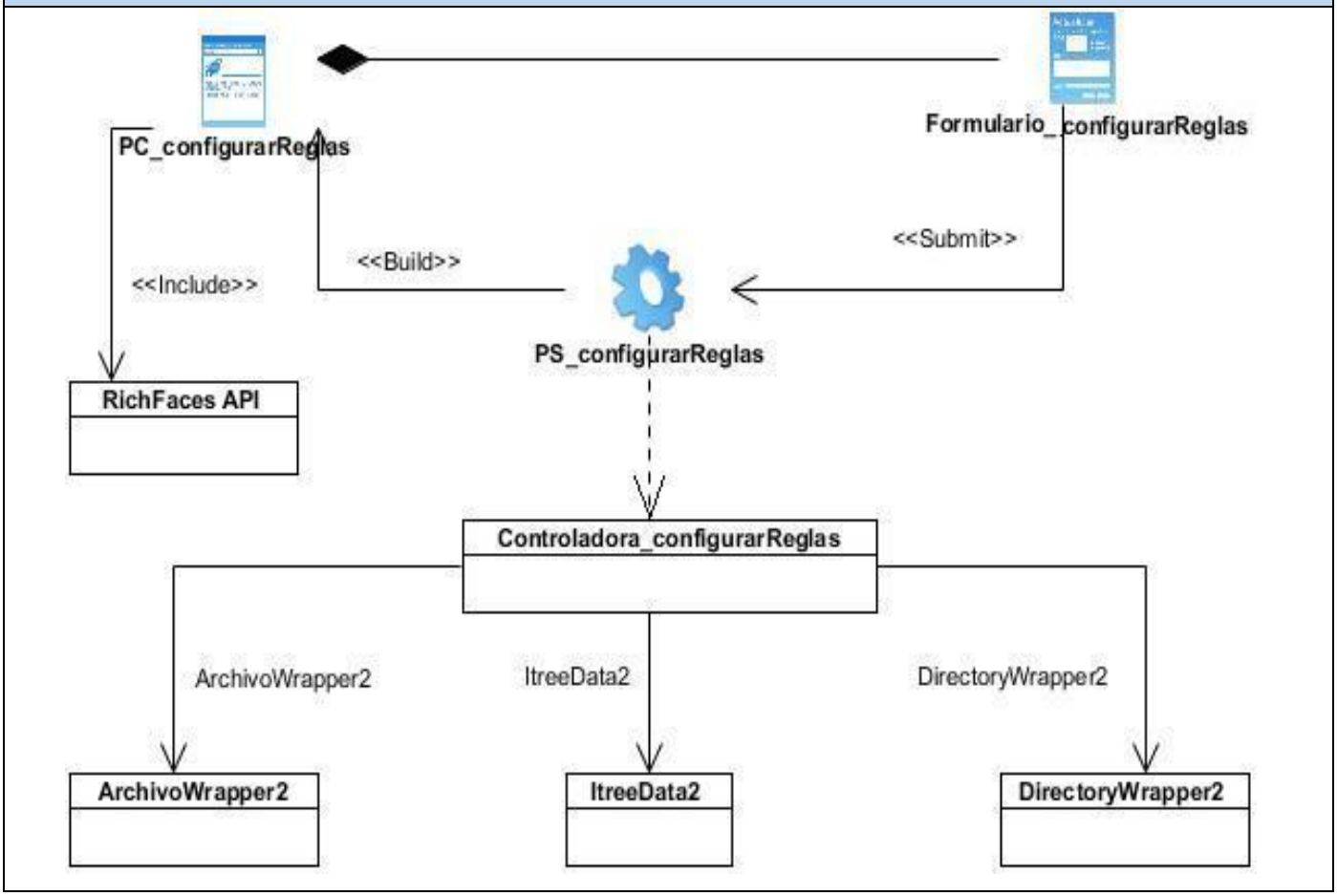


Figura 7 Diagrama de clases del diseño: configurarReglas

Diagrama de clases del diseño: modificarRegla

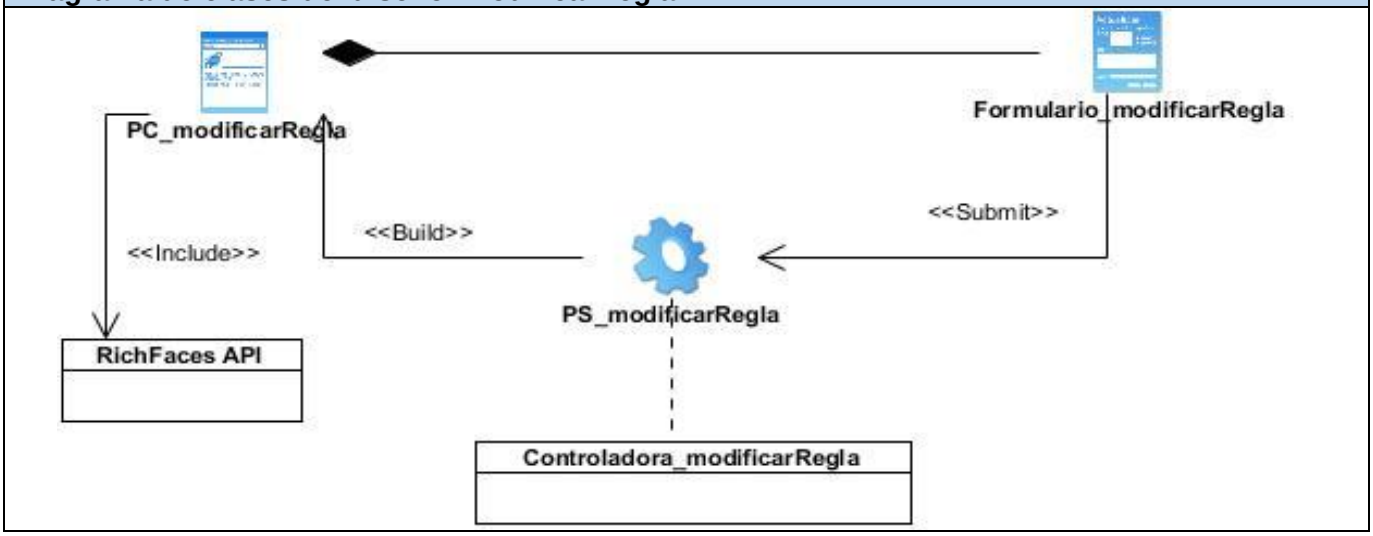


Figura 8 Diagrama de clases del diseño: modificarRegla

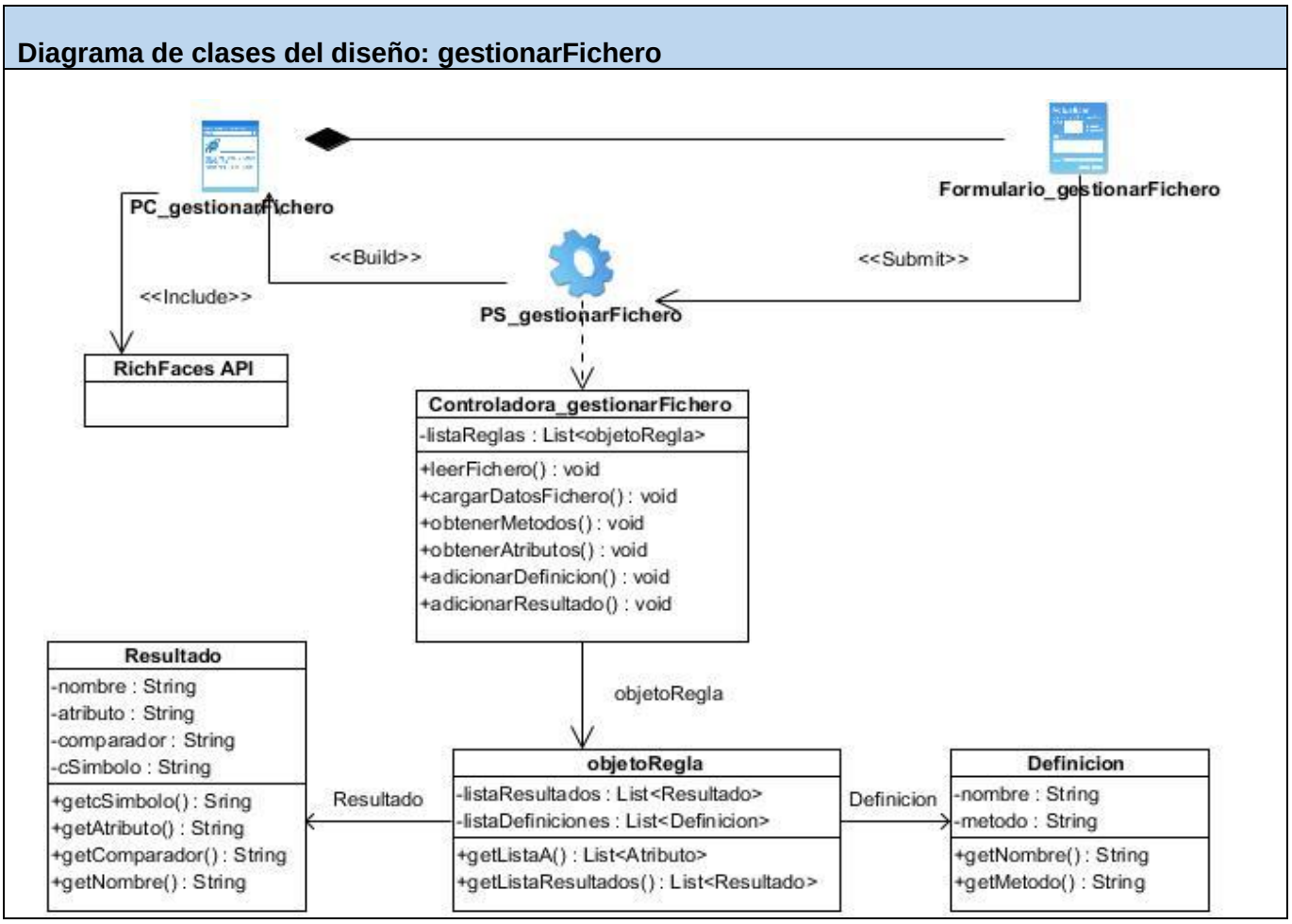


Figura 9 Diagrama de clases del diseño: gestionarFichero

4.4 Descripción de clases del diseño

Las clases del diseño están agrupadas en:

Páginas Clientes: las páginas clientes o ClientPages (CP, como son llamadas comúnmente por sus siglas en inglés), están compuestas por código HTML, CSS, JavaScript. Estas son interpretadas por los navegadores web presentándole al usuario la interfaz, que le permite interactuar con el sistema.

Páginas Servidoras: son conocidas también como Server Pages (SP por sus correspondientes siglas en inglés). Estas están compuestas por componentes Facelets, RichFaces, JSF, Seam UI, así como código HTML. Éste código es ejecutado en el servidor web, generando páginas clientes que pueden ser representadas por los navegadores web.

Formularios: un formulario HTML es una sección de un documento enmarcado entre tags<form> y que puede contener elementos especiales llamados controles (casillas de verificación (checkboxes), radiobotones (radio buttons), menús, entre otros.), y rótulos (labels) en esos controles. Los usuarios normalmente "*completan*" un formulario modificando sus controles (introduciendo texto, seleccionando objetos de un menú, entre otras operaciones.), y lo envían al servidor donde estos son procesados. Es una manera de obtener en el servidor información entrada por el usuario en el cliente.

Controladoras: las clases controladoras implementan la lógica del negocio que se está informatizando, generalmente cada una se encarga de la implementación de un caso de uso o un proceso en dependencia de la complejidad de los mismos.

Build: representa una asociación especial que relaciona las páginas cliente con las páginas servidor, es decir, las páginas que se encuentran en el servidor construyen las páginas en el cliente.

Submit: es la relación que se crea siempre entre una página servidor y un formulario, a través de esta relación el formulario manda los valores de sus campos al servidor, para ser procesados por la página servidor.

A continuación serán explicadas algunas de las clases identificadas para su implementación describiendo las responsabilidades que realizarán las páginas servidoras que corresponden a la lógica de negocio, con el objetivo de tener una mayor comprensión del funcionamiento que tendrá el sistema propuesto.

Nombre: configurarReglas	
Tipo de Clase: Controladora	
Atributos	Tipo
treeData	TreeNode
Para cada Funcionalidad	
Nombre	void OnNodeCollapseExpand(org.richfaces.event.NodeExpandedEvent event)
Descripción	Verifica si el árbol está expandido o no
Nombre	void expandDirectorio(TreeNode selected)
Descripción	Expande un directorio a través de un nodo seleccionado
Nombre	void constructor()
Descripción	Construye el árbol a partir de una dirección física
Nombre	void prune(TreeNode selected)
Descripción	Elimina los directorios de un nodo colapsado
Nombre	TreeNode getTreeData()
Descripción	Devuelve el atributo treeData
Nombre	void setTreeData(TreeNode treeData)
Descripción	Cambia el valor del atributo treeData por uno pasado por parámetro

Tabla 6 Descripción de la clase Controladora gestionarArbol

Nombre: gestionarFichero	
Tipo de clase: Controladora	
Atributo	Tipo
listMethod	List<Method>
listAtribute	List<Field>
direccionClase	List<String>
dirImport	List<String>
listaGlobal	List<String>
listaReglas	List<objetoRegla>
listMetodo	List<String>
listAtributo	List<String>
listaComparador	List<String>
listaSimbolos	List<String>
direccionRegla	String
nombreDRL	String
nombreObjetoRegla	String
nombreResultado	Resultado
objComparador	comparador
direccion	String
Para cada Funcionalidad	
Nombre	gestionarFichero ()
Descripción	Inicializa la clase gestionarFichero.
Nombre	void leerFichero(String dir)
Descripción	Lee un fichero de texto a través de una dirección especiada por parámetro
Nombre	void cargarDatosFichero()
Descripción	Obtiene los datos de un fichero
Nombre	void obtenerMetodos(objetoRegla aux)
Descripción	Obtiene todos los métodos de la clase java en la que se expresa la regla
Nombre	void obtenerAtributos(objetoRegla aux)
Descripción	Obtiene todos los atributos de la clase java en la que se expresa la regla
Nombre	void guardarRegla()
Descripción	Guarda un fichero drl con las nuevas modificaciones
Nombre	String nombre()
Descripción	Obtiene el nombre de la clase java en la que se expresa la regla
Nombre	void adicionarDefinicion()
Descripción	Adiciona una nueva definición a la regla
Nombre	void adicionarResultado()
Descripción	Adiciona un nuevo resultado a la regla
Nombre	void auxEliminar(String aux)
Descripción	Recibe el nombre de un objeto regla para luego eliminarlo
Nombre	void auxObtenerNombreDefinicion(Definicion aux)
Descripción	Recibe el nombre de un objeto Definicion para luego eliminarlo
Nombre	void auxObtenerNombreResultado(Resultado aux)
Descripción	Recibe el nombre de un objeto Resultado para luego eliminarlo
Nombre	void eliminarResultado()

Descripción	Elimina un Resultado
Nombre	void eliminarDefinicion()
Descripción	Elimina una Definicion
Nombre	void eliminarRegla()
Descripción	Elimina una Regla
Nombre	String mostrarDireccion()
Descripción	Muestra la dirección física del archivo drl
Nombre	void llenarListaSimbolos()
Descripción	Obtiene las listas con los comparadores de cada definición
Nombre	void llenarListaValor()
Descripción	Obtiene las listas con los comparadores de cada definición en lenguaje natural
Nombre	void auxDuplicarRegla(objetoRegla aux)
Descripción	Obtiene un nuevo objeto Regla para luego ser duplicado
Nombre	void duplicarRegla()
Descripción	Duplica un objeto regla seleccionado
Nombre	String mostrarFichero()
Descripción	Muestra el fichero drl con las nuevas modificaciones
Nombre	int getCont()
Descripción	Obtiene la cantidad de líneas leídas en el fichero drl
Nombre	void setCont(int cont)
Descripción	Cambia el valor de la variable cont por uno pasado por parámetro
Nombre	List<Method> getListMethod()
Descripción	Devuelve la variable listMethod
Nombre	void setListMethod(List<Method> listMethod)
Descripción	Cambia el valor de la variable listMethod por uno pasado por parámetro
Nombre	List<Field> getListAtribute()
Descripción	Devuelve la variable listAtribute
Nombre	void setListAtribute(List<Field> listAtribute)
Descripción	Cambia el valor de la variable listAtribute por uno pasado por parámetro
Nombre	String getDireccionRegla()
Descripción	Devuelve la variable direccion
Nombre	void setDireccion(String direccion)
Descripción	Cambia el valor de la variable direccion por uno pasado por parámetro
Nombre	List<objetoRegla> getListaReglas()
Descripción	Devuelve la variable listaReglas
Nombre	void setListaReglas(List<objetoRegla> listaReglas)
Descripción	Cambia el valor de la variable listaReglas por uno pasado por parámetro
Nombre	String getNombreDRL()
Descripción	Devuelve la variable nombreDRL
Nombre	void setNombreDRL(String nombreDRL)
Descripción	Cambia el valor de la variable listaReglas por uno pasado por parámetro
Nombre	String getDireccionRegla()
Descripción	Devuelve la variable direccionRegla
Nombre	setDireccionRegla(String direccionRegla)
Descripción	Cambia el valor de la variable direccionRegla por uno pasado por parámetro
Nombre	comparador getObjComparador()
Descripción	Devuelve la variable objComparador

Nombre	void setObjComparador(comparador objComparador)
Descripción	Cambia el valor de la variable objComparador por uno pasado por parámetro
Nombre	List<String> getListMetodo()
Descripción	Devuelve la variable listMetodo
Nombre	void setListMetodo(List<String> listMetodo)
Descripción	Cambia el valor de la variable listMetodo por uno pasado por parámetro
Nombre	List<String> getListAtributo()
Descripción	Devuelve la variable listAtributo
Nombre	void setListAtributo(List<String> listAtributo)
Descripción	Cambia el valor de la variable listAtributo por uno pasado por parámetro
Nombre	List<String> getListaSimbolos()
Descripción	Devuelve la variable listaSimbolos
Nombre	void setListaSimbolos(List<String> listaSimbolos)
Descripción	Cambia el valor de la variable listaSimbolos por uno pasado por parámetro
Nombre	List<String> getlistaComparador()
Descripción	Devuelve la variable listaSimbolos
Nombre	setlistaComparador(List<String> listaComparador)
Descripción	Cambia el valor de la variable listaSimbolos por uno pasado por parámetro
Nombre	Definicion getDefinicionNew()
Descripción	Devuelve la variable definicionNew
Nombre	void setDefinicionNew(Definicion definicionNew)
Descripción	Cambia el valor de la variable definicionNew por uno pasado por parámetro
Nombre	Resultado getResultadoNew()
Descripción	Devuelve la variable resultadoNew
Nombre	void setResultadoNew(Resultado resultadoNew)
Descripción	Cambia el valor de la variable resultadoNew por uno pasado por parámetro
Nombre	String getNombreNuevaRegla()
Descripción	Devuelve la variable nombreNuevaRegla
Nombre	void setNombreNuevaRegla(String nombreNuevaRegla)
Descripción	Cambia el valor de la variable nombreNuevaRegla por uno pasado por parámetro
Nombre	String getIdCombo()
Descripción	Devuelve la variable idCombo
Nombre	void setIdCombo(String idCombo)
Descripción	Cambia el valor de la variable idCombo por uno pasado por parámetro
Nombre	String getIdInputDef()
Descripción	Devuelve la variable idInputDef
Nombre	void setIdInputDef(String idInputDef)
Descripción	Cambia el valor de la variable idInputDef por uno pasado por parámetro
Nombre	String getIdInputResul()
Descripción	Devuelve la variable idInputDef
Nombre	void setIdInputResul(String idInputResul)
Descripción	Cambia el valor de la variable idInputResul por uno pasado por parámetro

Tabla 7 Descripción de la clase Controladora gestionarFichero

Nombre: objetoRegla	
Tipo de clase: Entidad	
Atributo	Tipo
nombreObjeto	String
listaResultados	List<Resultado>
listaA	List<Atributo>
listaDefiniciones	List<Definicion>
listaM	List<Metodo>
objetosCreados	List<String>
Para cada Funcionalidad	
Nombre	objetoRegla()
Descripción	Inicializa la clase objetoRegla
Nombre	objetoRegla(String nombreObjeto, List<Resultado> listaResultados, List<Atributo> listaA, List<Definicion> listaDefiniciones, List<Metodo> listaM, List<String> objetosCreados)
Descripción	Inicializa la clase objetoRegla con el valor de cada variable
Nombre	objetoRegla(String nombreObjeto)
Descripción	Inicializa la clase objetoRegla con el valor de la variable nombreObjeto
Nombre	String getNombreObjeto()
Descripción	Devuelve la variable nombreObjeto
Nombre	void setNombreObjeto(String nombreObjeto)
Descripción	Cambia el valor de la variable nombreObjeto por uno pasado por parámetro
Nombre	List<Resultado> getListaResultados()
Descripción	Devuelve la variable listaResultados
Nombre	void setListaResultados(List<Resultado> listaResultados)
Descripción	Cambia el valor de la variable listaResultados por uno pasado por parámetro
Nombre	List<Atributo> getListaA()
Descripción	Devuelve la variable listaA
Nombre	void setListaA(List<Atributo> listaA)
Descripción	Cambia el valor de la variable listaA por uno pasado por parámetro
Nombre	List<Definicion> getListaDefiniciones()
Descripción	Devuelve la variable listaDefiniciones
Nombre	void setListaDefiniciones(List<Definicion> listaDefiniciones)
Descripción	Cambia el valor de la variable listaDefiniciones por uno pasado por parámetro
Nombre	List<Metodo> getListaM()
Descripción	Devuelve la variable listaM
Nombre	void setListaM(List<Metodo> listaM)
Descripción	Cambia el valor de la variable listaM por uno pasado por parámetro
Nombre	List<String> getObjetosCreados()
Descripción	Devuelve la variable objetosCreados
Nombre	void setObjetosCreados(List<String> objetosCreados)
Descripción	Cambia el valor de la variable objetosCreados por uno pasado por parámetro

Tabla 8 Descripción de la clase Entidad objetoRegla

Nombre: Definicion
Tipo de clase: Entidad

Atributo	Tipo
nombre	String
atributo	String
comparador	String
cSimbolo	String
Para cada Funcionalidad	
Nombre	Definicion()
Descripción	Inicializa la clase Definicion
Nombre	Definicion(String nombre,String atributo, String comparador,String cSimbolo, String valor)
Descripción	Inicializa la clase Definicion con el valor de cada variable
Nombre	String getcSimbolo()
Descripción	Devuelve la variable cSimbolo
Nombre	void setcSimbolo(String cSimbolo)
Descripción	Cambia el valor de la variable cSimbolo por uno pasado por parámetro
Nombre	String getAtributo()
Descripción	Devuelve la variable atributo
Nombre	void setAtributo(String atributo)
Descripción	Cambia el valor de la variable atributo por uno pasado por parámetro
Nombre	String getComparador()
Descripción	Devuelve la variable comparador
Nombre	void setComparador(String comparador)
Descripción	Cambia el valor de la variable comparador por uno pasado por parámetro
Nombre	String getValor()
Descripción	Devuelve la variable valor
Nombre	void setValor(String valor)
Descripción	Cambia el valor de la variable valor por uno pasado por parámetro
Nombre	String getNombre()
Descripción	Devuelve la variable nombre
Nombre	void setNombre(String nombre)
Descripción	Cambia el valor de la variable nombre por uno pasado por parámetro

Tabla 9 Descripción de la clase Entidad Definición

Nombre: Resultado	
Tipo de Clase: Entidad	
Atributos	Tipo
metodo	String
nombre	String
ListaV	List<Atributo>
Para cada Funcionalidad	
Nombre	Resultado(String metodo, String nombre, List<Atributo> listaV)
Descripción	Inicializa la clase Resultado con el valor de cada variable
Nombre	Resultado(String metodo, String nombre)
Descripción	Inicializa la clase Resultado con el valor de cada variable
Nombre	Resultado()
Descripción	Inicializa la clase Resultado

Nombre	String getNombre()
Descripción	Devuelve la variable nombre
Nombre	void setNombre(String nombre)
Descripción	Cambia el valor de la variable nombre por uno pasado por parámetro
Nombre	String getMetodo()
Descripción	Devuelve la variable metodo
Nombre	void setMetodo(String metodo)
Descripción	Cambia el valor de la variable metodo por uno pasado por parámetro
Nombre	List<Atributo> getListaV()
Descripción	Devuelve la variable listaV
Nombre	void setListaV(List<Atributo> listaV)
Descripción	Cambia el valor de la variable listaV por uno pasado por parámetro

Tabla 10 Descripción de la clase Entidad Resultado

Conclusiones

Como resultado del estudio realizado en este capítulo, correspondiente al flujo de diseño, se han identificado las clases necesarias para la implementación de las funcionalidades para la gestión de las Reglas del Negocio. Se realizó el modelo de diseño, explicándose la estructura y la definición de elementos que este posee. Además se definieron los diagramas de clases del diseño y el diagrama de paquetes relacionado con la solución.

Capítulo 4: Implementación de las funcionalidades para la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM.

Después de realizar el análisis y el diseño de la solución propuesta, se tienen creadas todas las condiciones para comenzar la implementación de la misma. En este capítulo se exponen los componentes del sistema, así como la forma en la que interactúan para posibilitar su correcto funcionamiento. Se elaborará el Modelo de Despliegue, el cual muestra la ubicación física de los componentes necesarios para que se puedan ejecutar los procesos, también se exponen los aspectos referentes a la seguridad del sistema, las estrategias de codificación, así como la forma en que se tratarán los errores.

5.1 Implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se encuentran: datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.

El propósito del modelo de implementación es definir la organización del código, planificar las integraciones de sistema necesarias en cada iteración e implementar las clases y subsistemas encontrados durante el Diseño. (37)

Se debe proponer una estrategia de codificación que defina los formatos para la asignación de nombres a las variables, estilo de programación y métodos de documentación.

5.1.1 Diagrama de Componentes

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación. Modelan la vista estática de un sistema. Muestran la organización y las dependencias entre un conjunto de componentes. (38)

De los estereotipos estándar que se aplican a los componentes según el lenguaje de modelado UML se emplean: library y file, los cuales representan, una biblioteca de objetos estática o dinámica y un documento que contiene código fuente, respectivamente. Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente se refiere a los servicios ofrecidos por otro componente.

Diagrama de componentes

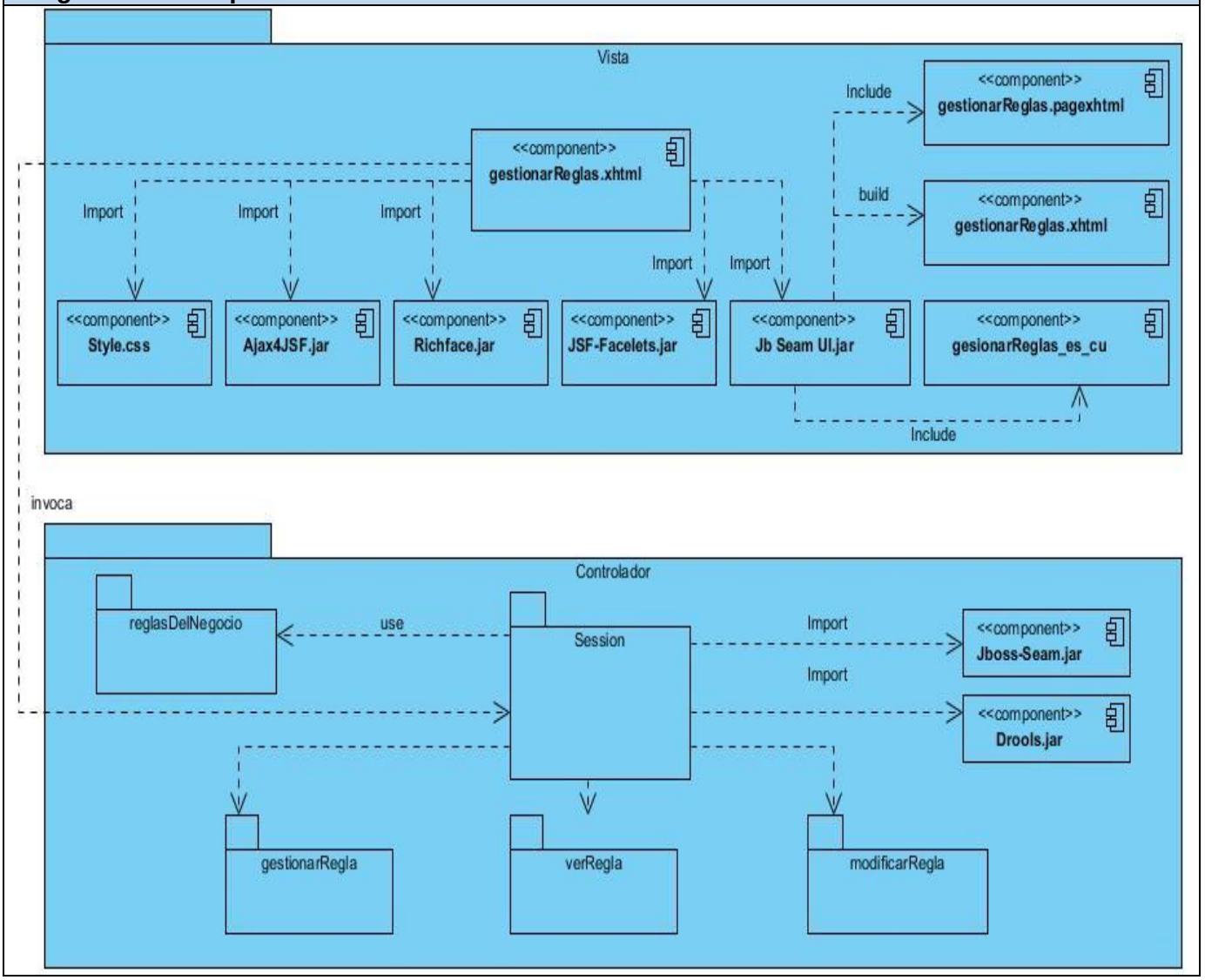


Figura 10 Diagrama de componentes

5.1.2 Diagrama de Despliegue

El Modelo de Despliegue es un tipo de Diagrama de Estructura que describe la distribución física del sistema. Se emplea para modelar el hardware utilizado en las implementaciones del sistema y las relacionen entre sus componentes. (38)

Para la implantación y la utilización de la aplicación en una institución hospitalaria el usuario debe conectarse a esta mediante una PC cliente utilizando un navegador web. Las peticiones por el protocolo HTTP serán procesadas por el servidor de aplicaciones que enviará la respuesta al cliente. El servidor de aplicaciones interactúa con el servidor de base de datos a través del protocolo TCP/IP.

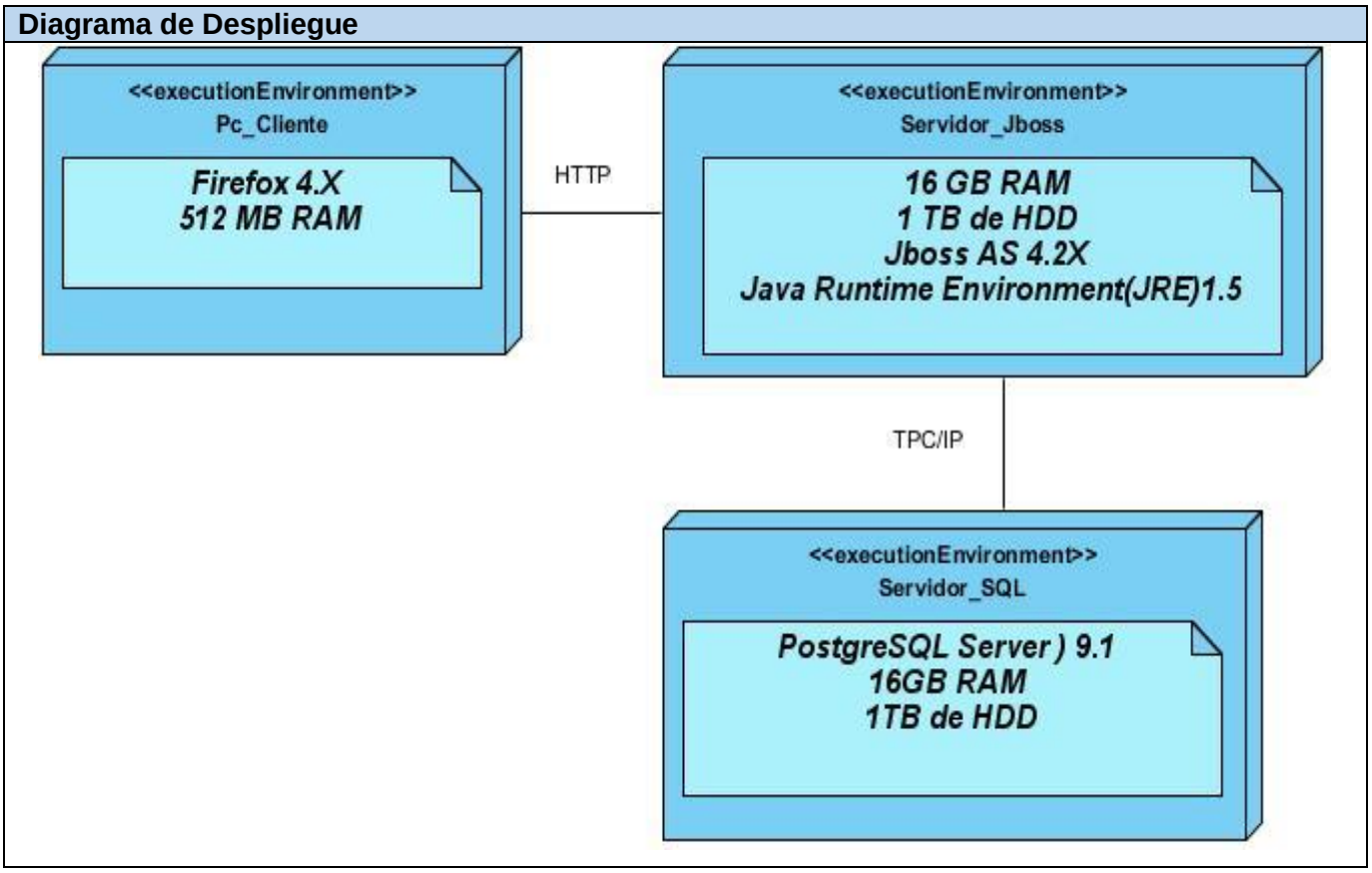


Figura 11 Diagrama de Despliegue

5.2 Tratamiento de errores

Un error es un fallo que se incluye en un programa, no logrando el producto deseado y afectando al cliente. Las excepciones son el mecanismo recomendado para la propagación de errores que se produzcan durante la ejecución de las aplicaciones. Cuando dicho error ocurre dentro de un método Java, automáticamente se crea un objeto 'Excepcion' el cual es tratado en el sistema de ejecución. Este objeto contiene información sobre la excepción, incluyendo su tipo y el estado del programa cuando ocurrió el error. Otro punto de vista es cuando se desarrolla un software, que se intenta proveer de cierta funcionalidad al usuario, si esa funcionalidad no se cumple, se puede decir que el software presenta errores. Estos son detectables por el usuario, lo que influye en la calidad externa del software.

En la solución propuesta se propone el tratamiento de excepciones principalmente en las regiones críticas de código, que son los fragmentos donde se manipulan datos que son insertados o modificados y en las validaciones de los datos que son insertados por el usuario.

El manejo de las excepciones o errores, en las clases controladoras de procesos, utilizará el bloque **try** para detectar cuando ocurra algún fallo y un bloque **catch** donde se manejarán dichas excepciones, mediante mensajes que se muestran en la interfaz de usuario.

Ejemplo de uso:

```
try{
//declaración que causa la excepción
}
catch
 (NombredeExcepcion obj)
{
//código para tratar el error
}
```

5.3 Seguridad

La seguridad es un tema de gran importancia para cualquier Sistema de Información y toma mayor relevancia cuando se gestiona información médica. Para garantizar la seguridad de la solución desarrollada se sostendrá un control a nivel de usuarios y contraseñas, permitiendo el acceso por tipo de usuario logrando así la visibilidad sólo a las áreas establecidas de acorde a la función que realizan. Las contraseñas solo podrán ser cambiadas por el usuario o por el administrador del sistema.

Otra cuestión es lograr la fiabilidad en las estaciones de trabajo, para lograr esto se define un segundo nivel de seguridad a nivel de estaciones de trabajo lo que posibilita la ejecución sólo de las aplicaciones que hayan sido definidas para la estación en cuestión.

5.4 Estrategias de codificación. Estándares y estilos a utilizar

Un estándar de codificación comprende todos los aspectos de la generación de código y deber ser práctico. Un código fuente en su totalidad debe ser de fácil entendimiento y reflejar un estilo armonioso, como si un único programador lo hubiese programado de una sola vez. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse que todos los programadores del proyecto trabajen de forma coordinada. Este debería indicar como operar con la base del código existente en caso de realizar modificaciones y/o mantenimiento al sistema.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación, es de gran importancia para la calidad del software. La aplicación de estándares de codificación además posibilita que el software que se obtiene sea fácil de comprender y de mantener en el tiempo.

Algunas de las especificaciones que se utilizan en el código son:

- Deben escribirse comentarios al inicio de cada clase y método, con el objetivo de brindar una breve descripción de los propósitos de cada funcionalidad.
- Los nombres de las clases deben ser lo más simple y sugerente posible, utilizando palabras completas y abreviaturas conocidas.
- Los nombres de las variables deben ser cortos y significativos, de manera que se entienda con facilidad su significado. Se deben evitar las variables de una sola letra, excepto para las temporales de corto uso.
- Todos los nombres de variables de instancia o de clase deben estar constituidos por palabras con la primera letra de la primera palabra en minúscula y la primera letra de las palabras internas en mayúscula.

5.4.1 Elementos de los estándares de codificación

Se empleará notación CamellCasing para denotar variables y parámetros, esta especifica que la palabra de inicio del identificador comienza con minúscula. Si el identificador está compuesto por más de una palabra entonces estás deben comenzar con mayúsculas.

Para definir una robusta estructura y organización del código, se deben definir algunos estándares para su posterior entendimiento y cumplir con las buenas prácticas establecidas en la Ingeniería de Software. A continuación se resumen algunas de las convenciones tomadas en relación a estos aspectos.

Identación: lograr una estructura uniforme para los bloques de código así como para los diferentes niveles de anidamiento.

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.

Comentarios, separadores, líneas, espacios en blanco y márgenes: establecer un modo común para comentar el código de forma tal que sea comprensible con sólo leerlo una vez.

Se recomienda comentar al inicio de la clase o función especificando el objetivo de la misma así como los parámetros que usa (especificar tipos de dato, y objetivo del parámetro) entre otras cosas.

Se recomienda 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.

Se recomienda usar espacios en blanco entre los operadores para lograr una mayor legibilidad en el código. Ejemplo: producto = nomproducto.

Variables y constantes: el nombre que se le da a las variables debe comenzar con la primera letra en minúscula, en caso de que sea un nombre compuesto se empleará notación CamellCasing. El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito de la misma.

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 PascalCasing. Ejemplo: MiClase(). Para el caso de las instancias se comenzara con un prefijo que identificara el tipo de dato, este se escribirá en minúscula.

Para nombrar las funciones se debe tratar de utilizar verbos que denoten la acción que hace la función. Se empleará notación PascalCasing. Ejemplo: function BuscarUnidad(). Si son funciones que obtienen un dato se emplea el prefijo get y si fijan algún valor se emplea el prefijo set.

El nombre empleado para las clases, objetos, atributos y funciones debe permitir que con sólo leerlo se conozca el propósito de los mismos.

Conclusiones

Durante la etapa de implementación los principales artefactos obtenidos fueron los Diagramas de Componentes y el Modelo de Datos, que representan los componentes del software y la representación lógica y física de la información. Se logró alcanzar una aplicación con todas las funcionalidades previstas, que compense las principales necesidades de los clientes. Se analizó también la importancia que representa el tratamiento de errores y la seguridad para la construcción de la aplicación.

Conclusiones Generales

La realización del presente trabajo ha posibilitado de manera general cumplir con el objetivo propuesto, se logró desarrollar las funcionalidades necesarias para gestionar las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM, por lo que se pueden plantear las siguientes conclusiones:

- La detallada descripción de los procesos relacionados con la gestión de las Reglas del Negocio ha permitido obtener una mejor comprensión del problema e identificar las principales necesidades a resolver. La posterior definición de los requerimientos ha sido punto de partida en el proceso de desarrollo de la solución propuesta para lograr que la misma cumpla con las funcionalidades que debe brindar.
- El estudio de los principales sistemas existentes que gestionan las Reglas del Negocio permitió identificar las características indispensables para el desarrollo de la solución.
- El análisis y uso de la arquitectura Modelo Vista Controlador permitió la obtención del diseño de los componentes a desarrollar y corregir las deficiencias detectadas durante la implementación en un período corto de tiempo.
- El desarrollo de las funcionalidades facilita la gestión de las Reglas del Negocio del Sistema de Información Hospitalaria del CESIM, permitiéndole al usuario comprender y modificar sus definiciones sin depender de personal informáticos para poder realizar estas acciones.

Recomendaciones

Luego de culminada la investigación y cumplido el objetivo general de la misma, en vista a mejorar la solución propuesta y con el fin de que se incrementen las funcionalidades brindadas para un mejor uso de éste, se recomienda:

- Implementar otras funcionalidades que además de gestionar las reglas permitan modelar otros procesos del negocio.
- Facilitar una posible integración con otros sistemas y lenguajes de programación.
- Incluir una guía o manual de ayuda que brinde soporte a los usuarios del sistema.

Referencias Bibliográficas

1. **Nieves, Rolando Corratge.** *Software de Propagación en Interiores para Sistemas RFID: Análisis y Diseño.* La Habana : s.n., 2011.

2. **Granados, Eduardo.** odraudek99'. [En línea] [Citado el: 10 de 11 de 2013.] http://odraudek99.wordpress.com/2012/06/12/reglas-de-negocio.

3. bizagi. [En línea] [Citado el: 11 de 11 de 2013.] http://help.bizagi.com/bpmsuite/es/index.html?definir_reglas_de_negocio.htm.

4. Wordpress. [En línea] [Citado el: 11 de 11 de 2013.] http://msaffirio.wordpress.com/2011/08/20/reglas-de-negocio-business-rules.

5. Developer network. [En línea] [Citado el: 11 de 11 de 2013.] http://msdn.microsoft.com/es-es/library/aa561216.aspx .

6. Drools Business Logic Integration Plataform. [En línea] [Citado el: 12 de 11 de 2013.] http://www.jboss.org/drools/drools-guvnor.html.

7. RULEXPRESS. [En línea] [Citado el: 12 de 11 de 2013.] http://www.rulearts.com/DecisionManagement-ESP.

8. Semantic Systems. [En línea] [Citado el: 15 de 11 de 2013.] http://www.semantic-systems.com/productos/software-de-gestion-del-conocimiento-industrial-soluciones-repcon-4/sistema-de-gestion-de-reglas-de-negocio-motor-de-reglas-para-bpm-tarificacion-incentivos-78.

9. **Consultor especializado en Dirección Estratégica, Business Intelligence y Cuadro de Mando Integral, Páez, Francisco.** Cmigestion. [En línea] [Citado el: 14 de 11 de 2013.] http://www.cmigestion.es/2008/business-intelligence/brms-business-rules-management-system.

10. MVC. [En línea] [Citado el: 11 de 12 de 2013.] http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html.

11. Rational Unified Process. [En línea] [Citado el: 15 de 11 de 2013.] http://oscarrofio.netne.net/resumen.html.

12. UML. [En línea] [Citado el: 15 de 11 de 2013.] http://www.ecured.cu/index.php/UML.

13. **Gallardo, David.** DeveloperWorks. [En línea] [Citado el: 20 de 11 de 2013.] http://www.ibm.com/developerworks/ssa/library/os-ecov.

14. Seam - Contextual Components. Introduction to JBoss Seam. [En línea] [Citado el: 23 de 11 de 2013.] http://www.seamframework.org/Documentation.

15. **Desarrollador de software en la Universidad de Deusto, Canarias, Iker.** Slideshare. [En línea] 23 de 11 de 2013. http://www.slideshare.net/ikercanarias/jboss-11467757.

16. Free Download Manager. [En línea] [Citado el: 20 de 11 de 2013.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p.

17. **Alvarez, Miguel Angel.** desarrolloweb.com. [En línea] [Citado el: 27 de 11 de 2013.] http://www.desarrolloweb.com/articulos/497.php.

18. EcuRed. JSF. [En línea] [Citado el: 1 de 12 de 2013.] http://www.ecured.cu/index.php/JSF.

19. RedHat. [En línea] [Citado el: 1 de 12 de 2013.] http://www.jboss.org/richfaces.

20. **ing Ramos, Juan Alonso.** adictosaltrabajo.com. [En línea] [Citado el: 3 de 12 de 2013.] http://adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf.

21. **Bauer Christian, King Gavin.** *Java Persistence with Hibernate.* s.l. : Manning Publications Co, 2007.

22. **Herrera, Cristhian.** adictosaltrabajo.com. [En línea] [Citado el: 3 de 12 de 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=EJB3vsSpring.. 7855-uusji.

23. XML.com. [En línea] O'Reilly Media, Inc. [Citado el: 4 de 12 de 2013.] http://www.xml.com.

24. Facelets y JSF. [En línea] [Citado el: 4 de 12 de 2013.] http://es.scribd.com/doc/44720276/Facelets-y-JSF.

25. **Guaita, Alvaro Martínez.** desarrolloweb.com. [En línea] [Citado el: 5 de 12 de 2013.] http://www.desarrolloweb.com/de_interes/hojas-resumen-xhtml-2961.html.

26. w3c.es. [En línea] [Citado el: 7 de 12 de 2013.] http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo.

27. **Alvarez, Miguel Angel.** desarrolloweb.com. [En línea] [Citado el: 7 de 12 de 2013.] http://www.desarrolloweb.com/articulos/25.php.

28. Drools. [En línea] [Citado el: 7 de 12 de 2013.] http://reglasnegocio.blogspot.com.

29. Oracle. [En línea] [Citado el: 7 de 12 de 2013.] http://docs.oracle.com/javase/tutorial/reflect.

30. Tecnologia y synergix. [En línea] [Citado el: 11 de 12 de 2013.] http://synergix.wordpress.com/2008/07/10/modelo-de-dominio.

31. Slideshare. [En línea] http://www.slideshare.net/maryme/2-requerimientos-del-software.

32. scribd. [En línea] [Citado el: 11 de 12 de 2013.] http://es.scribd.com/doc/37187866/Requerimientos-funcionales-y-no-funcionales.

33. sparxsystems. [En línea] [Citado el: 11 de 12 de 2013.] http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html.

34. **Larman, C.** UML y patrones, Tomo I Capítulos 18, Páginas 185-215. *UML y patrones.*

35. **Robaina, Irilis Ledón.** *Tesis de Diploma de Arquitectura de BioSyS.* La Habana, Cuba : s.n., 2008.

36. Slideshare. [En línea] [Citado el: 15 de 12 de 2013.] http://www.slideshare.net/jpbthames/diagramas-de-clases.

37. Merinde. [En línea] [Citado el: 15 de 12 de 2013.] http://merinde.net/index.php?option=com_content&task=view&id=495&Itemid=291.

38. Slideshare. [En línea] [Citado el: 15 de 12 de 2013.] http://www.slideshare.net/joshell/diagramas-uml-componentes-y-despliegue.

Bibliografía

1. **Nieves, Rolando Corratge.** *Software de Propagación en Interiores para Sistemas RFID: Análisis y Diseño.* La Habana : s.n., 2011.

2. **Granados, Eduardo.** odraudek99'. [En línea] [Citado el: 10 de 11 de 2013.] http://odraudek99.wordpress.com/2012/06/12/reglas-de-negocio.

3. bizagi. [En línea] [Citado el: 11 de 11 de 2013.] http://help.bizagi.com/bpmsuite/es/index.html?definir_reglas_de_negocio.htm.

4. Wordpress. [En línea] [Citado el: 11 de 11 de 2013.] http://msaffirio.wordpress.com/2011/08/20/reglas-de-negocio-business-rules.

5. Developer network. [En línea] [Citado el: 11 de 11 de 2013.] http://msdn.microsoft.com/es-es/library/aa561216.aspx .

6. Drools Business Logic Integration Plataform. [En línea] [Citado el: 12 de 11 de 2013.] http://www.jboss.org/drools/drools-guvnor.html.

7. RULEXPRESS. [En línea] [Citado el: 12 de 11 de 2013.] http://www.rulearts.com/DecisionManagement-ESP.

8. Semantic Systems. [En línea] [Citado el: 15 de 11 de 2013.] http://www.semantic-systems.com/productos/software-de-gestion-del-conocimiento-industrial-soluciones-repcon-4/sistema-de-gestion-de-reglas-de-negocio-motor-de-reglas-para-bpm-tarificacion-incentivos-78.

9. **Consultor especializado en Dirección Estratégica, Business Intelligence y Cuadro de Mando Integral, Páez, Francisco.** Cmigestion. [En línea] [Citado el: 14 de 11 de 2013.] http://www.cmigestion.es/2008/business-intelligence/brms-business-rules-management-system.

10. MVC. [En línea] [Citado el: 11 de 12 de 2013.] http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html.

11. Rational Unified Process. [En línea] [Citado el: 15 de 11 de 2013.] http://oscarrofio.netne.net/resumen.html.

12. UML. [En línea] [Citado el: 15 de 11 de 2013.] http://www.ecured.cu/index.php/UML.

13. **Gallardo, David.** DeveloperWorks. [En línea] [Citado el: 20 de 11 de 2013.] http://www.ibm.com/developerworks/ssa/library/os-ecov.

14. Seam - Contextual Components. Introduction to JBoss Seam. [En línea] [Citado el: 23 de 11 de 2013.] http://www.seamframework.org/Documentation.

15. **Desarrollador de software en la Universidad de Deusto, Canarias, Iker.** Slideshare. [En línea] 23 de 11 de 2013. http://www.slideshare.net/ikercanarias/jboss-11467757.

16. Free Download Manager. [En línea] [Citado el: 20 de 11 de 2013.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p.

17. **Alvarez, Miguel Angel.** desarrolloweb.com. [En línea] [Citado el: 27 de 11 de 2013.] http://www.desarrolloweb.com/articulos/497.php.

18. EcuRed. JSF. [En línea] [Citado el: 1 de 12 de 2013.] http://www.ecured.cu/index.php/JSF.

19. RedHat. [En línea] [Citado el: 1 de 12 de 2013.] http://www.jboss.org/richfaces.

20. **ing Ramos, Juan Alonso.** adictosaltrabajo.com. [En línea] [Citado el: 3 de 12 de 2013.] http://adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf.

21. **Bauer Christian, King Gavin.** *Java Persistence with Hibernate.* s.l. : Manning Publications Co, 2007.

22. **Herrera, Cristhian.** adictosaltrabajo.com. [En línea] [Citado el: 3 de 12 de 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=EJB3vsSpring.. 7855-uusji.

23. XML.com. [En línea] O'Reilly Media, Inc. [Citado el: 4 de 12 de 2013.] http://www.xml.com.

24. Facelets y JSF. [En línea] [Citado el: 4 de 12 de 2013.] http://es.scribd.com/doc/44720276/Facelets-y-JSF.

25. **Guaita, Alvaro Martínez.** desarrolloweb.com. [En línea] [Citado el: 5 de 12 de 2013.] http://www.desarrolloweb.com/de_interes/hojas-resumen-xhtml-2961.html.

26. w3c.es. [En línea] [Citado el: 7 de 12 de 2013.] http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo.

27. **Alvarez, Miguel Angel.** desarrolloweb.com. [En línea] [Citado el: 7 de 12 de 2013.] http://www.desarrolloweb.com/articulos/25.php.

28. Drools. [En línea] [Citado el: 7 de 12 de 2013.] http://reglasnegocio.blogspot.com.

29. Oracle. [En línea] [Citado el: 7 de 12 de 2013.] http://docs.oracle.com/javase/tutorial/reflect.

30. Tecnologia y synergix. [En línea] [Citado el: 11 de 12 de 2013.] http://synergix.wordpress.com/2008/07/10/modelo-de-dominio.

31. Slideshare. [En línea] http://www.slideshare.net/maryme/2-requerimientos-del-software.

32. scribd. [En línea] [Citado el: 11 de 12 de 2013.] http://es.scribd.com/doc/37187866/Requerimientos-funcionales-y-no-funcionales.

33. sparxsystems. [En línea] [Citado el: 11 de 12 de 2013.] http://www.sparxsystems.com.ar/resources/tutorial/use_case_model.html.

34. **Larman, C.** UML y patrones, Tomo I Capítulos 18, Páginas 185-215. *UML y patrones.*

35. **Robaina, Irilis Ledón.** *Tesis de Diploma de Arquitectura de BioSyS.* La Habana, Cuba : s.n., 2008.

36. Slideshare. [En línea] [Citado el: 15 de 12 de 2013.] http://www.slideshare.net/jpbthames/diagramas-de-clases.

37. Merinde. [En línea] [Citado el: 15 de 12 de 2013.] http://merinde.net/index.php?option=com_content&task=view&id=495&Itemid=291.

38. Slideshare. [En línea] [Citado el: 15 de 12 de 2013.] http://www.slideshare.net/joshell/diagramas-uml-componentes-y-despliegue.

Anexos

Anexo 1. Guía de observación de la entrevista. Aspectos a tener en cuenta.

1. Correcta administración y seguridad de los procesos relacionados con la gestión de las Reglas del Negocio en el Sistema de Información Hospitalaria del Centro de Informática Médica con un nivel de detalle acorde con las necesidades existentes.
2. El módulo Configuración del Sistema de Información Hospitalaria del Centro de Informática Médica no cuentan con una configuración detallada y precisa que permita una mejor gestión de las Reglas del Negocio.
3. Para la navegación del usuario el Sistema de Información Hospitalaria del Centro de Informática Médica no cuenta con funcionalidades que permitan una mejor gestión de las Reglas del Negocio.
4. La gestión de los procesos relacionados con la gestión de las Reglas del Negocio en el Sistema de Información Hospitalaria del Centro de Informática Médica se realiza dependiendo de los módulos que administran esta información en el sistema.

Anexo 2. Guía con aspectos a considerar en las entrevistas.

Objetivo: obtener criterios acerca de las funcionalidades correspondientes a la gestión de las Reglas del Negocio en el Sistema de Información Hospitalaria del Centro de Informática Médica.

1. Características del módulo Configuración que deben prevalecer en las nuevas funcionalidades a desarrollar.
2. Características que no se encuentran en las funcionalidades asociadas a los procesos relacionados con la gestión de las Reglas del Negocio que actualmente existen en el Sistema de Información Hospitalaria del CESIM y que son necesarias agregarlas para el desarrollo de las nuevas funcionalidades.
3. Nuevos aspectos que deba manejar el sistema en los módulos relacionados con la gestión de las Reglas del Negocio.
4. Posible uso de las mismas tecnologías y herramientas usadas para desarrollar el Sistema de Información Hospitalaria del CESIM para gestionar las nuevas funcionalidades a agregar al sistema existente.

Anexo 3

Figura 12 Ubicación Interfaz Módulo Configuración

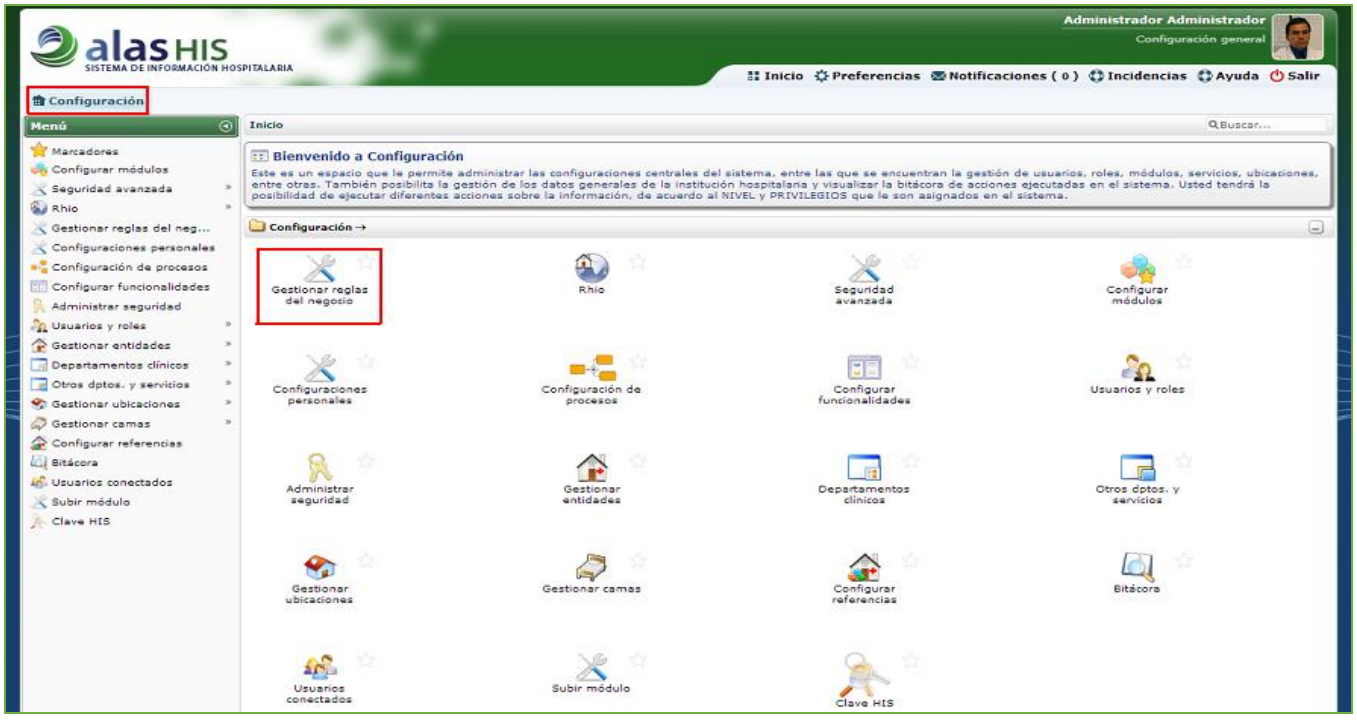


Figura 13 Interfaz Módulo Configuración

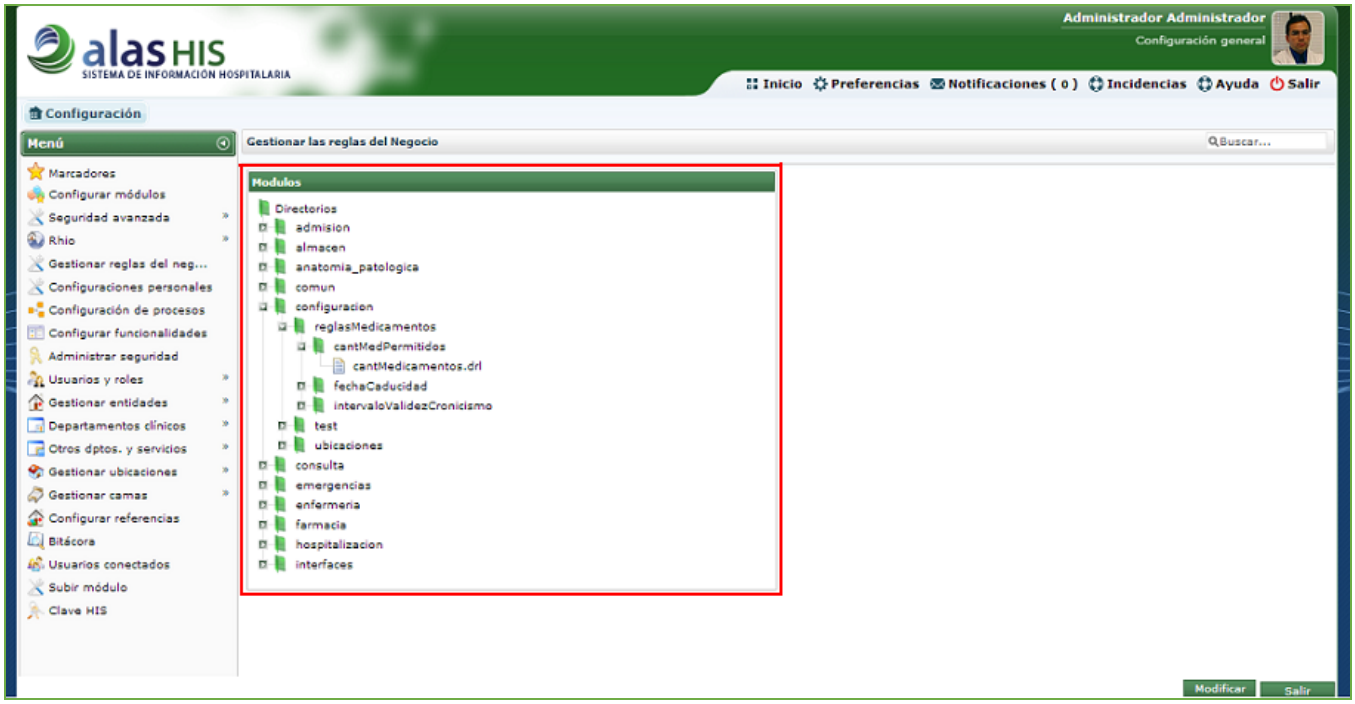


Figura 14 Interfaz Gestionar Reglas del Negocio

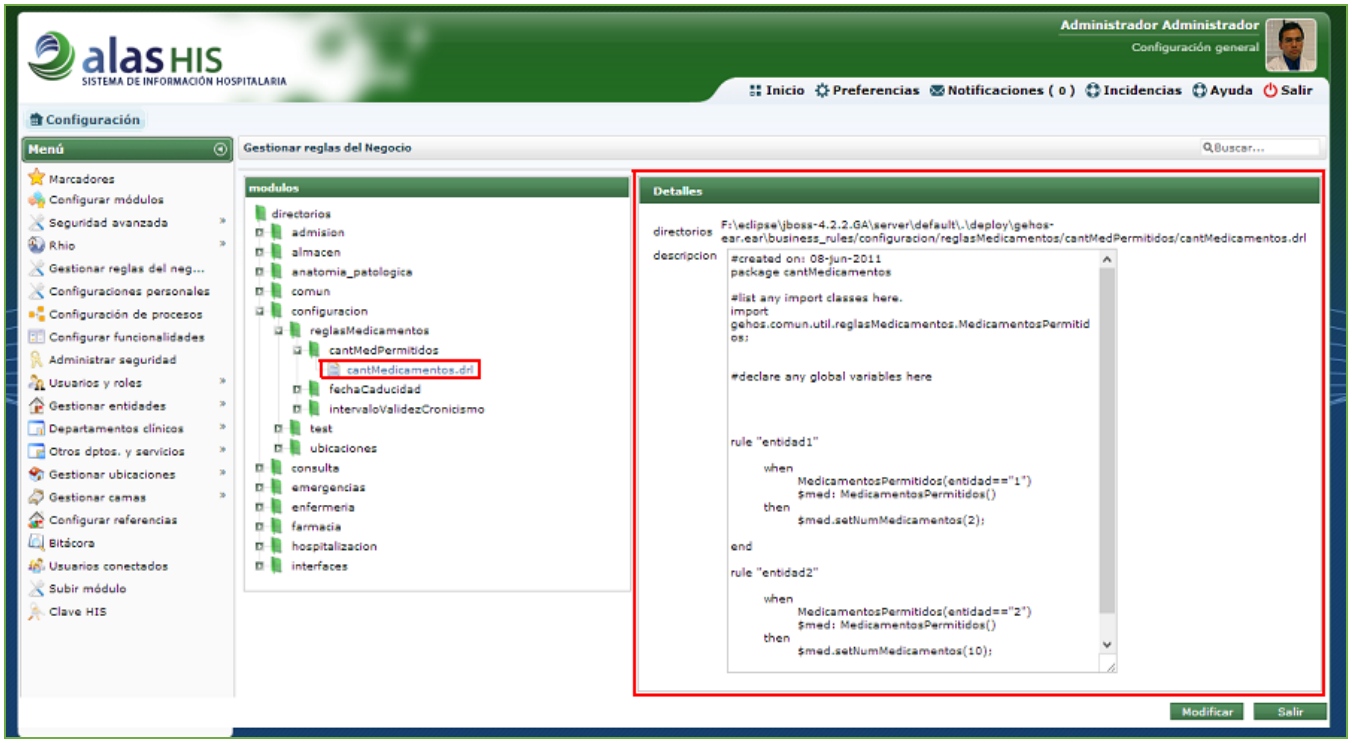


Figura 15 Interfaz Ver Regla del Negocio

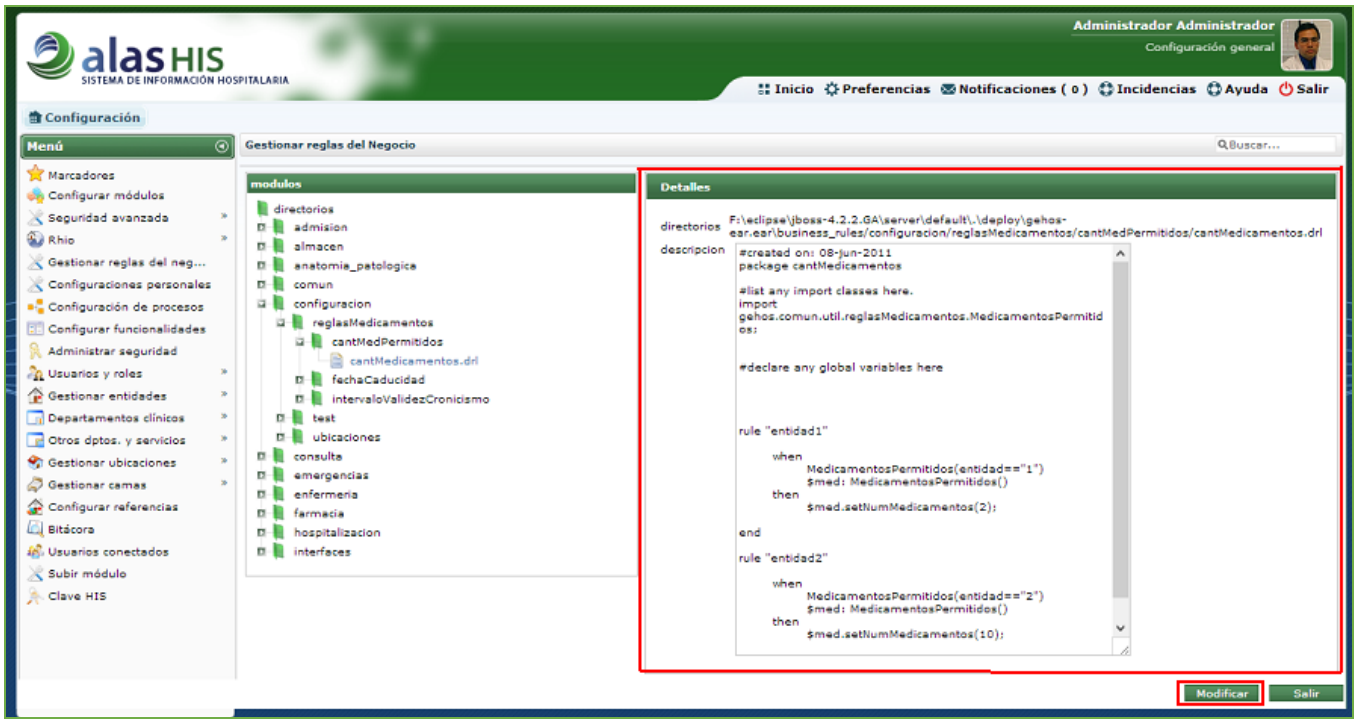


Figura 16 Interfaz Ver Regla del Negocio 2

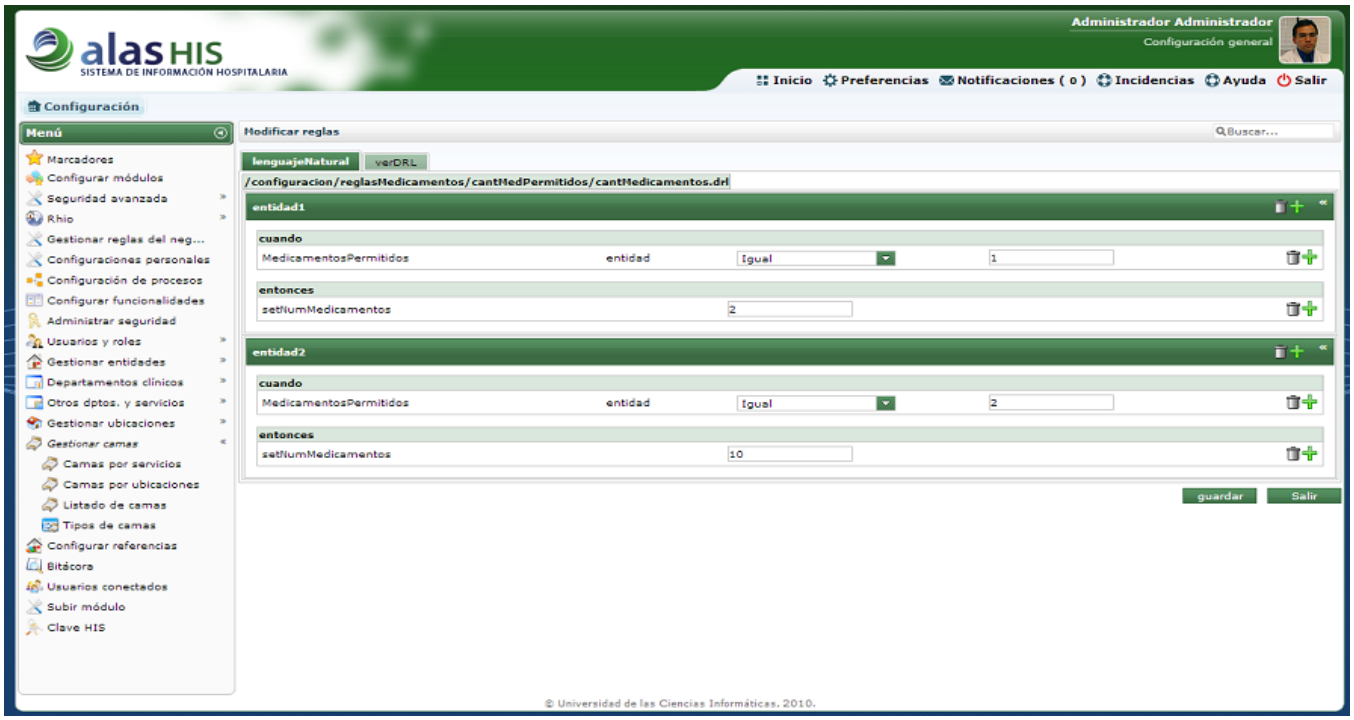


Figura 17 Interfaz Modificar Regla del Negocio

Glosario de Términos

Plug-ins: es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal.

Framework: es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

Caso a cado o ad-hoc: es una locución latina que significa literalmente "*para esto*". Generalmente se refiere a una solución específicamente elaborada para un problema o fin preciso y, por tanto, no generalizable ni utilizable para otros propósitos. Se usa pues para referirse a algo que es adecuado sólo para un determinado fin o en una determinada situación.

BRMS (Business Rules Management System): se refiere a los Sistemas de Gestión de Reglas del Negocio

HTML: (Hyper Text Markup Language): es el lenguaje para la representación de la información en la web (ver Web).

Templates: un template es un conjunto de archivos que determinan la estructura y el aspecto visual de un sitio web y tiene como ventaja principal disminuir tiempos y costos de desarrollo.

Algoritmo Rete: el algoritmo Rete (cuya pronunciación suele ser 'REET', 'REE-tee' o, en Europa, 're-tay' que viene de su pronunciación en Latín, dado que 'rete' significa red en Latín) es la base de diversas implementaciones más eficientes de sistemas expertos. Es un algoritmo de reconocimiento de patrones eficiente para implementar un sistema de producción de reglas.

MVEL: es un lenguaje de expresión y Runtime de tiempo de ejecución híbrido, de tipo dinámico-estático, e incrustable para la Plataforma Java. Se utiliza normalmente para exponer lógica básica para usuarios finales y programadores a través de configuración tal como archivos XML o anotaciones de Java. También puede utilizarse para analizar expresiones simples de JavaBean.

API: Interfaz de Programación de Aplicaciones, cuyo acrónimo en inglés es API (Application Programming Interface), es un conjunto de funciones residentes en bibliotecas generalmente dinámicas. Permiten que una aplicación corra bajo un determinado sistema operativo.

Multiplataforma: es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.

Plain Old Java Object (POJO): enfatiza el uso de clases simples y que no dependen de un framework en especial.

IDE: ambiente integrado de desarrollo (Integrated Development Environment). Es un conjunto de software que permite el desarrollo de aplicaciones.

CamellCasing: notación que utilizan algunos leguajes de programación para denotar variables y parámetros de una clase.

PascalCasing: notación que utilizan algunos leguajes de programación para denotar variables y parámetros de una clase.