Universidad de las Ciencias Informáticas

Facultad 6

TÍTULO: Sistema para la generación de reportes en la plataforma alasGRATO: Desarrollo del módulo "Reportador"

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores: Yanoski Agneri Martínez Hernández
José Rolando Lafaurie Olivares

Tutores: Dr. Ramón Carrasco Velar
MSc. Aurelio Antelo Collado
Ing. Julio Omar Prieto

Junio, 2008

"El genio es uno por ciento de inspiración y un noventa y nueve por ciento de dedicación."

Thomas Alva Edison

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 _____.

José Rolando Lafaurie Olivares

_____

Firma del Autor

Yanoski Agneri Martínez Hernández

_____

Firma del Autor

Dr. Ramón Carrasco Velar

_____

Firma del Tutor

M.Sc. Aurelio Antelo Collado

_____

Firma del Tutor

Ing. Julio Omar Prieto Entenza

_____

Firma del Tutor

DATOS DE CONTACTO

Tutores:

Ramón Carrasco Velar: Doctor en Ciencias Químicas, Investigador Titular.
Centro de Química Farmacéutica, Ciudad de la Habana, Cuba.
Email: ramon.carrasco@cqf.sld.cu
rcarrasco@uci.cu

Aurelio Antelo Collado: Ingeniero Industrial, Máster en Matemática Aplicada, Profesor Asistente con 4 años de experiencia.
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
Email: aantelo@uci.cu

Julio Omar Prieto Entenza: Ingeniero en Ciencias Informáticas, Profesor Instructor con 1 año de experiencia.
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
Email: jprieto@uci.cu

AGRADECIMIENTOS

Agradecemos infinitamente a nuestro siempre Comandante en Jefe Fidel, a Raúl y a la Revolución por darnos la oportunidad de estudiar, de superarnos y poder llegar a ser alguien en la vida.

A todos los profesores que a lo largo de la carrera nos han brindado su apoyo y ayuda para lograr este sueño de convertirnos en ingenieros.

A nuestros tutores Julio Omar, Aurelio y Carrasco por su ayuda en el desarrollo de este trabajo.

A la Universidad por ser nuestra casa durante estos 5 años.

A Dismey por donarnos algunas horas de su tiempo.

A todos los que de una forma u otra contribuyeron y nos apoyaron en el desarrollo de este trabajo.

A todos muchas gracias.

DEDICATORIA

Dedico esta tesis, ya que marca el fin y el comienzo de etapas trascendentales en mi vida, primeramente, a las cuatro mujeres que más yo quiero, mi madre querida , a mi abuela, a mi hermana, y mi novia , que como no tengo palabras de agradecimiento y cariño que pueda expresar como hombre, adolescente y niño . Solamente "Gracias por confiar en mi"

A mi hermano Pedro, por estar siempre tan preocupado por su hermano mayor. Te quiero.

A mi padre que siempre ha estado ahí, aunque siempre peleando, pero con sus buenos consejos.

A toda mi familia en general, por todo su apoyo.

También dedico todo este trabajo a mi compañero de tesis Yanoski, por ser tan persistente, y por haber confiado en mí como un hermano, jajaja ... te quiero man ... y no llores ... jaja.

A todos mis amigos y compañeros que han compartido junto conmigo estos 5 años que han hecho de verdad mi vida más interesante... para todos ellos muchas gracias...

José Rolando Lafaurie Olivares

A mi mamá, que ha sido más que eso para mí pues ha sido madre, padre y amiga al mismo tiempo. Por ser mi faro guía, mi ejemplo a seguir, mi fortaleza, mi fuente de inspiración. Por su amor, tú amor. Te quiero mucho mami.

A mi hermano Lesmy por su apoyo y preocupación.

A toda mi familia por sus consejos y la ayuda que me han brindado, a mi abuelo Ernesto, a mis tías Ana, Eida, China, Leonides y Juana María, a mis tíos Tomás, Mayito, Lázaro, a mis primas y primos Midiala, Ana Yelen, Irialis, Ernest, Addiel, Evanis, Evadis, a todos.

A mi otra familia: Mamá Elda, Papá Polito, Normita, Liuva y demás también por su gran apoyo.

A mi compañero de tesis José Rolando, por su dedicación y energía inagotable, por ser mí amigo y mí hermano.

A mis otros, aunque no de sangre, hermanos de siempre del Yagua: Alexei, Duniel, Yordán, Davichel, Yandriel, los que siempre han estado cada vez que los necesito.

A los nuevos amigos que cree en la Universidad: Yasmil, Yendry, José Rolando, Raúl (el flaco), Isbel, Luis (Dante), Lesniel, Richard, Soto, entre muchos otros con los que compartí buenos y malos momentos durante estos años en la Universidad.

A todos y todas los que me han apoyado y ayudado de una forma u otra a hacer posible este sueño.

Los quiero a todos(as).

Yanoski Agneri Martínez Hernández

RESUMEN

Los generadores de reportes son herramientas complementarias de los sistemas de información. Siendo su función principal permitir al usuario realizar, de forma transparente, consultas a la base de datos y obtener información de ella en un determinado formato.

En el presente trabajo se tiene como objetivo desarrollar una herramienta en el lenguaje JAVA que permita la generación dinámica de reportes para el proyecto productivo alasGRATO de la Facultad 6 de la Universidad de las Ciencias Informáticas.

Se emplea como metodología de desarrollo OpenUp/Basic. Los artefactos se generaron usando como lenguaje de modelado el Lenguaje Unificado de Modelado (UML) y auxiliados por el Visual Paradigm como herramienta de Ingeniería de Software Asistida por Computadora (CASE) y por el Eclipse como Entorno Integrado de Desarrollo (IDE).

Este generador cuenta con una serie de características importantes para construir y guardar reportes; además puede generar informes con información extraída de diversos gestores de bases de datos.

PALABRAS CLAVES

Reportes, generador de reportes, base de datos.

ABSTRACT

Report generators are complementary tools of the information systems. Being its main function to allow the user, in a clear way, make queries to the database and obtain information from it in a certain format. The present work aims at developing a tool in Java language that allows dynamic generation of reports for the Productive Project "alasGRATO" of faculty 6 in the University of Informatics Sciences.
It is used as development methodology OpenUp/Basic. The artifacts were generated using as modeling language Unified Modeling Language (UML) and assisted by Visual Paradigm as a tool of Computer Aided Software Engineering (CASE), and by Eclipse as Integrated Development Environment (IDE).
This generator features a number of important characteristics to create and keep reports, it can also generate reports with information provided by different Database Management Systems.

KEYWORDS

Reports, reports generator, database

TABLA DE CONTENIDO

INTRODUCCIÓN

El presente trabajo está enmarcado dentro del proyecto de investigación alasGRATO que tiene como producto final la aplicación de igual nombre, la cual es una herramienta para la predicción de actividad farmacológica usando la teoría de grafos. La misma cuenta con un gran volumen de información, referente a la estructura y actividad biológica de diferentes compuestos, utilizado por especialistas en Química Medicinal, a quienes va dirigida la plataforma, para su estudio y creación de nuevos fármacos.

Estos especialistas no poseen necesariamente los conocimientos informáticos que les permita acceder a la información contenida en las bases de datos estructurales y de actividad biológica con las que trabajan y además, no tienen conocimientos de la estructura de las mismas. Lo anteriormente planteado conduce a la formulación del siguiente **problema**:

¿Cómo extraer la información contenida en la Base de Datos, de forma que sea innecesario para el usuario no especializado, el conocimiento de términos informáticos?

Por lo que se define como **objeto de estudio:**

Las herramientas utilizadas para realizar reportes a las bases de datos.

La plataforma alasGRATO está implementada sobre el lenguaje de programación JAVA, por lo que se formula como **campo de acción**:

Herramientas que generan reportes empleando la plataforma JAVA.

Se trazó como **objetivo general:**

Desarrollar un módulo para la plataforma alasGRATO capaz de generar reportes.

Para dar cumplimiento al objetivo general se definieron como **objetivos específicos:**

- Diseñar un módulo que permita generar reportes.

- Implementar el modulo diseñado.
- Integrar el módulo a la plataforma alasGRATO.
- Realizar pruebas exploratorias al módulo.

Para alcanzar dichos objetivos se planteó desarrollar las siguientes **tareas**:

- Análisis del estado del arte acerca de sistemas para la generación de reportes desarrollados.
- Definición de los requisitos funcionales y no funcionales de la aplicación.
- Elaboración del modelo conceptual.
- Desarrollo del diagrama de casos de uso del sistema.
- Desarrollo del diagrama de clases del diseño.
- Implementación del diagrama de clases del diseño.

El presente documento está estructurado en tres capítulos:

Capítulo 1: Fundamentación Teórica.

En este capítulo se lleva a cabo un estudio y análisis de varias herramientas de generación de reportes en el mundo y en la Universidad, se selecciona la metodología de desarrollo a utilizar y se describen los roles a desempeñar, así como, los artefactos a obtener. Además, se seleccionan los patrones de software, el lenguaje de programación y las herramientas empleadas en el desarrollo de la solución.

Capítulo 2: Características del Sistema.

En este capítulo se realiza el diagrama del modelo conceptual propuesto y se definen los requerimientos a implementar en la aplicación. Además, se realiza el diagrama de casos de usos del sistema y la descripción de estos.

Capítulo 3: Diseño del Sistema.

En este capítulo se desarrolla el diagrama de clases del diseño así como la realización de los diagramas de secuencia y el modelo de despliegue. Se realizan las descripciones de las principales clases y el uso de los patrones de software elegidos para el desarrollo de la aplicación.

Capítulo 4: Implementación del Sistema.

En este capítulo se muestran los diagramas de componentes que se definieron durante la implementación de la aplicación, se describen implementaciones relevantes y se muestran algunas pantallas de la aplicación. Además, se detallan pruebas realizadas sobre la aplicación para comprobar sus funcionalidades.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se hace un estudio acerca del estado de los sistemas de generación de reportes en la Universidad de las Ciencias Informáticas y en el resto del mundo. Se analizan varias herramientas haciendo referencia al proceso de desarrollo de software así como las metodologías de desarrollo, tecnologías y tendencias actuales que pueden ser útiles en la propuesta de solución. Sobre la base de la metodología seleccionada se realiza un estudio de los principales artefactos que se obtienen, así como los roles que intervienen caracterizándose además los lenguajes y herramientas empleados en el desarrollo del sistema.

1.1 Sistemas de generación de reportes

Antiguamente los reportes se confeccionaban en formato duro, ya que no existía una herramienta capaz de realizarlos. En la actualidad el empleo del formato digital ha venido ganando terreno debido a su facilidad y la presencia de aplicaciones que son capaces de generarlos automáticamente. Dichos programas, denominados generadores de reportes, permiten obtener la información deseada con mayor rapidez ofreciéndoles a los trabajadores calificados un mejor nivel de detalle y flexibilidad, además de la capacidad de interactuar con los resultados obtenidos basándose en los datos para tomar sus propias decisiones.

1.1.1 Herramientas de generación de reportes en el mundo

En el mundo del software existen varias herramientas para la generación dinámica de reportes, tales son los casos de Active Reports, Agata Report, Crystal Reports, IdeaReports, Jasper Reports y Dynamic Jasper Reports cuyas principales características son:

1.1.1.1 Active Reports

El Active Reportes es una herramienta de plataforma propietaria caracterizándose por soportar gráficos de tipo 2D ó 3D, imágenes y sub-reportes. Es capaz de emplear una amplia variedad de orígenes de

datos, cuenta con un lenguaje de programación propio. Permite la adición de nuevos reportes sin necesidad de recompilar la aplicación dando soporte tanto a aplicaciones web como de escritorio y exporta los reportes en diferentes formatos como PDF, XML y HTML. [1]

1.1.1.2 Agata Report

El Agata Report es una herramienta libre, liberada bajo la licencia *GNU GPL* y escrita en *PHP GTK* que se caracteriza por ser multiplataforma (Windows y Linux), por soportar gráficos, imágenes y sub-reportes, así como varios orígenes de datos. Puede exportar los reportes en diferentes formatos como HTML, XML, PDF y CSV. Permite establecer los niveles de datos, subtotales y totales para el reporte. Da la posibilidad de generar un diagrama de Entidad relación ER a partir de la base de datos y está diseñado solo para Web usando PHP. [2]

1.1.1.3 Crystal Reports

El Crystal Reports es una herramienta propietaria, incluida en el Visual Studio desde 1993, que soporta gráficos, imágenes, sub-reportes y una gran variedad de orígenes de datos. Da soporte para aplicaciones Web y de escritorio. Permite exportar los reportes en diferentes formatos como PDF, XML y HTML. Elabora los reportes en tiempo de diseño pues está dirigida más bien para el programador y no para el usuario, además para modificar el reporte es preciso recompilar el proyecto donde se encuentra. [3]

1.1.1.4 IdeaReports

El IdeaReports es una herramienta desarrollada en 2005 por la compañía InterSoft sobre plataforma JAVA, es decir, plataforma libre. Está compuesto de dos partes: Módulo IdeaReports Programador y Módulo IdeaReports Usuario Final. Soporta gráficos, imágenes y sub-reportes, además de gran variedad de orígenes de datos. Es independiente de la plataforma ya que esta desarrollado en JAVA. Permite exportar los reportes en diferentes formatos como Excel, PDF y HTML. Permite el armado automático de la sintaxis de las consultas y la selección mediante pop-ups de distintos esquemas, tabla y campos a consultar. Posibilita crear automáticamente uniones (joins) entre las distintas tablas

seleccionadas y seleccionar múltiples condiciones de búsqueda, pudiendo generar consultas parametrizadas. En tiempo de ejecución visualiza un formulario solicitando los parámetros de la consulta. También almacena las consultas para su posterior utilización y admite el ordenamiento de la información y las restricciones de acceso a esquemas, tablas y campos (por columnas). [4]

1.1.1.5 Jasper Reports

El Jasper Reports es una herramienta desarrollada sobre plataforma JAVA. Puede ser usada por cualquier aplicación desarrollada en esa plataforma, incluyendo J2EE o aplicaciones Web. Puede usar gran variedad de origen de datos, lo que posibilita extender las capacidades de crear reportes para casi cualquier tercera aplicación. Permite múltiples fuentes de información de múltiples tipos en un mismo reporte. Exporta a diferentes formatos como HTML, PDF, XML, XLS, RTF, CSV y TXT. Soporta imágenes y maneja fácilmente sub-reportes. Integrado con la librería JFreeChart, escrita igualmente en JAVA, puede generar gráficas. Además, es más orientada al desarrollador. [5]

1.1.1.6 Dynamic Jasper Reports

El Dynamic Jasper Reports es una herramienta desarrollada sobre plataforma JAVA que mantiene las principales características del Jasper Reports ya que trabaja con él reduciendo su complejidad para reportes simples y de mediana complejidad ayudando a los desarrolladores a ahorrar tiempo cuando los diseñan. Añade un mayor grado de dinamismo a los reportes en tiempo de ejecución. Soporta gran variedad de orígenes de datos, gráficos, imágenes y sub-reportes, que pueden ser creados dinámicamente y exporta a diferentes formatos como PDF, HTML y XLS. Es una API (Application Programming Interface o Interfaz de Programación de Aplicaciones) amigable, intuitiva, robusta y estable. Permite definir en tiempo de ejecución las columnas, controlando su posicionamiento, ancho, título. Así como, crear grupos usando expresiones simples como criterios o expresiones complejas personalizadas, los cuales pueden tener variables que contengan el resultado de operaciones (Suma, Conteo u otras facilitadas por el Jasper) realizadas sobre determinados campos (columnas). Dado un conjunto mínimo de opciones definidas se encarga de diseñar el reporte y admite generar el mismo para distintos tamaños de páginas y orientación. Además, da la posibilidad de ensamblar en un mismo reporte informes de distinta naturaleza y añadir fácilmente gráficos simples. [6]

1.1.2 Herramientas para la generación de reportes en la Universidad

En la Universidad de Ciencias Informáticas como parte de proyectos productivos se han desarrollado varias herramientas para la generación de reportes en busca de una mejor interacción de los clientes de dichos proyectos con la información almacenada en cada una de sus bases de datos. Ejemplo de esas herramientas son los casos del Olympia y Akademos.

1.1.2.1 Olympia

Olympia es una aplicación desarrollada sobre el framework Symfony y escrita en PHP. Es multiplataforma. Soporta imágenes, gráficas y sub-reportes, así como varios orígenes de datos. Proporciona a los usuarios, entre otras opciones, agilizar la toma de decisiones, generar reportes en varios formatos y con gran variedad de opciones en su diseño, marcando una diferencia entre los reportes tradicionales y los reportes dinámicos, objetos de este producto. Está compuesto por varias aplicaciones entre las que se encuentran el Visor de reportes, Diseñador de modelos y el Diseñador de reporte. Aunque permite abstraerse en parte de los conocimientos relacionados con los gestores de bases de datos, el usuario aún debe poseer conocimientos básicos. Además, se entorno de trabajo está estructurado de forma que es difícil guiarse en el creación y generación de reportes.

1.1.2.2 Akademos

Akademos es una aplicación desarrollada en plataforma .NET, en el 2006, en la Universidad de las Ciencias Informáticas, la cual brinda actualmente importantes servicios académicos tanto a los estudiantes como a los departamentos docentes. Cuenta con una herramienta de generación de reportes, la cual los genera con eficiencia, pero desde el punto de vista del dinamismo de la generación, se encuentra adaptada solamente al negocio de la gestión académica en la UCI, restringiendo que sean elaborados con la información contenida en la base de datos específicamente utilizada por el software, lo cual dificulta que esta herramienta sea de propósito general. Además, está diseñada sólo para la Web.

1.1.3 Conclusiones parciales

Las herramientas anteriormente analizadas tienen funcionalidades similares, con pocas diferencias, aunque fueron creadas para un mismo fin, apoyar a los usuarios a elaborar reportes informativos a partir de las bases de datos y agilizar el tiempo de desarrollo de las aplicaciones.
Algunas son desarrolladas para la Web como son los casos de Olympia, Akademos y el Agata Report; otras son software propietario, siendo sus licencias costosas, lo que representa una gran barrera desde el punto de vista de la situación política y económica de nuestro país, tal es el caso del Crystal Reports, Active Reports y el IdeaReports. Varias presentan el inconveniente que para modificar el reporte es necesario recompilar el proyecto donde se encuentra y otras están principalmente dirigidas al programador y no al usuario final como Jasper Reports, Agata Report y Crystal Reports.

Sin embargo, se eligió para su uso la herramienta Dynamic Jasper Reports gracias a las ventajas que aporta al tomar la mayoría de sus funciones del Jasper Reports y agregarle otras para facilitar la creación de reportes y disminuir su complejidad. Brinda además la posibilidad de ensamblar varios reportes en uno y da soporte a aplicaciones de escritorio.

1.2 Metodología de desarrollo a utilizar

Una metodología de desarrollo de software indica paso a paso todas las actividades a realizar para lograr el producto informático deseado, indicando además qué personas deben participar en el desarrollo de las actividades y qué papel deben tener. Detalla la información que se debe producir como resultado de una actividad y la necesaria para comenzarla.

La metodología a utilizar en el desarrollo del presente trabajo es OpenUp/Basic según se definió en la arquitectura definida para el proyecto alasGRATO. [7]

1.2.1 OpenUp/Basic

OpenUp/Basic [8] fue liberado por Eclipse Process Framework (EPF). Se construyó sobre una donación realizada por IBM del Basic Unified Process. Fue entregada a Eclipse a fines de 2005 y renombrado como Proceso Unificado Abierto (OpenUp) en 2006. Es un marco de trabajo para

procesos de desarrollo de software.

Conserva las características principales del modelo de desarrollo RUP, incluye el desarrollo iterativo, permite identificar los requisitos operacionales del sistema, prever las interacciones con los usuarios y prevenir los posibles riesgos en el desarrollo del sistema.

Es una forma de desarrollo ágil y ligera, consiste en equipos a los cuales se les asigna una fase del desarrollo que tienen que complementarse entre sí para obtener un buen producto final, no puede ser una sola persona la que realice todo el trabajo pues esto podría ocasionar que se pierda de vista ciertas características importantes, por ejemplo para un proyecto pequeño constituyen equipos de tres a seis personas e implican tres a seis meses de esfuerzo del desarrollo. Describe el conjunto mínimo de roles, y para cada uno un conjunto de tareas a realizar y artefactos que obtendrán en el desarrollo del software. [9]

1.3 Roles desempeñados

De acorde a la metodología seleccionada para el desarrollo de la solución del sistema, se definieron los roles a ser desempeñados partiendo de los roles que propone la misma. Entre ellos se encuentra: [9]

1.3.1 Analista

La persona en este papel representa el cliente y usuario final se refiere a la recopilación de las aportaciones de los interesados para entender el problema a resolver y por la captura y el establecimiento de las prioridades para las necesidades.

Según OpenUp, el Analista necesita poseer experiencia en la identificación y compresión de los problemas y oportunidades, así como la capacidad de articular las necesidades que están asociadas con el problema clave a resolver o la oportunidad de ser realizado, la habilidad para colaborar eficazmente con el equipo ampliado de colaboración a través de sesiones de trabajo, talleres, sesiones JAD y otras técnicas y buenas habilidades de comunicación, verbal y por escrito. Además, debe tener conocimiento de los ámbitos de negocio y la tecnología o la capacidad de absorber y comprender

rápidamente esa información.

Este Rol puede ser asignado de la siguiente manera:

- Uno o más miembro del equipo realiza esta función exclusivamente. Este enfoque comúnmente adoptado es adecuado para las necesidades complejas que son difíciles de reunir.
- Uno o más miembros del equipo cumple esta función y las pruebas del software. Esta es una buena opción para los más pequeños o con recursos limitados equipos de prueba.

1.3.2 Desarrollador

Es el responsable para el desarrollo de una parte del sistema, incluido diseñarlo para ajustar en la arquitectura, posibles prototipos de interfaz de usuario y, a continuación, la implementación, la unidad de pruebas, y la integración de los componentes que forman parte de la solución.

Según OpenUp, el desarrollador necesita conocimientos técnicos y experiencia suficiente para definir y crear soluciones técnicas dentro del proyecto. Debe tener la capacidad de comprender y ajustarse a la arquitectura, de identificar y construir las pruebas de desarrollador que cubran el requerido comportamiento de los componentes técnicos. Además, debe tener la habilidad para comunicar el diseño de una manera que otros miembros del equipo entiendan.

Además, para crear un modelo visual del sistema, la persona en este papel necesita la capacidad de hacer el diseño en el Lenguaje de Modelado Unificado (UML).

La persona que desempeñe este rol puede tener conocimientos especializados en un área técnica, pero también deben tener un amplio conocimiento de todas las tecnologías que intervienen para poder trabajar con otros miembros del equipo técnico.

Incluso en el más pequeño equipo, varios individuos deberían trabajar juntos para crear la solución técnica. En los pequeños, ágiles equipos esta función suele ser compartida entre varios de sus miembros que también realizan otras funciones.

1.4 Artefactos generados

En dependencia de la metodología y los roles definidos a ser utilizados en el desarrollo de la solución del sistema se generan los siguientes artefactos: [9]

1.4.1 Artefactos a ser generados por el Analista

1.4.1.1 Requerimientos del Sistema

Este artefacto captura de todo el sistema las necesidades que no capturados en los escenarios o casos de uso, incluidos los requisitos de los atributos de calidad y los requisitos funcionales.

Es utilizado con los fines siguientes:

- Describir los atributos de la calidad del sistema y las limitaciones que las opciones de diseño deben satisfacer para cumplir las metas del negocio, sus objetivos o capacidades.
- Capturar los requisitos funcionales que no son expresados como casos de uso.
- Negociar entre, y seleccionar, opciones de diseño que compiten.
- Evaluar el tamaño, costo, y la viabilidad del sistema propuesto.
- Entender el nivel de los servicios necesarios para la gestión operativa de la solución.

1.4.1.2 Casos de Uso

Este artefacto captura el comportamiento del sistema para producir un resultado observable de valor para quienes interactúan con el sistema.

Casos de uso se utilizan para los siguientes fines:

- Para llegar a un entendimiento común del comportamiento del sistema.
- Para el diseño de los elementos que apoyan el comportamiento requerido.
- Para identificar casos de prueba.
- Para planificar y evaluar el trabajo.
- Para escribir la documentación de usuario.

1.4.1.3 Modelo de Casos de Uso

Este artefacto captura un modelo de las funciones y el ambiente del sistema y sirve como un contrato entre el cliente y el equipo.

Presenta un panorama general del comportamiento del sistema. Es la base para el acuerdo entre las partes interesadas y el equipo de proyecto en cuanto a la funcionalidad del sistema. Asimismo, guía diversas tareas en el ciclo de vida de desarrollo de software.

1.4.2 Artefactos a ser generados por el Desarrollador

1.4.2.1 Estructura

Una versión operativa de un sistema o parte del sistema que muestra un subconjunto de las capacidades que se incluirá en el producto final.

Su propósito es ofrecer un valor incremental para el usuario y cliente, y proporcionar un artefacto comprobable para la verificación.

Esta versión del sistema es el resultado de poner la implementación del sistema a través de un proceso de construcción (por lo general, un script automatizado) que crea una versión ejecutable del sistema, o una que se ejecuta. Esta versión ejecutable del sistema suele tener un número de archivos auxiliares que también se consideran parte del artefacto compuesto.

1.4.2.2 Diseño

Este artefacto describe la realización de las funcionalidades del sistema requeridas y sirve como una abstracción del código fuente.

Su propósito es describir los elementos del sistema para que puedan ser examinados y entendidos en

un sentido que no sea posible por la lectura del código fuente.

Este producto puede describir múltiples puntos de vista estático y dinámico del sistema a examinar. Aunque, diversos puntos de vista pueden centrarse en divergencias, aparentemente cuestiones independientes de cómo el sistema se integre y trabaje, deberían ajustarse sin contradicción.

Describe los elementos que componen el sistema. Comunica abstracciones de determinadas porciones de la implementación y puede describir un subsistema encapsulado, un alto nivel de análisis, un punto de vista del sistema en un solo contexto, u otras perspectivas que explican la solución a un problema específico que necesita ser comunicado.

1.4.2.3 Implementación

Archivos de código del Software, de datos y de apoyo (tales como archivos de ayuda en línea) que representan las partes de un sistema que pueda ser construido.

Su propósito es representar las partes físicas que componen el sistema a ser construido, y organizar las partes de una forma que sea comprensible y gestionable.

Este artefacto es la colección de uno o más de estos elementos:

- Archivos de código fuente.
- Archivos de datos.
- Crear scripts.
- Otros archivos que se transforman en el ejecutable del sistema.

1.5 Patrones de Software

El planteamiento de formalizar soluciones a distintas situaciones, de modo que puedan ser entendidas por otros profesionales, es lo a lo que se llama patrones. Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto, es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto. [10]

Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Capturan la experiencia existente y probada para promover buenas prácticas. Permiten y han permitido en diferentes áreas del conocimiento humano rehusar la esencia de la solución de un problema al enfrentar nuevos problemas similares.

1.5.1 Patrones de Arquitectura

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Los patrones de arquitectura representan el nivel más alto en el sistema de patrones. Ayudan a especificar la estructura fundamental de una aplicación. Cada actividad de desarrollo es gobernada por esta estructura; por ejemplo, el diseño detallado de los subsistemas, la comunicación y colaboración entre diferentes partes del sistema. Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global. [11]

1.5.1.1 Modelo de tres Capas

En el modelo de 3 capas cada nivel tiene un tipo de abstracción diferente al de los demás, aunque cada capa tiene comunicación con las demás: [12]

- Capa de presentación: esta es la que el usuario puede ver en su ordenador, es donde se tratan los datos que se van a mostrar. Se intenta que en esta haya el mínimo de procesamiento y se comunicará solamente con la capa de negocio.
- Capa de lógica del negocio: en esta se encuentra la lógica, se recibe las peticiones del usuario, y tras ejecutar una acción se le envía las respuestas del proceso. Se comunica como se ha dicho con la de presentación, la cual le envía peticiones y esta le responde con los resultados y con la capa de acceso a datos, para pedirle datos.
- Capa de datos: Esta es la encargada de abstraer los datos a utilizar en la aplicación pudiéndose utilizar uno o más sistemas gestores de bases de datos (SGDB). Recibe peticiones

desde la capa de negocio.

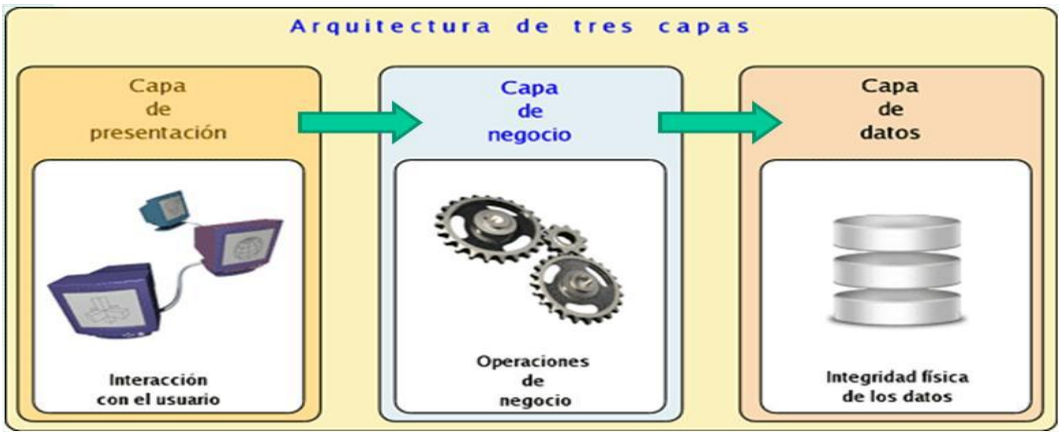


Figura 1: Representación del patrón Modelo de tres Capas.

1.5.2 Patrones de Diseño

Una solución a un problema que se usa repetidamente en contextos similares con algunas variantes en la implementación es a lo que se define como un patrón de diseño.

Un patrón de este tipo identifica, abstrae y nombra los aspectos elementales de una estructura de diseño, donde los componentes, son las clases y objetos, y sus mecanismos de interacción son mensajes.

Cada patrón prescribe una estructura de clases, sus roles y colaboraciones, y una adecuada asignación de métodos para resolver un problema de diseño en una manera flexible y adaptable.

Ayudan a elegir diseños alternativos que hacen un sistema reutilizable y evitan alternativas que comprometan la reutilización.

1.5.2.1 GRASP

GRASP son patrones generales de software para asignación general de responsabilidades, es el acrónimo de "*General Responsability Assignment Software Patterns*". [13]

Se destacan 5 patrones principales que son:

- **Experto**

Experto en información es el principio básico de asignación de responsabilidades. Indica que la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Mantiene el encapsulamiento, los objetos usan su propia información para llevar a cabo sus tareas, soportando un bajo acoplamiento que favorece el hecho de tener sistemas más robustos y de fácil mantenimiento. Distribuye el comportamiento entre las clases que contienen la información requerida, brindando una alta cohesión.

- **Creador**

Identifica quien debe ser el responsable de la creación o instanciación de nuevos objetos o clases. Se brinda un soporte al bajo acoplamiento.

- **Alta Cohesión**

Plantea que la información que almacena una clase debe ser coherente y estar en la mayor medida relacionada con la clase. Estas clases mejoran la claridad y la facilidad con que se entiende el diseño. Simplifican el mantenimiento y las mejoras en funcionalidad. A menudo genera un bajo acoplamiento, además la ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización ya que una clase muy cohesiva puede destinarse a un propósito en específico.

- **Bajo Acoplamiento**

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las

clases. Estas clases no se afectan por los cambios en otros componentes, fáciles de entender por separado y de fácil reutilización.

- **Controlador**

Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Le ofrece mayor potencial de los componentes reutilizables, garantizando que la empresa o los procesos de dominio sean manejados por la capa de los objetos de dominio y no por la de la interfaz. Reflexionar sobre el estado del caso de uso.

1.5.2.2 DAO

Un objeto de acceso a datos o en inglés *Data Access Object* (DAO) [14] da resumen y encapsula todos los accesos a la fuente de datos suministrando una interfaz común entre esta y la aplicación. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

Este patrón implementa el mecanismo de acceso requerido para trabajar con la fuente de datos. La fuente de los datos puede ser un almacenamiento persistente como un Sistema Gestor de Bases de Datos, un servicio externo como un intercambio B2B, un repositorio como una base de datos LDAP (Protocolo Ligero de Acceso a Directorios o *Lightweight Directory Access Protocol*). El componente de negocio que se basa en el DAO utiliza el interfaz más simple expuesto por este para sus clientes. Oculta completamente la fuente de datos de los detalles de la ejecución. Debido a que la interfaz expuesta por el objeto de acceso a datos a los clientes no cambia cuando ocurren cambios en la implementación de las fuentes de datos subyacente, este patrón le permite adaptarse a diferentes sistemas de almacenamiento sin que ello afecte a sus clientes o los componentes del negocio. Esencialmente, actúa como un adaptador entre el componente y la fuente de datos. [14]

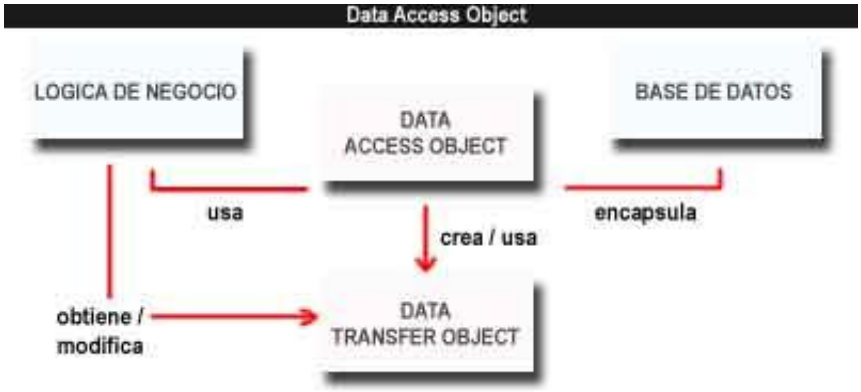


Figura 2: Representación del patrón DAO.

Se utilizará en el desarrollo del presente trabajo como patrón arquitectónico el modelo de tres capas ya que facilita que se pueda descomponer la aplicación en varios niveles de abstracción proporcionando la evolución del sistema, ya que los cambios sólo deben afectar a la capa donde se encuentre la modificación. Si la interfaz accede a la misma función, no se repetirá código lo que conlleva la ventaja de una mayor facilidad de mantenimiento de la aplicación. Las interfaces ya no acceden de forma independiente a los datos pues no se accede a través de ellas al modificar los datos, esto implica que no se nos mostrará diferencia alguna.

Además se usara el patrón de diseño DAO, pues su uso brinda como ventaja que cualquier objeto de negocio (aquel que contiene detalles específicos de operación o aplicación) no requiera el conocimiento directo del destino final de la información que manipula y los patrones GRASP, debido a que fue una decisión de la dirección del proyecto alasGRATO, del cual esta solución representa un módulo, al definirlo en su arquitectura además por sus características que se ajustan a las necesidades para el desarrollo de la solución.

1.6 Lenguaje de programación y herramientas utilizadas en el desarrollo de la solución

Para el desarrollo de una aplicación es sumamente importante una buena selección del lenguaje de programación y de las herramientas a utilizar ya que determinan el tiempo de desarrollo, los costos y el resultado general de todo el proceso.

Las herramientas y el lenguaje de programación a utilizar en el desarrollo del presente trabajo son Visual Paradigm como herramienta CASE, Eclipse como entorno integrado de desarrollo y JAVA como lenguaje de programación según se definió en la arquitectura definida para el proyecto alasGRATO. [7]

1.6.1 Herramienta CASE

El concepto de *CASE* (*Computer Aided Software Engineering* o por su traducción al español: Ingeniería de Software Asistida por Computadora) es muy amplio; se pudiera definir genéricamente como la aplicación de métodos y técnicas a través de las cuales las personas pueden modelar o diseñar sistemas por medio de programas, procedimientos y su respectiva documentación.

Las *herramientas CASE* son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de los mismos en tiempo y dinero. Son usadas en algunas fases del desarrollo de sistemas de información, incluyendo análisis, diseño y programación. [15]

1.6.1.1 Visual Paradigm para UML:

Es una herramienta de modelado que emplea UML, diseñada para apoyar a los arquitectos de sistema, desarrolladores y diseñadores para acelerar el proceso de análisis y diseño de aplicaciones complejas. [16]

Además, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, ingeniería inversa, generar código desde diagramas y generar documentación.

También ofrece un diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad, el uso de un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación, un modelo y código que permanece sincronizado en todo el ciclo de desarrollo. Disponibilidad en múltiples plataformas y versiones, así como de integrarse en los principales IDE.

1.6.2 Lenguaje de programación

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; permitiendo al desarrollador comunicarse con los dispositivos de hardware y software existentes. [17]

1.6.2.1 JAVA:

Es un lenguaje de programación desarrollado por *Sun Microsystems* a principios de los 90. Está diseñado para ser lo suficientemente simple permitiendo que los programadores puedan lograr fluidez con el lenguaje. Trabaja con sus datos como objetos y con interfaces a estos, además proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando. [18]

Sus principales características son:

- Orientado a objetos: soporta las características esenciales del paradigma de la programación orientado a objetos: encapsulación, herencia y polimorfismo.
- Robusto: elimina el uso de apuntadores para referenciar áreas de memoria, además libera al desarrollador de la necesidad de desalojar la memoria que la aplicación ya no usa. Además, requiere la declaración explícita tanto de los tipos de datos como de los métodos.
- Multiplataforma: el mismo código Java que funciona en un sistema operativo, funciona en cualquier otro sistema operativo que tenga instalada la máquina virtual de Java.
- Multitareas: A pesar de que las capacidades multitarea que pueden ser implementadas en Java dependen en gran parte del sistema operativo en el cual se ejecuten, digamos Windows o Unix, dichas capacidades superan en gran manera a los entornos de flujo único que ofrecen otros lenguajes de programación. Permite la ejecución concurrente de varios procesos ligeros o hilos de ejecución.

1.6.3 Entorno Integrado de Desarrollo

Un Entorno Integrado de Desarrollo o en inglés *Integrated Development Enviroment* (IDE) es un programa compuesto por un conjunto de herramientas para un desarrollador que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). [19]

1.6.3.1 Eclipse:

En su web oficial se define a Eclipse como *"An IDE for everything and for nothing in particular"* (un IDE para todo y para nada en particular). Eclipse es una de las herramientas que se engloban bajo el denominado Proyecto Eclipse. [20]

Es una plataforma basada en Java y de tipo código abierto. Este IDE ofrece el control del editor de código, del compilador y del depurador desde una única interfaz de usuario. Su misión consiste en evitar tareas repetitivas, facilitar la escritura correcta del código, disminuir el tiempo de depuración e incrementar la productividad del desarrollador.

1.7 Conclusiones

En este capítulo se realizó un estudio de los sistemas de generación de reportes en el mundo y en la Universidad de las Ciencias Informáticas decidiéndose hacer uso de la herramienta Dynamic Jasper Reports por sus ventajas para el desarrollo de la solución. Se decidió utilizar como metodología de desarrollo OpenUp, Visual Paradigm como herramienta CASE, como lenguaje de programación JAVA y el entorno integrado de desarrollo Eclipse. También se tomó la decisión de usar los patrones arquitectónicos Modelo de 3 capas, de diseño GRASP y DAO. Además, sobre la base de la metodología elegida, se expuso como roles a desempeñar el de Analista a partir del cual se obtendrán los artefactos Requerimientos del Sistema, Casos de Uso y el Modelo de Casos de Uso y el rol de Desarrollador el cual generará los artefactos Estructura, Diseño e Implementación.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

Este capítulo constituye la propuesta de solución del trabajo. Para describir la solución se realiza el modelo conceptual definiendo los diversos elementos que serán manejados así como las relaciones que entre estos se establecen. Se define lo que debe hacer la aplicación a través de los requerimientos funcionales y no funcionales, se realiza el diagrama de caso de uso del sistema, así como las descripciones de los casos de uso del mismo.

2.2 Modelo Conceptual

El modelo conceptual es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema. Representa conceptos del mundo real, no de los componentes de software. [13]

Se representa en UML con un diagrama de clases en el que se muestra conceptos u objetos del dominio del problema (clases conceptuales), asociaciones entre las clases conceptuales y atributos de estas. No muestra comportamiento y las clases conceptuales no pueden tener métodos.

Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión).

2.3 Modelo del Conceptual Propuesto

Usuario de Reporte: Cualquier persona que necesite obtener o diseñar reportes de una base de datos.

Reportador: Aplicación visual para crear modelos de reporte y a partir de este generarlos.

Servicio: Sistema que permite la comunicación entre la clase Reportador y la Fuente de datos.

Fuente de Datos: Contiene los datos de interés para el Usuario de Reporte.

Visualizador Reporte: Interfaz donde se muestra una vista previa del reporte generado.

Reporte: Información de interés para el usuario.

Archivo: Objeto donde se guardará la información solicitada.

Impresora: Dispositivo en el cual se puede imprimir un reporte creado.

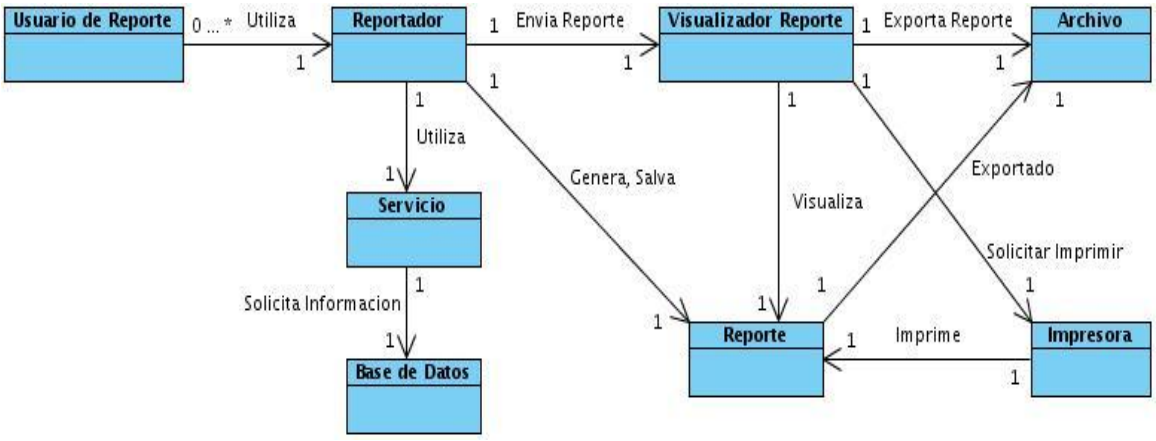


Figura 3: Diagrama del Modelo Conceptual.

2.4 Especificación de los requisitos del software

El objetivo de la captura de requisitos es guiar el desarrollo de software hacia el sistema correcto, definiendo objetivos generales concretos de manera tal que tanto el negocio como los actores se beneficien.

Los requisitos han sido descritos como una característica del sistema a entregar; se pueden clasificar en funcionales y no funcionales.

2.4.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. La aplicación a desarrollar a través de la realización del presente trabajo debe cumplir los siguientes requisitos:

RF1 Elaborar modelo del reporte.
RF2 Adicionar condiciones al modelo del reporte
RF3 Eliminar condiciones al modelo del reporte.
RF4 Guardar el modelo del reporte.
RF5 Cargar un modelo de reporte determinado.
RF6 Generar resultado de un reporte a partir del modelo creado.
RF7 Visualizar el reporte confeccionado.
RF7.1 Exportar el reporte elaborado.
RF7.2 Imprimir el reporte.

2.4.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. La aplicación a desarrollar será un modulo del proyecto productivo alasGRATO, el cual tiene su arquitectura definida, por lo que debe poseer los requerimientos no funcionales definidos en ella: [7]

Requisitos de usabilidad

- ✓ La interfaz de usuario final debe permitir ser utilizada por usuarios con pocos o ningún conocimiento informático.

Requisitos de Software

- ✓ Máquina virtual de Java 1.5 o superior.
- ✓ Sistema Operativo Microsoft Windows 95 o superior.
- ✓ Sistema Operativo Linux con ambiente gráfico KDE o GNOME.

Requisitos de Hardware

- ✓ Se requiere un mínimo de 256 Mb de RAM, 512 Mb recomendado.
- ✓ 1.3 GHz de velocidad de procesamiento.

Requerimientos de apariencia e interfaz externa

- ✓ La interfaz de usuario final debe ser amigable y sencilla para la generación de reportes.

Requerimientos de portabilidad

- ✓ La aplicación debe correr tanto en el sistema operativo Windows como en Linux.

2.5 Modelo de Casos de uso del Sistema

El modelo de casos de uso del sistema contiene actores, casos de uso y sus relaciones.

2.5.1 Actores del Sistema

Un actor es la idealización de una persona externa, un proceso o cosa que interactúa con un sistema, subsistema, o clase. Caracteriza a las interacciones que pueden tener los usuarios con el mismo. Se puede incluir que un usuario puede llegar a desempeñarse como varios actores en tiempo de ejecución, además que diferentes usuarios pueden estar reflejados en un mismo actor, y por lo tanto, representan distintas instancias de la misma definición de actor. Cada actor participa en uno o más casos de uso. [20]
Se muestran a continuación en la Tabla 1 los actores del sistema con su respectiva descripción.

Actor del Sistema	Descripción
Usuario de Reporte	Se beneficia con el diseño y obtención del reporte, puede ser cualquier persona que necesite obtener o diseñar reportes a partir de una base de datos.

Tabla 1: Actor del Sistema.

2.5.2 Diagrama de Casos de Uso del Sistema

A continuación se muestra el Diagrama de Casos de Uso del Sistema en la figura 2, en el mismo está representado el actor que interactúa con el sistema. También se pueden encontrar el Caso de Uso del Sistema (representa los principales procesos del mismo) el cual constituye una funcionalidad dentro de dicho sistema a desarrollar. En el siguiente acápite se muestra la descripción textual de cada caso de uso.

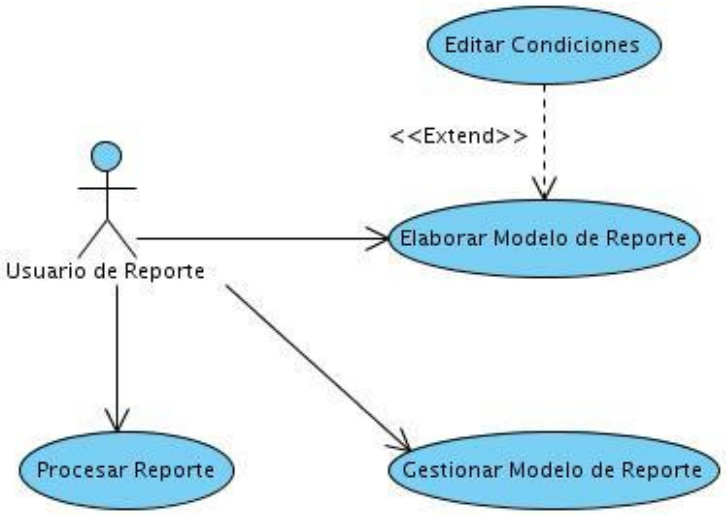


Figura 4: Diagrama de Casos de Uso del Sistema.

2.5.3 Descripción textual de los Casos de Uso del Sistema

A continuación se muestran las descripciones de los casos de uso.

2.5.3.1 Caso de uso Elaborar Modelo de Reporte

Caso de Uso:	Elaborar Modelo de Reporte.

Actores:	Usuario de Reporte.
Resumen:	El caso de uso representa el proceso de creación del modelo de reporte: seleccionar las variables a mostrar o eliminarlas.
Referencia:	RF1
CU Asociados:	
Precondiciones:	Servidor de aplicaciones y de base de datos en funcionamiento.
Prioridad:	Crítico.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El Usuario del Reporte puede realizar varias acciones diferentes: - "Elegir atributo". - "Elegir tabla".	1.1 La aplicación ejecuta una de las siguientes secciones: - "Elegir atributo" ver sección 1. - "Elegir tabla" ver sección 2.
Sección 1: "Elegir atributo"	
2. El Usuario del Reporte elige un atributo para el reporte.	2.1 La aplicación muestra el atributo dentro de una ventana que representa la tabla a la que pertenece con 3 acciones: - "Cerrar ventana de la tabla" ver sección 3. - "Eliminar atributo" ver sección 4. - "Editar Condiciones".
	2.2 La aplicación adiciona los nombres de la tabla y del atributo en la configuración de modelo de reporte
Flujos Alternos	
	2.1 a) La aplicación adiciona el atributo en la ventana que representa la tabla a la que pertenece y en la configuración del modelo de reporte.
	2.1 b) La aplicación muestra un mensaje informando que el atributo ya existe en el área de trabajo.
	2.1 c) La aplicación muestra un mensaje informando que la tabla a la que pertenece el atributo elegido no tiene relación directa o indirecta con las existentes en el área de trabajo.

Sección 2: "Elegir tabla"	
2. El Usuario del Reporte elige una tabla para el reporte.	2.1 La aplicación muestra una ventana, con 3 acciones, que representa la tabla con todos los atributos pertenecientes a ella dentro: Acciones: - "Cerrar ventana de la tabla" ver sección 3. - "Eliminar atributo" ver sección 4. - "Editar Condiciones" ver sección 5.
	2.2 La aplicación adiciona la tabla y sus atributos al modelo de reporte.
Flujos Alternos	
	2.1 a) La aplicación muestra un mensaje informando que la tabla ya existe en el área de trabajo.
	2.1 b) La aplicación muestra un mensaje informando que la tabla no tiene relación directa o indirecta con las existentes en el área de trabajo.
Sección 3: "Cerrar ventana de la tabla"	
3. El Usuario de Reporte elige la acción.	3.1 La aplicación elimina la ventana del área de trabajo con todos los atributos que en ella se encontraban.
	3.1 La aplicación elimina la tabla y sus atributos seleccionados del modelo de reporte.
Sección 4: "Eliminar atributo"	
3. El Usuario de Reporte selecciona atributo a eliminar.	3.1 La aplicación elimina el atributo seleccionado de la ventana.
	3.2 La aplicación elimina el atributo del modelo y las condiciones relacionadas con el, si existen.
Flujo Alterno	
	3.2 a) Si se eliminan todos los atributos de una tabla la aplicación elimina su ventana del área de trabajo y elimina la tabla del modelo.
Pos Condiciones:	Se ha creado una estructura o modelo de reporte.

Interfaces:	Anexo 1

Tabla 2: Descripción textual del Caso de uso: "Elaborar Modelo de Reporte".

2.5.3.2 Caso de uso Gestionar Condiciones

Caso de Uso:	Editar Condiciones.
Actores:	Usuario de Reporte.
Resumen:	El caso de uso representa todo el proceso de gestión de condiciones o sea: establecer condiciones sobre las variables o campos seleccionados en la creación del modelo de reporte.
Referencia:	RF2, RF3
CU Asociados:	
Precondiciones:	Caso de uso "Elaborar Modelo de Reporte".
Prioridad:	Secundario.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
1. El Usuario de Reporte elige la "Editar Condiciones".	1.1 La aplicación muestra una ventana con la lista de los atributos de la tabla, un área para creación de condiciones desactivada, una lista de las condiciones creadas relacionadas con la tabla sobre la que se trabaja, en caso de que existan, y 3 opciones: - "Adicionar". - "Aceptar". (desactivada) - "Cancelar".
2. El Usuario de Reporte elige atributo.	2.1 La aplicación activa las opciones para la creación de la condición en el área destinada para esa función.
3. El Usuario de Reporte presiona el botón "Adicionar".	3.1 La aplicación valida la estructura de la condición.
	3.2 La aplicación adiciona la condición a la lista de condiciones creadas con la opción: - "Eliminar".

	3.3 La aplicación activa la opción “Aceptar”.
4. El Usuario de Reporte presiona el botón “Aceptar”.	4.1 La aplicación adiciona la o las condiciones existentes en la lista al modelo de reporte.
	4.2 La aplicación devuelve la ventana a sus valores por defecto y la cierra.
Flujo Alterno	
4. a) El Usuario de Reporte elige la opción “Eliminar” para eliminar una condición.	4.1 La aplicación elimina la condición elegida de la lista de condiciones creadas.
5. El Usuario de Reporte presiona el botón “Aceptar”.	5.1 La aplicación actualiza la lista de condiciones en el modelo de reporte.
	5.2 La aplicación devuelve la ventana a sus valores por defecto y la cierra.
4. b) El Usuario de Reporte presiona el botón “Cancelar”.	4.1 La aplicación cierra la ventana de edición de condiciones sin guardar ningún cambio que se haya efectuado.
Pos Condiciones:	Se han creado o eliminado condiciones en el modelo de reporte.
Interfaces:	Anexo 1

Tabla 3: Descripción textual del Caso de uso: “Gestionar Condiciones”.

2.5.3.3 Caso de uso Gestionar Modelo de Reporte

Caso de Uso:	Gestionar Modelo de Reporte.
Actores:	Usuario de Reporte.
Resumen:	El caso de uso representa todo el proceso de gestión de modelos de reporte creados o sea: eliminar, guardar o cargar un modelo de reporte.
Referencia:	RF4, RF5
CU Asociados:	
Precondiciones:	Caso de uso “Elaborar Modelo de Reporte”.
Prioridad:	Secundario.
Flujo Normal de Eventos	

Acción del Actor	Respuesta del sistema
1. El Usuario del Reporte puede realizar varias acciones diferentes: - "Nuevo". - "Abrir". - "Guardar".	1.1 La aplicación ejecuta una de las siguientes secciones: - "Nuevo" ver sección 1. - "Abrir" ver sección 2. - "Guardar" ver sección 3.
Sección 1: "Nuevo"	
2. El Usuario de Reporte elige la opción.	2.1 La aplicación muestra una ventana emergente preguntando si desea guardar el modelo de reporte creado hasta ese momento y con dos opciones: - "Si". - "No".
3. El Usuario de Reporte presiona el botón "Si".	3.1 La aplicación muestra una ventana emergente que da la opción de elegir nombre y destino del modelo de reporte a guardar y 2 opciones: - "Aceptar". - "Cancelar".
4. El Usuario de Reporte presiona el botón "Aceptar".	4.1 La aplicación guarda el modelo de reporte creado con el nombre insertado en el destino elegido y devuelve todo a su estado inicial en la ventana principal.
Flujos Alternos	
3. El Usuario de Reporte presiona el botón "No".	3.1 La aplicación cierra la cierra la ventana emergente y devuelve todo a su estado inicial en la ventana principal.
4. El Usuario de Reporte elige la opción "Cancelar".	4.1 La aplicación cierra la ventana emergente y devuelve todo a su estado inicial en la ventana principal.
Sección 2: "Abrir"	
2. El Usuario de Reporte elige la opción.	2.1 La aplicación muestra una ventana emergente que da la opción de elegir el dirección del modelo de reporte a cargar y 2 opciones:

	- “Aceptar”. - “Cancelar”.
3. El Usuario de Reporte busca y elige el modelo de reporte a cargar y presiona el botón “Aceptar”.	3.1 La aplicación estructura la ventana principal según los datos del modelo cargado (tablas, atributos y condiciones).
Flujos Alternos	
3. El Usuario de Reporte presiona el botón “Cancelar”.	3.1 La aplicación cierra la cierra la ventana emergente
Sección 3 “Guardar”	
2. El Usuario de Reporte elige la opción.	2.1 La aplicación muestra una ventana emergente que da la opción de elegir nombre y destino del modelo de reporte a guardar y 2 opciones: - “Aceptar”. - “Cancelar”.
3. El Usuario de Reporte presiona el botón “Aceptar”.	3.1 La aplicación guarda el modelo de reporte creado con el nombre insertado en el destino elegido y devuelve todo a su estado inicial en la ventana principal.
Flujo Alterno	
3. El Usuario de Reporte elige la opción “Cancelar”.	3.1 La aplicación cierra la ventana emergente.
Pos Condiciones:	Se ha eliminado, guardado o cargado una estructura o modelo de un reporte.
Interfaces:	Anexo 1

Tabla 4: Descripción textual del Caso de uso: “Gestionar Modelo de Reporte”.

2.5.3.4 Caso de uso Procesar Reporte

Caso de Uso:	Procesar Reporte.
Actores:	Usuario de Reporte.
Resumen:	El caso de uso representa todo el proceso de creación de los reportes o sea:

	generar el reporte después elaborado y cargado un modelo de reporte, visualizarlo, exportarlo o imprimirlo.
Referencia:	RF6, RF7, RF7.1, RF7.2
CU Asociados:	
Precondiciones:	Caso de uso “Elaborar Modelo de Reporte” o caso de uso “Gestionar Modelo de Reporte”.
Prioridad:	Crítico.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del sistema**
Sección 9: “Generar”	
1. El Usuario de Reporte elige la opción “Generar”.	1.1 La aplicación toma el modelo de reporte creado (tablas, atributos y condiciones), crea una consulta, genera el reporte mostrando en una tabla en el área inferior de la ventana principal los resultados obtenidos y activa la acción: - “Exportar” ver sección 2.
Flujo Alterno	
	1.1 a) La aplicación muestra un mensaje informando que el modelo esta vacío.
Sección 2: “Exportar”	
2. El Usuario de Reporte elige la opción.	2.1 La aplicación muestra en una ventana una vista previa del reporte con varias acciones , las dos principales son: - “Guardar”. - “Imprimir” ver sección 3.
3. El Usuario de Reporte elige la opción “Guardar".	3.1 La aplicación muestra una ventana emergente para elegir destino, nombre y extensión para exportar el reporte y dos botones: - “Guardar”. - “Cancelar”.
4. El Usuario de Reporte elige destino, nombre y extensión para el reporte y	4.1 La aplicación cierra la ventana emergente y salva el reporte en el destino elegido exportado con el

presiona el botón “Guardar”.	nombre y la extensión elegidos.
Flujo Alterno	
4. a) El Usuario de Reporte presiona el botón “Cancelar”.	4.1 La aplicación cierra la ventana emergente.
Sección 3: “Imprimir”	
3. El Usuario de Reporte elige la opción.	3.1 La aplicación muestra una ventana emergente para elegir los datos de impresión y dos botones: - “Aceptar”. - “Cancelar”.
4. El Usuario de Reporte elige los datos de impresión y presiona el botón “Aceptar”.	4.1 La aplicación cierra la ventana emergente y envía el reporte a la impresora.
Flujo Alterno	
4. a) El Usuario de Reporte presiona el botón “Cancelar”.	4.1 La aplicación cierra la ventana emergente.
Pos Condiciones:	Se ha generado y visualizado, exportado o imprimido un reporte.
Interfaces:	Anexo 1

Tabla 5: Descripción textual del caso de uso: “Procesar Reporte”.

2.6 Conclusiones

En el presente capitulo se definió el modelo conceptual para identificar los conceptos más significativos del dominio del problema. Se analizaron y aprobaron los requisitos funcionales necesarios para obtener un sistema eficiente, se definieron las responsabilidades de cada persona mediante un diagrama de casos de uso del sistema dando cumplimiento a las necesidades de los requisitos funcionales propuestos y se realizaron además las descripciones de los casos de uso involucrados en el desarrollo de la aplicación.

CAPÍTULO 3: DISEÑO DEL SISTEMA

3.1 Introducción

En el presente capítulo se desarrolla el flujo de trabajo del diseño, a través del cual se modela la aplicación que debe ser capaz de cumplir los requisitos antes analizados. Para ello se desarrollarán diferentes artefactos como los diagramas de clases del diseño y los diagramas de interacción, definiéndose además la distribución del sistema mediante el modelo de despliegue.

3.2 Objetivos del diseño

El diseño tiene como principales objetivos comprender detalladamente los requisitos funcionales y no funcionales, sistemas operativos, tecnologías de distribución, restricciones relacionadas con el lenguaje de programación, tecnologías de interfaz de usuario. Además, crea un punto de partida para las actividades de implementación que siguen.

3.3 Diagrama de Clases del Diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases y las relaciones entre ellas. Son utilizados durante el proceso de análisis y diseño, donde se describe la realización de los casos de uso y sirve de abstracción al modelo de implementación y al código fuente correspondiente. [20]

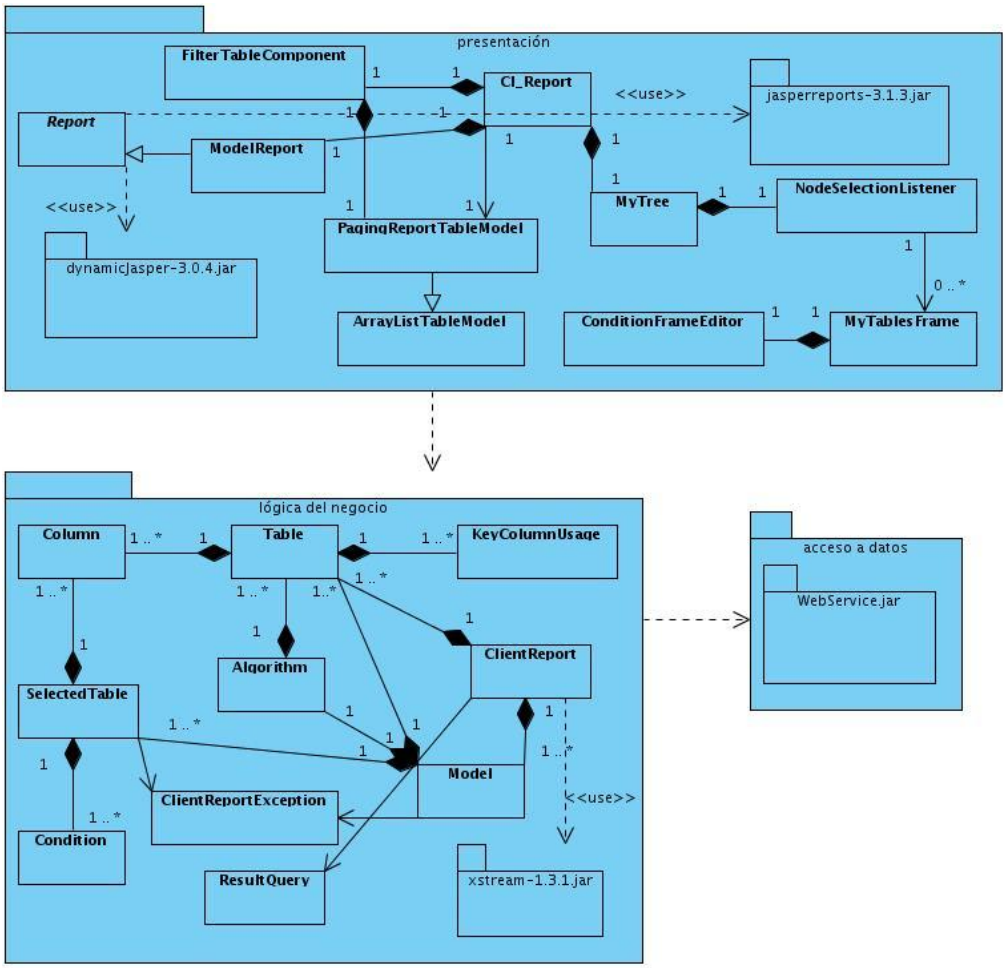


Figura 5: Diagrama de clases del diseño.

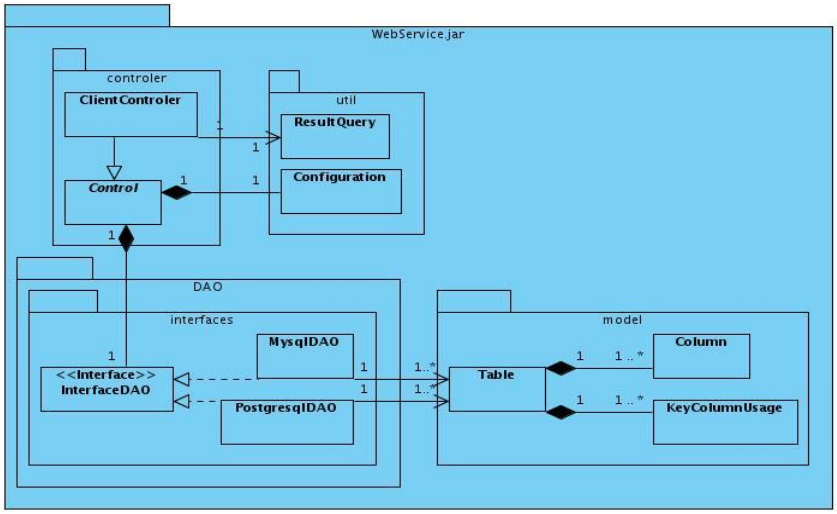


Figura 6: Diagrama de clases del diseño del paquete WebService.jar.

3.4 Patrones utilizados en la resolución del problema

3.4.1 Patrón Modelo de tres capas

Se evidencia el uso de este patrón de arquitectura en el diagrama de clases del diseño (ver figura 5) donde estas se encuentran ubicadas según su función en tres paquetes, presentación, lógica del negocio y acceso a datos. Cada uno de estos paquetes representa una de las capas propuestas por este patrón: capa de presentación, lógica del negocio y acceso a datos y base de datos respectivamente.

En la capa “presentación” se encuentran las clases relacionadas con la interfaz visual, en la que se tratan los datos y se presentan al usuario, comunicándose solamente con la capa de negocio.

En la capa “lógica del negocio” se encuentran las clases encargadas de la intercomunicación entre las capas de presentación y la de datos. Esta capa reúne todos los aspectos del software que se deben automatizar o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican.

3.4.2 Patrón DAO

También en el diagrama de clases del caso de uso Gestionar Reportes (ver figura 6) se evidencia el uso del patrón DAO que no es más que un objeto o componente de software, en este caso el servicio web, que encapsula todos los accesos a la fuente de datos suministrando una interfaz común entre ella y la aplicación y además, maneja la conexión con la fuente de datos para obtener y almacenar información.

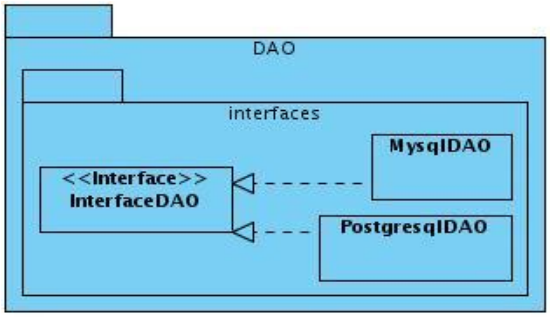


Figura 7: Ejemplo de la aplicación del patrón DAO en el paquete WebService.jar.

3.4.3 Patrones GRASP

De los patrones GRASP usamos los siguientes:

Experto

Este patrón se ve evidenciado en la clase Model, puesto que al poseer una lista de objetos Table que modelan las tablas de la base de datos escogida, una lista de objetos SelectedTable que contiene la información de los campos que han sido seleccionados por el usuario para generar la consulta y un objeto Algorithm para buscar camino entre las tablas seleccionadas, posee la información necesaria para crear la consulta a ejecutar sobre la base de datos. Así a esta clase se le asigna la responsabilidad de crear las consultas a ejecutar.

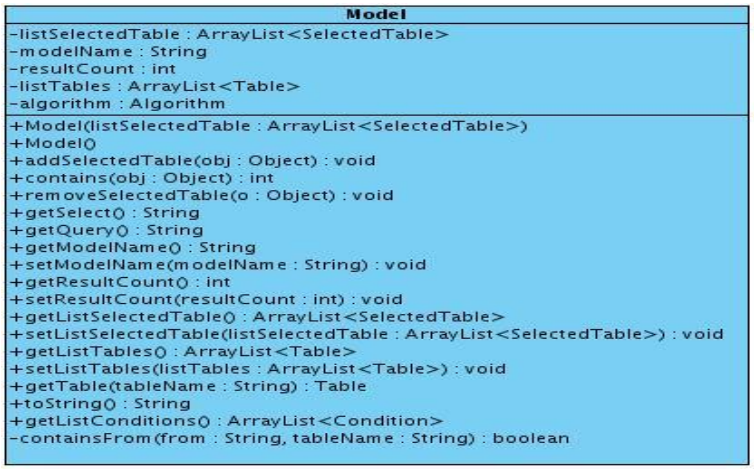


Figura 8: Ejemplo de la aplicación del patrón Experto.

Creador

La clase ClientReport contiene objetos Model y Table; por ello, este patrón sugiere que ClientReport es idónea para asumir la responsabilidad de crear las instancias de las clases Model y Table.

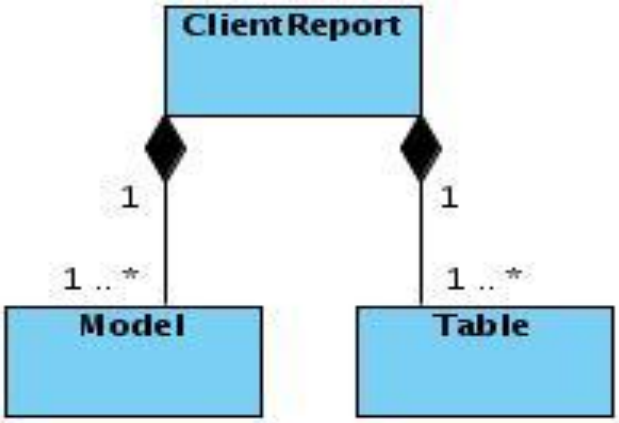


Figura 9: Ejemplo de la aplicación del patrón Creador.

Bajo Acoplamiento

Puesto que la entidad ClientReport tiene la responsabilidad de conocer los modelos creados durante el proceso de creación de reportes, ésta incluye la clase Model. La clase Model terminará acoplándose al conocimiento de objetos de la clase SelectedTable para almacenar información de los campos seleccionados para la generación del reporte, así para que el experto ClientReport pueda conocer la información de los atributos seleccionados, este no necesariamente debe acoplarse a la clase SelectedTable, pues su acoplamiento con Model hace que disponga de información para ello.

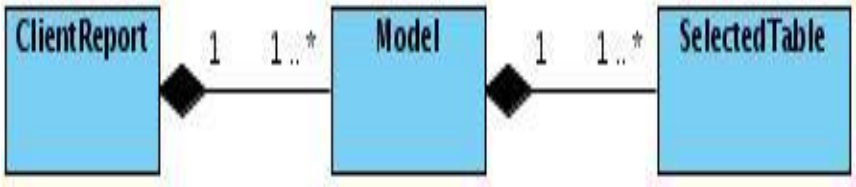


Figura 10: Ejemplo de la aplicación del patrón Bajo Acoplamiento.

3.5 Diagramas de secuencia

Los diagramas de secuencia forman parte de la vista del comportamiento del sistema, brindan una especificación de este representando un conjunto de clases que intercambian mensajes en una iteración organizada a lo largo del tiempo.

3.5.1 Caso de uso Elaborar Modelo de Reporte

En el presente diagrama se muestran el flujo de acciones que ocurren en la aplicación al ejecutar elegir un campo o atributo creando un modelo de reporte dentro del caso de uso Elaborar Modelo de Reporte.

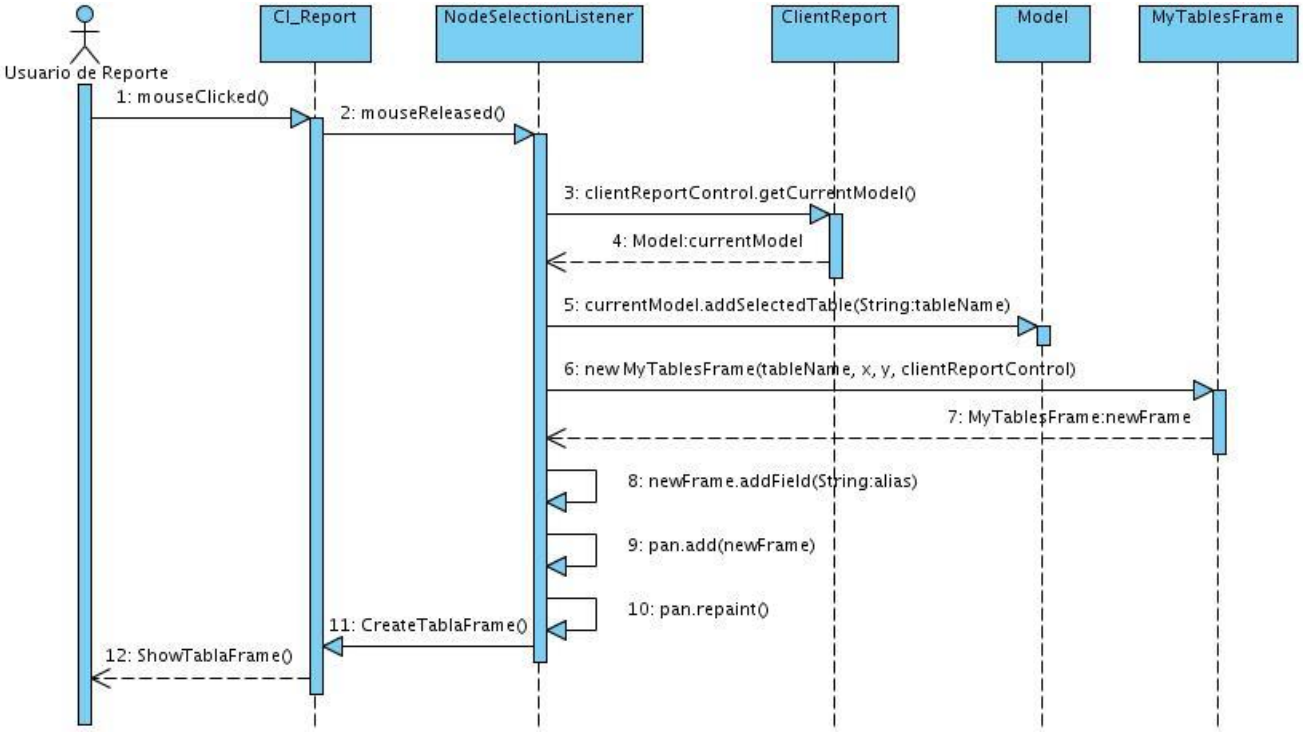


Figura 11: Diagrama de secuencia CU Elaborar Modelo de Reporte sección 1 “Elegir Atributo”.

3.5.2 Caso de uso Gestionar Modelo de Reporte

En el presente diagrama se muestran el flujo de acciones que ocurren en la aplicación al ejecutar la acción “Guardar” para salvar un modelo de reporte dentro del caso de uso Gestionar Modelo de Reporte.

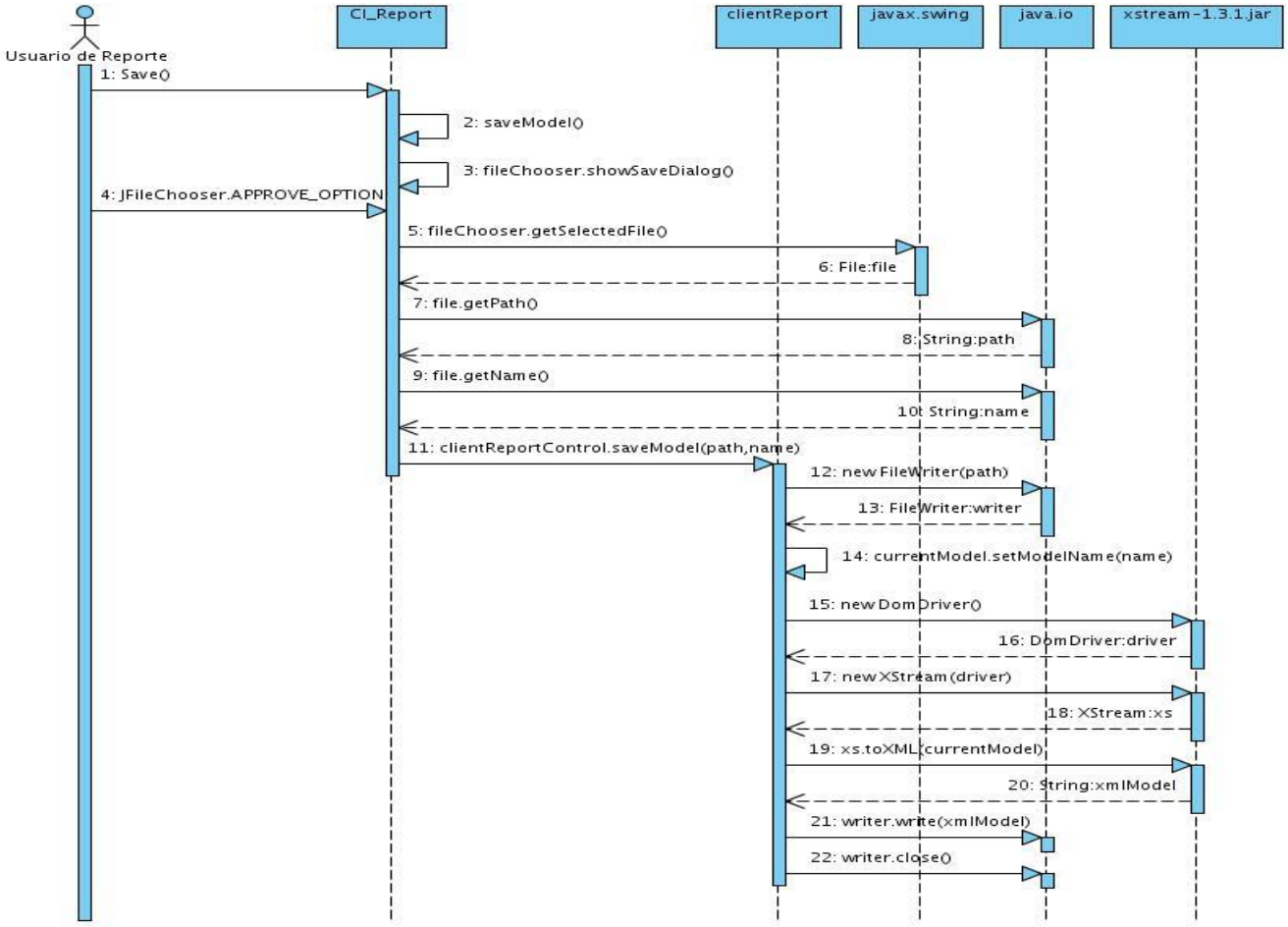


Figura 12: Diagrama de secuencia Caso de uso Gestionar Modelo de Reporte sección 3 “Guardar”.

3.5.3 Caso de uso Procesar Reporte

En el diagrama presentado a continuación se reflejan el flujo de acciones que ocurren en la aplicación al ejecutarse la acción “Generar” para generar el reporte dentro del caso de uso Procesar Reporte.

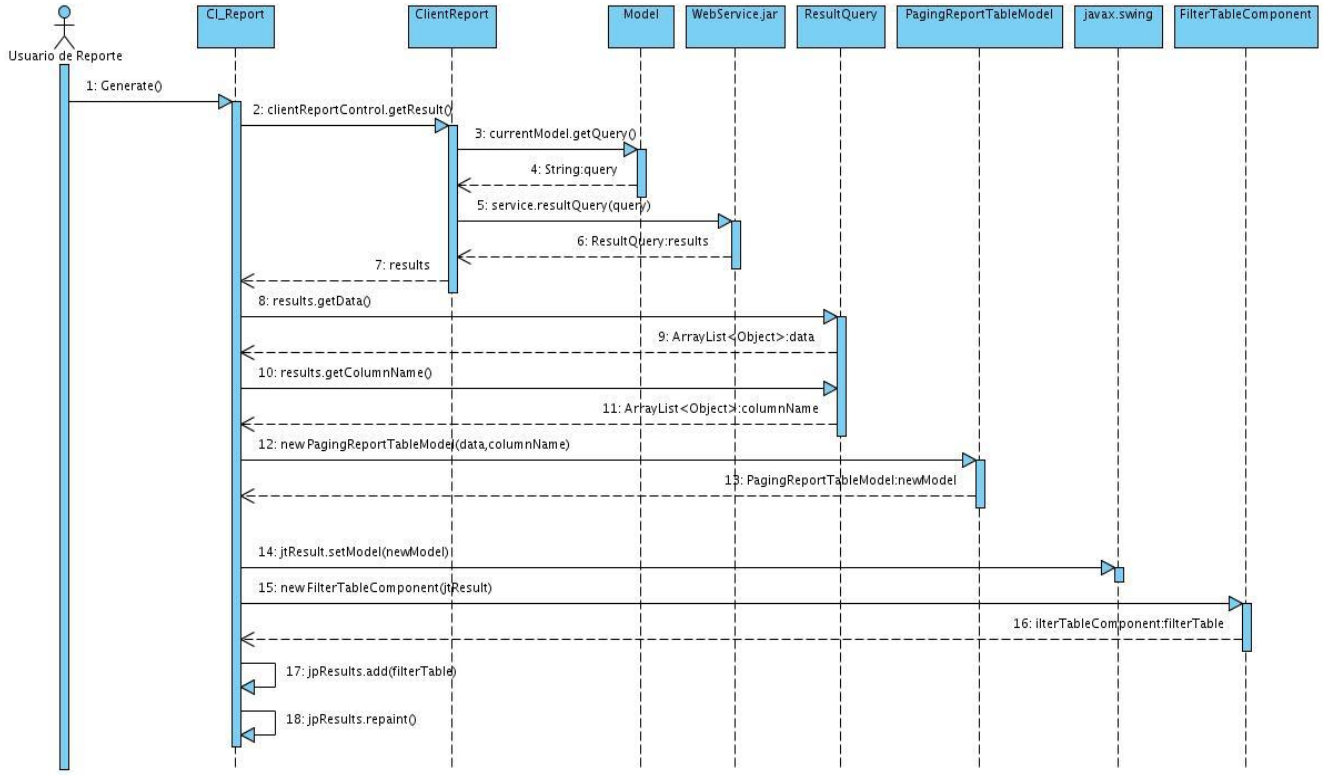


Figura 13: Diagrama de secuencia Caso de uso Gestionar reportes sección 9 “Generar”.

Los demás diagramas de secuencia se pueden ver en el Anexo 2.

3.5 Descripción de clases del diseño

En este epígrafe se describen las clases del diseño relacionadas con la lógica del negocio y con el acceso a datos.

Nombre de la Clase:	ClientReport
Descripción:	Contiene toda la información de la estructura de la base de datos, así como, los datos del modelo de reporte a generar. Controla la lógica del negocio.

Atributos	**Tipo**
- listTables	ArrayList<Table>
- listModel	ArrayList<Model>
- currentModel	Model

Operaciones	Descripción
+ ClientReport()	Constructor por defecto de la clase.
+ ClientReport(listTables:ArrayList<Table>)	Constructor de la clase pasándole la lista de tablas de la base de datos.
+ ClientReport (listTables:ArrayList<Table>, listModel:ArrayList<Model>, currentModel:Model)	Constructor de la clase pasándole la lista de tablas de la base de datos, una lista de modelos de reportes y el modelo de reporte actual.
+ getListTables():ArrayList<Table>	Retorna la lista de tablas de la base de datos.
+ setListTables(listTables: ArrayList<Table>): void	Permite cambiar la lista de tablas de la base de datos.
+ getColumn(nameField: String): Column	Retorna una columna dado un nombre o alias
+ getTableName(alias: String): String	Retorna dado un alias de un campo la tabla que lo contiene.
+ getTable(tableName: String): Table	Retorna una tabla a partir un nombre.
+ getCurrentModel():Model	Retorna el modelo de reporte actual.
+ setCurrentModel(currentModel: Model): void	Permite cambiar el modelo de reporte actual.
+ getListModel():ArrayList<Model>	Retorna la lista de modelos de reporte.
+ setListModel(listModel: ArrayList<Model>): void	Permite cambiar la lista de modelos de reporte.
+ loadModel(path:String): void	Permite cargar un modelo desde una locación elegida.
+ saveModel(path:String,nameFile:String): void	Permite salvar un modelo en una locación elegida.
+ initializeModel():void	Inicializa el modelo de reporte actual.
+ getresult():ResultQuery	Retorna los resultados de la consulta realizada a la base de datos.

Tabla 6: Descripción de la clase ClientReport del CU Gestionar Reportes.

Nombre de la Clase:	Model
Descripción:	Contiene todos los datos necesarios para generar el reporte.

Atributos	Tipo
- listSelectedTable	ArrayList<SelectedTable>
- listTables	ArrayList<Table>
- algorithm	Algorithm
- modelName	String
- modelPath	String
- resultCount	int
Operaciones	**Descripción**
+ Model()	Constructor por defecto de la clase.
+ Model(listSelectedTable: ArrayList<SelectedTable>)	Constructor de la clase pasándole la lista de tablas seleccionadas.
+ getModelName():String	Retorna el nombre del modelo de reporte.
+ setModelName(modelName:String):void	Permite cambiar el nombre del modelo.
+ getResultCount():int	Retorna la cantidad de datos que fueron obtenidos por el modelo una vez generado.
+ setResultCount(resultCount:int): void	Permite cambiar la cantidad de datos.
+ getListSelectedTable(): ArrayList<SelectedTable>	Retorna la lista de tablas seleccionadas.
+ setListSelectedTable (listSelectedTable:ArrayList<SelectedTable>): void	Permite cambiar la lista de tablas seleccionadas.
+ addSelectedTable(obj: SelectedTable):void	Permite adicionar una tabla a la lista de tablas seleccionadas.
+ contains(obj:SelectedTable):int	Retorna la posición de la tabla si existe en caso contrario retorna -1.
+ removeSelectedTable (o:SelectedTable):void	Permite eliminar una tabla de la lista de tablas seleccionadas.
+ getQuery():String	Crea y retorna la consulta que se ejecutara sobre la base de datos.
+ getListTables():ArrayList<Table>	Retorna la lista de tablas de la base de datos.
+ setListTables (listTables:ArrayList<Table>):void	Permite cambiar la lista de tablas de la base de datos.

+ getTable(tableName:String):Table	Retorna una tabla según un nombre.
+ containsFrom (from:String,tableName:String):boolean	
+ getModelPath():String	Retorna la dirección donde está guardado el modelo.
+ setModelPath(modelPath:String):void	Permite cambiar la dirección donde está guardado el modelo.

Tabla 7: Descripción de la clase Model del CU Gestionar Reportes.

Nombre de la Clase:	Table
Descripción:	Modela una tabla de la base de datos.
Atributos	**Tipo**
- tableName	String
- tableEngine	String
- listColumn	ArrayList<Column>
- listKeyColumnUsage	ArrayList<KeyColumnUsage>
Operaciones	**Descripción**
+ Table()	Constructor por defecto de la clase.
+ Table (tableName:String,tableEngine:String, listColumns: ArrayList<Column>)	Constructor de la clase pasándole el nombre, el tipo de tabla y la lista de columnas.
+ getTableName():String	Retorna el nombre de la tabla.
+ setTableName(tableName:String):void	Permite cambiar el nombre de la tabla.
+ getTableEngine():String	Retorna el tipo de tabla.
+ setTableEngine(tableEngine:String): void	Permite cambiar el tipo de tabla.
+ getListColumns():ArrayList<Column>	Retorna la lista de columnas de la tabla.
+ setListColumns (listColumns:ArrayList<Column>):void	Permite cambiar la lista de columnas de la tabla.
+ addColumn(o:Column):boolean	Permita adicionar una columna a la tabla.
+ clear():void	Permite eliminar todas las columnas de la tabla.
+ getColumn(index:int):Column	Retorna una columna según una posición.
+ containsColumn(name:String):Column	Retorna una columna según un nombre o alias.

+ getListKeyColumnUsage(): ArrayList<KeyColumnUsage>	Retorna la lista de campos que son foráneos a dicha tabla.
+ setListKeyColumnUsage (listKeyColumnUsage: ArrayList<KeyColumnUsage>):void	Permite cambiar la lista de campos que son foráneos a dicha tabla.
+ getKeyColumnUsage(pos:int): KeyColumnUsage	Retorna un campo foráneo dado una posición.
+ addKeyColumnUsage(o:KeyColumnUsage) :boolean	Permite adicionar un nuevo campo foráneo.
+ getkeyColumnUsage (referencedtableName:String): KeyColumnUsage	Retorna un campo foráneo dado un nombre de tabla.

Tabla 8: Descripción de la clase Table del CU Gestionar Reportes.

Nombre de la Clase:	SelectedTable
Descripción:	Contiene la información de los campos que han sido seleccionados por el usuario para generar el reporte.
Atributos	**Tipo**
- tableName	String
- listColumnSelected	ArrayList<Column>
- listConditions	ArrayList<Condition>
- position	Point
Operaciones	**Descripción**
+ SelectedTable()	Constructor por defecto de la clase.
+ SelectedTable (tableName:String, listColumnSelected: ArrayList<Column>, listConditions:ArrayList<Condition>)	Constructor de la clase pasándole el nombre de la tabla, una lista de campos seleccionados y una lista de condiciones.
+ getListColumnSelected(): ArrayList<Column>	Retorna la lista de campos seleccionados.
+ setListColumnSelected (listColumnSelected:ArrayList<Column>):void	Permite cambiar la lista de campos seleccionados.

+ getTableName():String	Retorna el nombre de la tabla.
+ setTableName(tableName:String):void	Permite cambiar el nombre de la tabla.
+ getListConditions(): ArrayList<Condition>	Retorna la lista de condiciones.
+ setListConditions (listConditions: ArrayList<Condition>): void	Permite cambiar la lista de condiciones.
+ removeField(nameField: String):void	Elimina, según un nombre, el campo de la lista de campos seleccionados y las condiciones en las que esté presente de la lista de condiciones.
+ addField(column:Column):void	Permite adicionar un nuevo campo a la lista de campos seleccionados.
+ contains(nameField: String):int	Retorna según un nombre la posición de un campo si existe en caso contrario retorna -1.
+ containsCondition(o:Condition):int	Retorna según una condición su posición si existe en caso contrario retorna -1.
+ addCondition(o:Condition):void	Permite adicionar una nueva condición.
+ removeCondition(o: Condition):void	Permite eliminar una condición de la lista de condiciones.
+ getAllConditions():String	Retorna en una cadena todas las condiciones creadas.
+ getPosition():Point	
+ setPosition(position:Point):void	

Tabla 9: Descripción de la clase SelectedTable del CU Gestionar Reportes.

Nombre de la Clase:	ClientControler
Descripción:	Clase que contiene e implementa las funcionalidades que brinda el servicio.
Operaciones	**Descripción**
+ ClientControler()	Constructor por defecto de la clase.
+ getXMLSchema():String	Retorna la información de la base de datos (Tablas, Campos, etc)
+ getSqlQuery(String sql, int begen, int end): String	Retorna los datos generados por la consulta ejecutada sobre la base de datos.

Tabla 10: Descripción de la clase ClientControler del paquete WebService.jar del CU Gestionar Reportes.

Nombre de la Clase:	Configuration
Descripción:	Clase que contiene toda la información referente a la configuración de la base de datos a conectarse.
Atributos	**Tipo**
- dbHost	String
- dbPort	String
- dbType	Int
- dbName	String
- dbUser	String
- dbPassword	String
- dbDirSchema	String
Operaciones	**Descripción**
+ Configuration()	Constructor por defecto de la clase.
+ Configuration(dbHost:String, dbPort:String, dbType:int, dbName:String, dbUser:String, dbPassword:String, dbDirSchema:String)	Constructor de la clase pasándole por parámetros los datos de configuración de la base de datos a conectarse.
+ getDbHost():String	Retorna la dirección de la base de datos.
+ setDbHost(dbHost:String):void	Permite cambiar la dirección de la base de datos.
+ getDbName():String	Retorna el nombre de la base de datos.
+ setDbName(dbName:String):void	Permite cambiar el nombre de la base de datos.
+ getDbPassword():String	Retorna la contraseña con la cual se va a conectar.
+ setDbPassword(dbPassword:String):void	Permite cambiar la contraseña con la cual se va a conectar.
+ getDbPort():String	Retorna el puerto de conexión.
+ setDbPort(dbPort:String):void	Permite cambiar el puerto de conexión.
+ getDbType():int	Retorna el tipo de base de datos (1 para Mysql y 2 para Postgresql).
+ setDbType(dbType:int):void	Permite cambiar el tipo de base de datos.
+ getDbUser():String	Retorna el usuario con el cual se va a conectar.

+ setDbUser(dbUser:String):void	Permite cambiar el usuario con el cual se va a conectar.
+ getDbDirSchema():String	Retorna la dirección del esquema de la base de datos.
+ setDbDirSchema(dbDirSchema:String):void	Permite cambiar la dirección del esquema de la base de datos.

Tabla 11: Descripción de la clase Configuration del paquete WebService.jar del CU Gestionar Reportes.

Nombre de la Clase:	MysqlDAO
Descripción:	Clase que brinda funcionalidades para el trabajo sobre bases de datos implementadas en MySql.
Atributos	**Tipo**
- connection	Connection
- config	Configuration
Operaciones	**Descripción**
+ MysqlDAO(xmlConfig: Configuration)	Constructor de la clase pasándole por parámetros la configuración de la pase de datos.
+ connect():void	Crea la conexión a la base de datos.
- getAllTablesName():ArrayList<Table>	Retorna todas las tablas contenidas en la base de datos.
+ getInformationFromInformationSchema(): ArrayList<Table>	Retorna toda la información del esquema de la base de datos.
+ close():void	Cierra la conexión a la base de datos.
- getReferencedColumnNamesFromInformationSchema (tableName:String, engine:String): ArrayList<KeyColumnUsage>	Retorna la lista de campos foráneos.
+ resultQuery(sql:String, begen:int, end:int):ResultQuery	Retorna los resultados de la consulta realizada sobre la base de datos.

Tabla 12: Descripción de la clase MysqlDAO del paquete WebService.jar del CU Gestionar Reportes.

Las demás descripciones de clases se pueden ver en el Anexo 3.

3.6 Diagrama de Despliegue

El modelo de despliegue muestra la configuración de nodos de procesamiento en tiempo de ejecución, los enlaces de comunicación entre ellos, y el componente de los casos y los objetos que residen en ellos. [20]

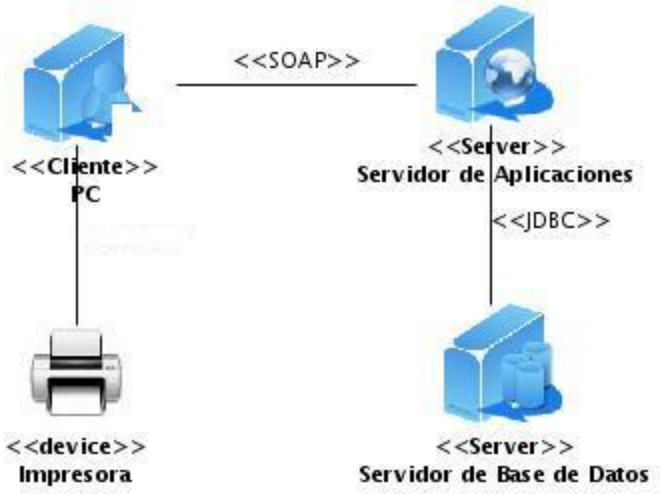


Figura 14: Diagrama de Despliegue de la aplicación.

3.6 Conclusiones

En el presente capítulo se realizó el diagrama de clases del diseño para los casos de uso, así como los diagramas de interacción. Además se mostró la posible distribución de los nodos del sistema mediante el diagrama de despliegue.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

Partiendo del resultado del diseño, se modela el sistema en términos de componentes, y se muestran además algunas pantallas de las nuevas funcionalidades de la aplicación con sus descripciones. En el presente capítulo se describe cómo se ha implementado la aplicación a desarrollar.

4.2 Diagrama de componentes

Los diagramas de componentes muestran la estructura de los componentes, incluyendo los clasificadores que especifican los componentes y los artefactos que los implementan. También se pueden utilizar para mostrar la estructura de alto nivel del modelo de implementación en términos de subsistemas de implementación, y las relaciones entre los elementos de implementación. [20]

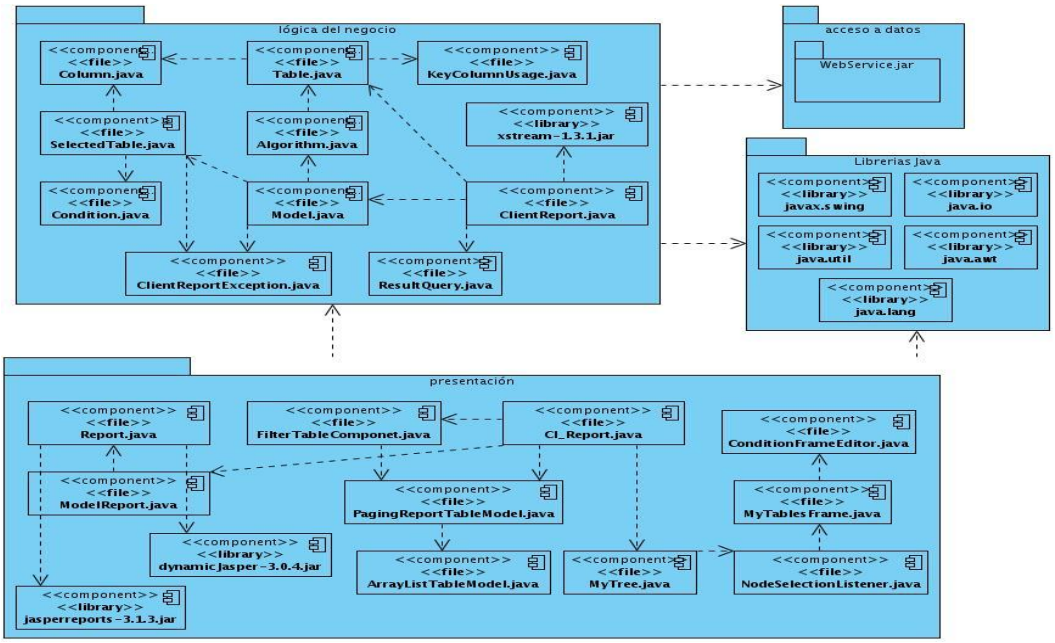


Figura 15: Diagrama de componentes Gestionar Reportes.

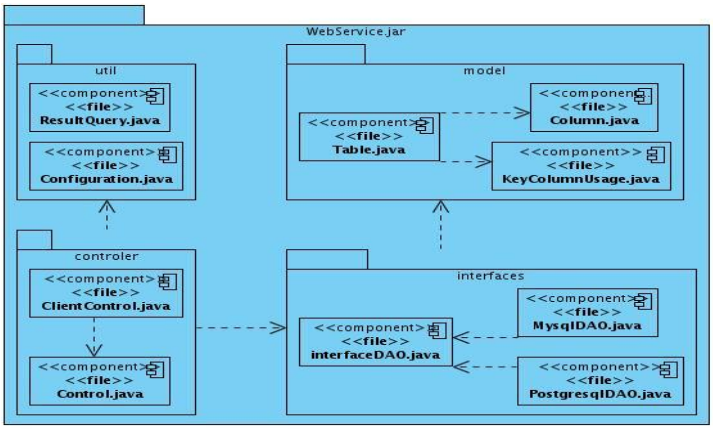


Figura 16: Diagrama de componentes del paquete de componentes WebService.jar.

4.3 Implementaciones relevantes

Dentro de los requerimientos funcionales de la aplicación está el generar el reporte según los atributos seleccionados y las condiciones creadas sobre ellos. Para ello se necesita conocer las conexiones entre las tablas seleccionadas, es decir, un camino entre ellas que será utilizado para crear la consulta que posteriormente se ejecutará sobre la base de datos.

Para crear este camino se implementó el método que se muestra en la figura 18 basado en algoritmos de inteligencia artificial de búsqueda a lo ancho como se explica a continuación. [21]

Una búsqueda primero a lo ancho (breadth-first) genera y explora primero todos los sucesores del nodo raíz. Si no se encuentra la meta, pasa a los sucesores del segundo nivel y así sucesivamente por niveles. Suponiendo que el objetivo a alcanzar es el nodo 7, el recorrido primero a lo ancho del espacio de búsqueda que se muestra en la figura 17, es 1-2-3-4-5-6-7.

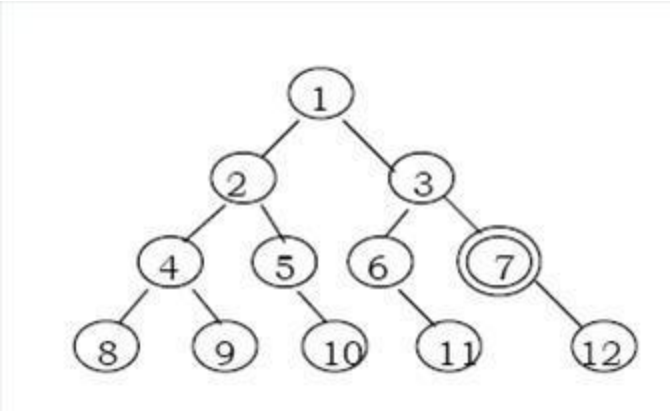


Figura 17: Un ejemplo de espacio de búsqueda.

Si el número máximo de hijos (o ramas) de un nodo, en este caso tablas de la base de datos, es *b* y la profundidad de la solución es *d*, entonces el número de nodos en el nivel *d* es *bd* y la cantidad de tiempo usada en la búsqueda es en el caso peor: 1 + b + b2 + ... + bd.

Para grandes valores de *d*, puede ser aproximada por *bd*. Es por esto que la complejidad temporal de este método de búsqueda es *O(bd)*, lo cual es una función exponencial de *d*.

Este método tiene como ventaja que siempre encuentra el camino más corto a la solución, si ésta existe, aún en el caso de que el espacio de búsqueda sea infinito, por eso la búsqueda primero a lo ancho siempre encuentra una solución óptima para esa medida.

El método es efectivo cuando el factor de ramificación, o sea, el número promedio de hijos de un nodo, es pequeño, pues entonces la cantidad de nodos por niveles será pequeña y es mejor explorar un nivel antes de pasar al siguiente. Sin embargo, tiene las siguientes desventajas:

- Necesita mucha memoria. Como cada nivel del árbol tiene que ser almacenado completamente para poder generar el próximo nivel y la cantidad de memoria es proporcional al número de nodos almacenados, su complejidad espacial es también *O(bd)*.
- Requiere mucho trabajo, especialmente si el camino más corto a la solución es muy largo, puesto que el número de nodos que necesita examinar se incrementa exponencialmente con la longitud del camino.
- Los operadores irrelevantes o redundantes incrementarán grandemente el número de nodos que deben explorarse.

Además, este método puede llevar a una búsqueda exhaustiva. Ella es particularmente inapropiada en situaciones donde hay muchos caminos que conducen a soluciones, pero cada uno de ellos es muy largo. Haciendo cada nodo una tabla que contiene atributos se puede aplicar este algoritmo para lograr establecer relaciones de unión entre éstas. Esto permitiría que el usuario seleccione los campos que desee visualizar en el reporte y la aplicación se encargaría entonces de establecer las relaciones de unión para poder realizar una consulta determinada.

```
public boolean Algorism(Table tabStart, Table tabFinal,
        ArrayList<Table> general) {
    boolean b = false;
    b = Search(tabStart, tabFinal, general);
    if (b)
        way.add(0, tabStart);
    return b;
}
public boolean Search(Table tabStart, Table tabFinal,
        ArrayList<Table> general) {
    boolean a = false;
    if (!tested.contains(tabStart)) {
        if (tabStart.equals(tabFinal))
            return true;
        else {
            ArrayList<Table> aux = new ArrayList<Table>();
            for (int i = 0; i < tabStart.getListKeyColumnUsage().size(); i++) {
                for (int j = 0; j < general.size(); j++) {
                    if (tabStart.getListKeyColumnUsage().get(i).getReferencedTableName().equals(
                                    general.get(j).getTableName()))
                        aux.add(general.get(j));
                }
            }
            ArrayList<Table> aux1 = new ArrayList<Table>();
            for (int i = 0; i < general.size(); i++) {
                for (int j = 0; j < general.get(i).getListKeyColumnUsage().size(); j++) {
                    if (tabStart.getTableName().equals(general.get(i).getListKeyColumnUsage().get(j)
                                    .getReferencedTableName()))
                        aux1.add(general.get(i));
                }
            }
            for (int j = 0; j < aux1.size(); j++) {
                if (!aux.contains(aux1.get(j)))
                    aux.add(aux1.get(j));
            }
            if (aux.contains(tabFinal)) {
                a = true;
                way.add(tabFinal);
                return a;
            } else {
                tested.add(tabStart);
                for (int i = 0; i < aux.size(); i++) {
                    a = Search(aux.get(i), tabFinal, general);
                    if (a) {
                        way.add(0, aux.get(i));
                        break;
                    }
                }
            }
        }
    }
    return a;
}
```

Figura 18: Algoritmo de búsqueda en profundidad utilizado para buscar caminos entre tablas.

4.4 Pantallas principales

En la siguiente imagen se muestra la ventana principal donde el especialista trabajara para crear sus reportes.

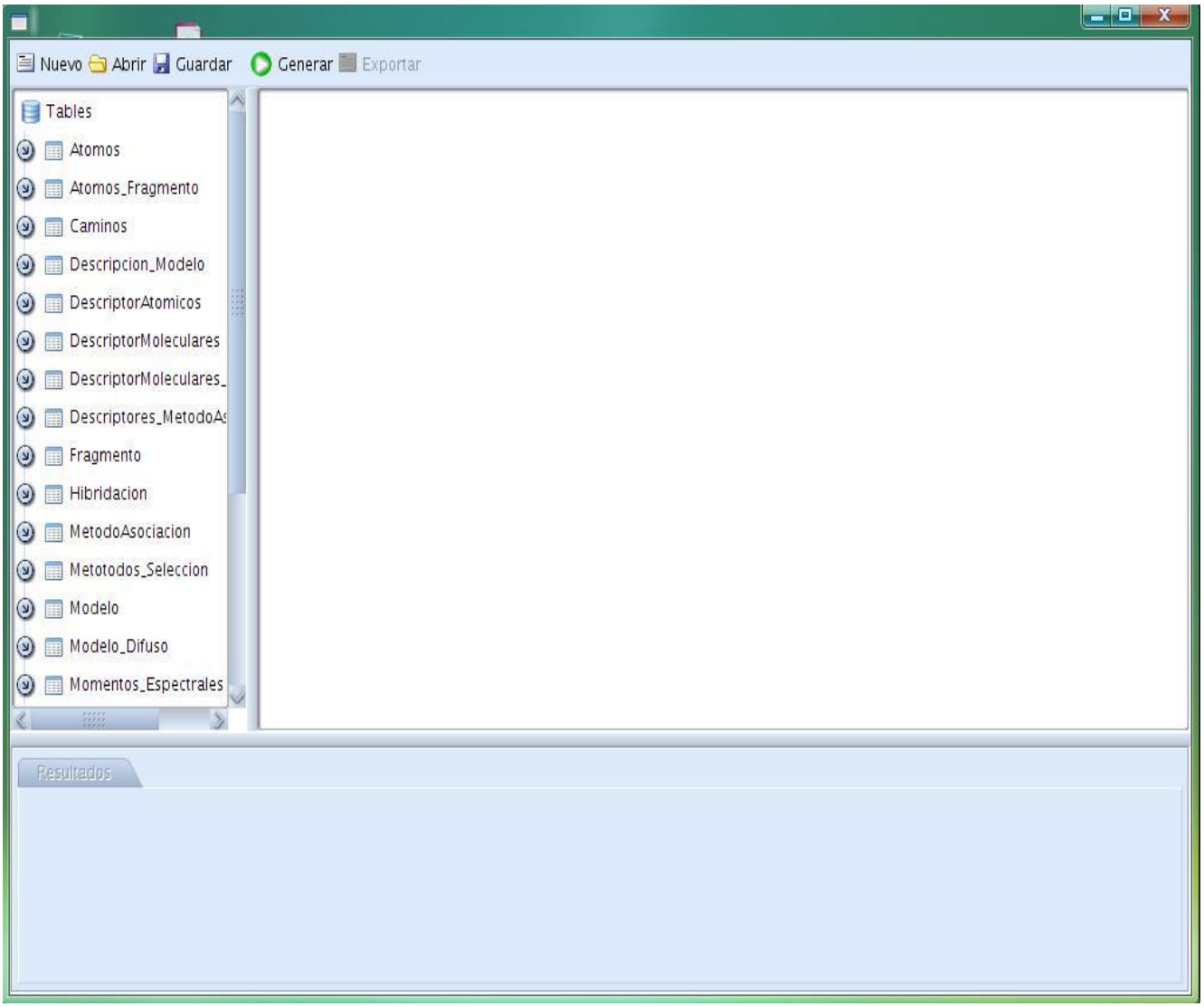
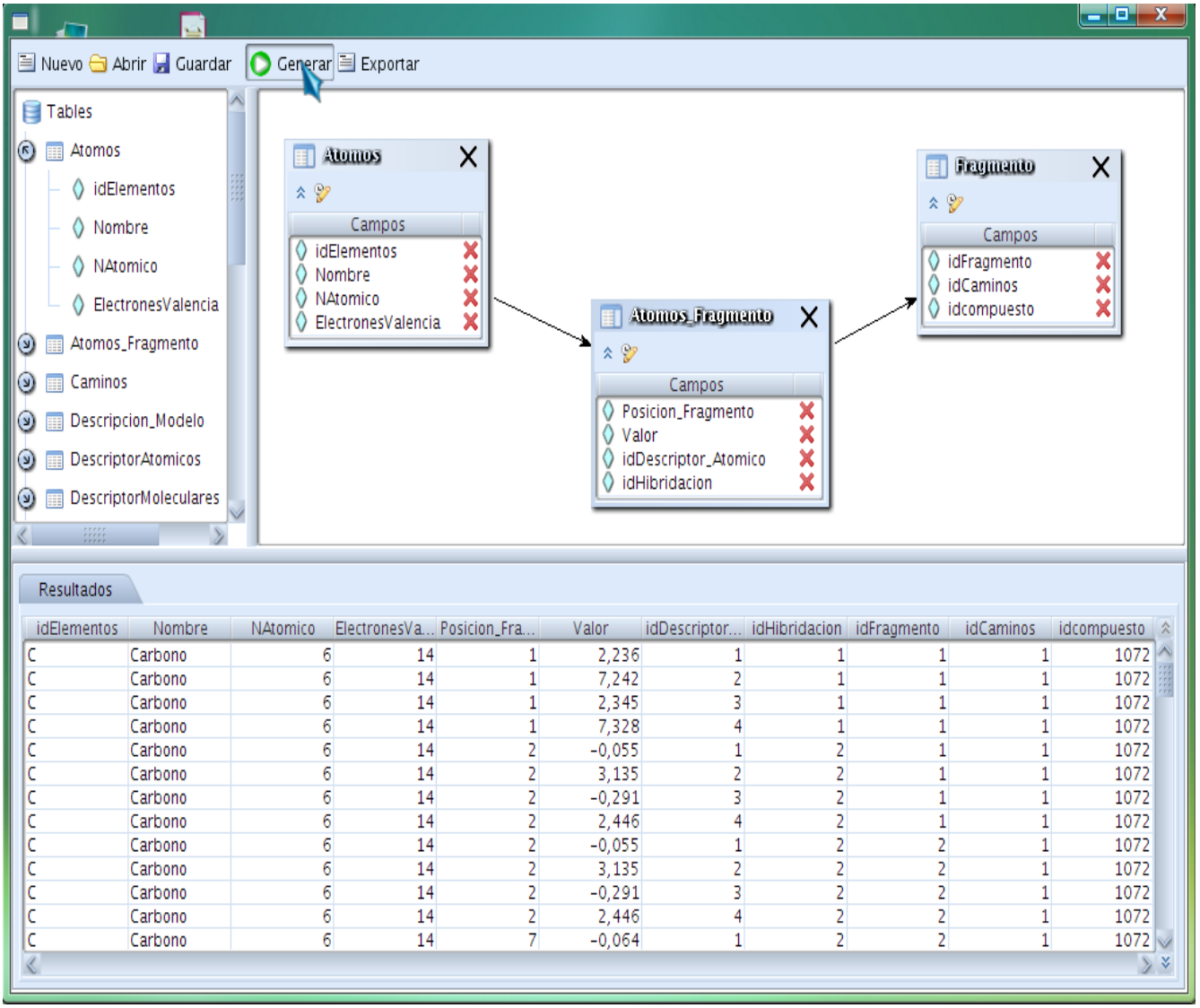


Figura 19: Pantalla inicial de la ventana principal de la interfaz para cliente.

En la siguiente imagen el especialista selecciona atributos para su reporte y lo genera mostrándose los resultados en una tabla.



idElementos	Nombre	NAtomico	ElectronesVa...	Posicion_Fra...	Valor	idDescriptor...	idHibridacion	idFragmento	idCaminos	idcompuesto
C	Carbono	6	14	1	2,236	1	1	1	1	1072
C	Carbono	6	14	1	7,242	2	1	1	1	1072
C	Carbono	6	14	1	2,345	3	1	1	1	1072
C	Carbono	6	14	1	7,328	4	1	1	1	1072
C	Carbono	6	14	2	-0,055	1	2	1	1	1072
C	Carbono	6	14	2	3,135	2	2	1	1	1072
C	Carbono	6	14	2	-0,291	3	2	1	1	1072
C	Carbono	6	14	2	2,446	4	2	1	1	1072
C	Carbono	6	14	2	-0,055	1	2	2	1	1072
C	Carbono	6	14	2	3,135	2	2	2	1	1072
C	Carbono	6	14	2	-0,291	3	2	2	1	1072
C	Carbono	6	14	2	2,446	4	2	2	1	1072
C	Carbono	6	14	7	-0,064	1	2	2	1	1072

Figura 20: Pantalla con atributos seleccionados y resultados del reporte.

En la siguiente imagen se muestra la ventana para crear condiciones sobre los atributos seleccionados de una tabla específica.

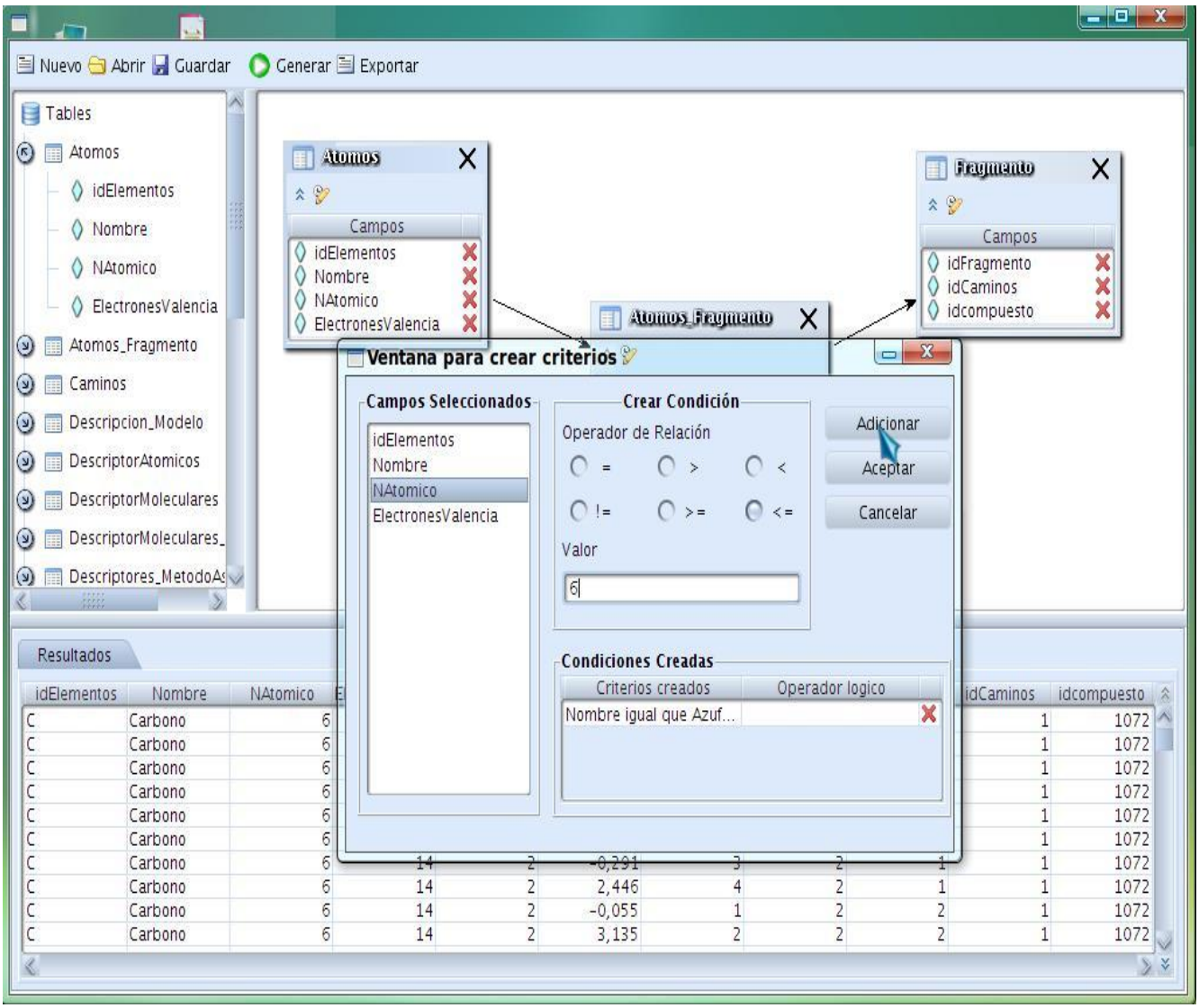
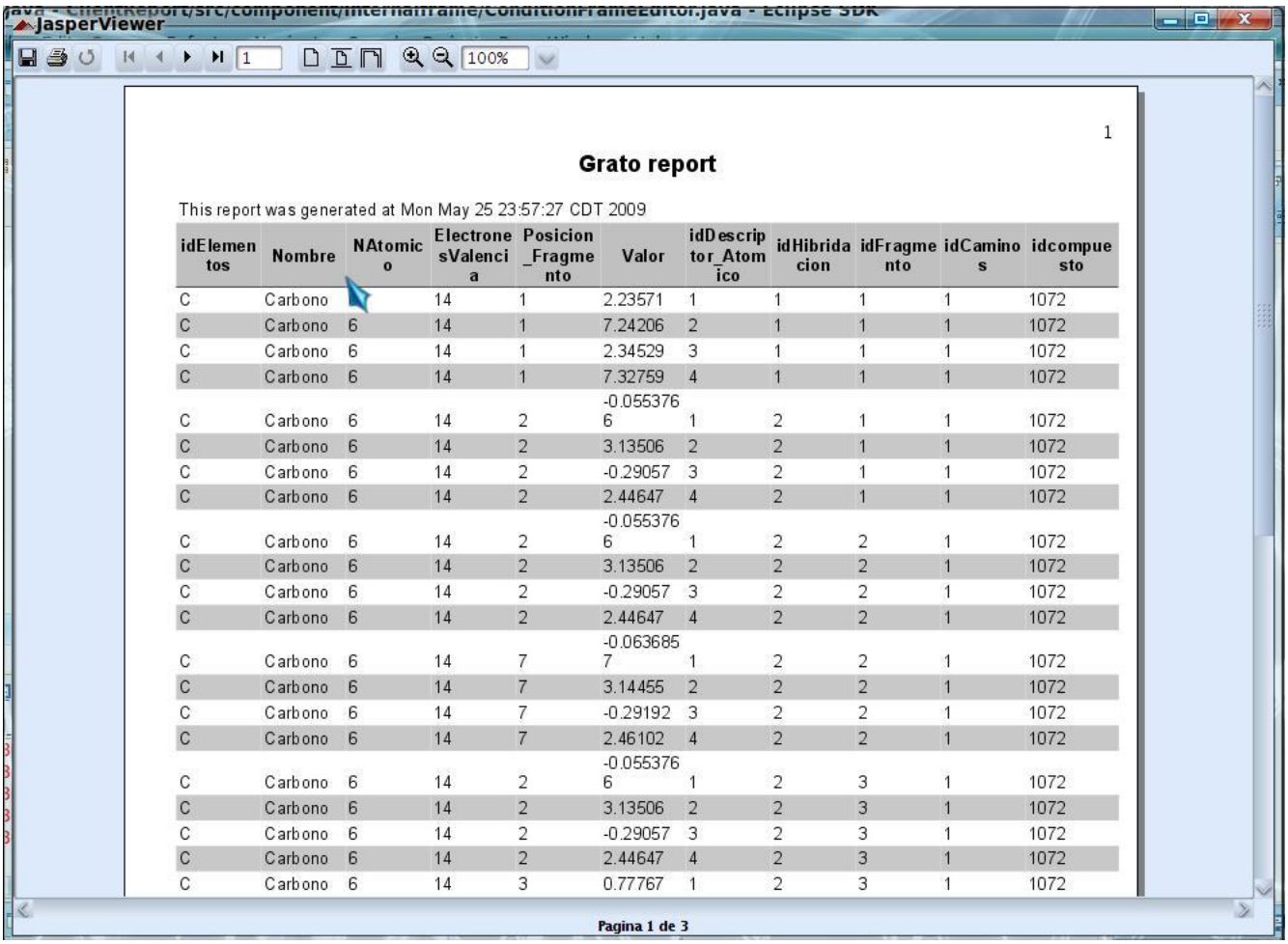


Figura 21: Pantalla de ventana para crear condiciones.

En la siguiente imagen se muestra una vista previa del reporte generado.



idElementos	Nombre	NAtomico	ElectronesValencia	Posicion_Fragmento	Valor	idDescriptor_Atomico	idHibridacion	idFragmento	idCaminos	idcompuesto
C	Carbono	6	14	1	2.23571	1	1	1	1	1072
C	Carbono	6	14	1	7.24206	2	1	1	1	1072
C	Carbono	6	14	1	2.34529	3	1	1	1	1072
C	Carbono	6	14	1	7.32759	4	1	1	1	1072
C	Carbono	6	14	2	-0.055376	1	2	1	1	1072
C	Carbono	6	14	2	3.13506	2	2	1	1	1072
C	Carbono	6	14	2	-0.29057	3	2	1	1	1072
C	Carbono	6	14	2	2.44647	4	2	1	1	1072
C	Carbono	6	14	2	-0.055376	1	2	2	1	1072
C	Carbono	6	14	2	3.13506	2	2	2	1	1072
C	Carbono	6	14	2	-0.29057	3	2	2	1	1072
C	Carbono	6	14	2	2.44647	4	2	2	1	1072
C	Carbono	6	14	7	-0.063685	1	2	2	1	1072
C	Carbono	6	14	7	3.14455	2	2	2	1	1072
C	Carbono	6	14	7	-0.29192	3	2	2	1	1072
C	Carbono	6	14	7	2.46102	4	2	2	1	1072
C	Carbono	6	14	2	-0.055376	1	2	3	1	1072
C	Carbono	6	14	2	3.13506	2	2	3	1	1072
C	Carbono	6	14	2	-0.29057	3	2	3	1	1072
C	Carbono	6	14	2	2.44647	4	2	3	1	1072
C	Carbono	6	14	3	0.77767	1	2	3	1	1072

Figura 22: Pantalla de la vista previa del reporte.

4.5 Pruebas exploratorias

En este acápite se describen pruebas realizadas sobre la interfaz de la aplicación desarrollada. Estas se llevaron a cabo mediante casos de prueba que tenían como objetivo demostrar que sus funcionalidades son operativas, que los datos de entrada se aceptan de forma adecuada y que se produce una salida correcta garantizando la integridad de la información que se almacena y procesa. Se examinan fundamentalmente algunos aspectos del modelo del sistema sin profundizar mucho en la estructura interna del software.

Caso de Prueba 1: Generar reporte.

Flujo central del caso de uso Procesar Reporte.

Clases Válidas	Clases inválidas	Resultado Esperado	Resultado de la Prueba
<Se seleccionan los siguientes datos: Tabla: - Átomos Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia y se elige la opción Generar.>		<El sistema busca los datos de los campos seleccionados y los muestra (86 filas).>	<Satisfactorio>
<Se seleccionan los siguientes datos: Tabla: - Átomos Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia Condiciones: - NAtomico <= 47 - ElectronesValencia > 10 y se elige la opción Generar.>		<El sistema verifica que los datos entrados sean correctos, busca los datos de los campos seleccionados que cumplan con las condiciones creadas y los muestra (22 filas).>	<Satisfactorio>
<Se seleccionan los siguientes datos: Tabla: - Átomos	<Se seleccionan los siguientes datos: Condiciones: - NAtomico <= Azufre	<El sistema verifica que los datos entrados sean correctos y muestra	<Satisfactorio>

Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia Condiciones: - ElectronesValencia > 10 y se elige la opción Generar.>	.>	en mensaje informando que el dato entrado para comparar debe ser numérico.>	
<Se seleccionan los siguientes datos: Tablas: - Átomos - Átomos_Fragmento - Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia - Posición_Fragmento, - Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto y se elige la opción Generar.>		<El sistema busca los datos de los campos seleccionados y los muestra (244068 filas). >	<Satisfactorio>
<Se seleccionan los siguientes datos: Tablas: - Átomos - Átomos_Fragmento		<El sistema verifica que los datos entrados sean correctos, busca los datos de los campos	<Satisfactorio>

- Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia - Posición_Fragmento, - Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto Condiciones: - Atomos.Nombre=Azufre - Atomos_Fragmento.Valor> 10 - Fragmento.idFragmento < 2950 y se elige la opción Generar.>		seleccionados que cumplan con las condiciones creadas y los muestra (85 filas).>	
<Se seleccionan los siguientes datos: Tablas: - Átomos - Átomos_Fragmento - Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia - Posición_Fragmento,	<Se seleccionan los siguientes datos: Condiciones: - Átomos_Fragmento.Valor > Nitrógeno .>	<El sistema verifica que los datos entrados sean correctos y muestra en mensaje informando que el dato entrado para comparar debe ser numérico.>	<Satisfactorio>

- Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto Condiciones: - Atomos.Nombre=Azufre - Fragmento.idFragmento < 2950 y se elige la opción Generar.>			

Tabla 13: Caso de Prueba 1: Generar reporte.

Caso de Prueba 2: Guardar modelo de reporte.

Flujo central del caso de uso Gestionar Modelo de Reporte sección 3: "Guardar".

Clases Válidas	Clases inválidas	Resultado Esperado	Resultado de la Prueba
<Se seleccionan los siguientes datos: Tabla: - Átomos Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia y se elige la opción Guardar.>		<El sistema guarda en un archivo (.modelreport) la estructura del modelo de reporte con los datos seleccionados.>	<Satisfactorio>
<Se seleccionan los siguientes datos: Tabla:		<El sistema guarda en un archivo (.modelreport) la	<Satisfactorio>

- Átomos Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia Condiciones: - NAtomico <= 47 - ElectronesValencia > 10 y se elige la opción Guardar.>		estructura del modelo de reporte con los datos seleccionados.>	
<Se seleccionan los siguientes datos: Tablas: - Átomos - Átomos_Fragmento - Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia - Posición_Fragmento, - Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto y se elige la opción Guardar.>		<El sistema guarda en un archivo (.modelreport) la estructura del modelo de reporte con los datos seleccionados.>	<Satisfactorio>

<Se seleccionan los siguientes datos: Tablas: - Átomos - Átomos_Fragmento - Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia - Posición_Fragmento, - Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto Condiciones: - Atomos.Nombre=Azufre - Atomos_Fragmento.Valor>10 - Fragmento.idFragmento<2950 y se elige la opción Guardar.>		<El sistema guarda en un archivo (.modelreport) la estructura del modelo de reporte con los datos seleccionados.>	<Satisfactorio>

Tabla 14: Caso de Prueba 2: Guardar modelo de reporte.

Caso de Prueba 3: Cargar modelo de reporte.

Flujo central del caso de uso Gestionar Modelo de Reporte sección 2: “Abrir”.

Clases Válidas	Clases inválidas	Resultado Esperado	Resultado de la Prueba
<Se elige la opción Abrir y se		<El sistema carga la	<Satisfactorio>

selecciona un archivo (*.modelreport) que contiene la estructura de un modelo de reporte con los siguientes datos: Tabla: - Átomos Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia y se elige la opción Guardar.>		estructura del modelo de reporte con los datos contenidos en el archivo y los muestra.>	
<Se elige la opción Guardar y se selecciona un archivo (*.modelreport) que contiene la estructura de un modelo de reporte con los siguientes datos: Tabla: - Átomos Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia Condiciones: - NAtomico <= 47 - ElectronesValencia > 10 .>		<El sistema carga la estructura del modelo de reporte con los datos contenidos en el archivo y los muestra.>	<Satisfactorio>
<Se elige la opción Guardar y se selecciona un archivo (*.modelreport) que contiene la estructura de un modelo de reporte		<El sistema carga la estructura del modelo de reporte con los datos contenidos en el	<Satisfactorio>

con los siguientes datos: Tablas: - Átomos - Átomos_Fragmento - Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia - Posición_Fragmento, - Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto .>		archivo y los muestra.>	
<Se elige la opción Guardar y se selecciona un archivo (*.modelreport) que contiene la estructura de un modelo de reporte con los siguientes datos: Tablas: - Átomos - Átomos_Fragmento - Fragmento Campos: - idElementos, - Nombre, - NAtomico, - ElectronesValencia		<El sistema carga la estructura del modelo de reporte con los datos contenidos en el archivo y los muestra.>	<Satisfactorio>

- Posición_Fragmento, - Valor, - idDescriptor_Atomico, - idHibridacion - idFragmento, - idCaminos, - idcompuesto Condiciones: - Atomos.Nombre=Azufre - Atomos_Fragmento.Valor>10 - Fragmento.idFragmento<2950 .>			

Tabla 15: Caso de Prueba 3: Cargar modelo de reporte.

Caso de Prueba 4: Exportar reporte.

Flujo central del caso de uso Procesar Reporte sección 2: “Exportar”.

Clases Válidas	Clases inválidas	Resultado Esperado	Resultado de la Prueba
<Se selecciona exportar un reporte generado a formato PDF (*.pdf).>		<El sistema exporta los datos del reporte en un archivo PDF.>	<Satisfactorio>
<Se selecciona exportar un reporte generado a formato ODT (*.odt).>		<El sistema exporta los datos del reporte en un archivo ODT.>	<Satisfactorio>
<Se selecciona exportar un reporte generado a formato HTML (*.htm,*.html).>		<El sistema exporta los datos del reporte en un archivo HTML.>	<Satisfactorio>
<Se selecciona exportar un reporte generado a formato CSV (*.csv).>		<El sistema exporta los datos del reporte en un archivo CSV.>	<Satisfactorio>

<Se selecciona exportar un reporte generado a formato XML (*.jrpxml,*.xml).>		<El sistema exporta los datos del reporte en un archivo XML.>	<Satisfactorio>

Tabla 16: Caso de Prueba 4: Exportar reporte.

4.6 Conclusiones

En el capítulo se analizó la forma en que han de estar distribuidos los diferentes componentes de la aplicación a través del modelo de componentes y algunas implementaciones relevantes. Además, se muestran algunas pantallas de las funcionalidades de la aplicación con una breve descripción de las mismas, así como algunas pruebas realizadas para comprobar su funcionamiento.

CONCLUSIONES

- Se diseñó e implementó un módulo de la plataforma alasGRATO para la generación de reportes.

- Se confeccionó un plug-in para su integración a la plataforma.

- Se realizaron pruebas exploratorias al módulo implementado reportando resultados satisfactorios.

RECOMENDACIONES

- Implementar la opción de exportar a los formatos RTF y XLS.
- Optimizar el algoritmo que busca un camino de relación entre tablas.

REFERENCIAS BIBLIOGRÁFICAS

1. GrapeCity-Data Dynamics. *GrapeCity-Data Dynamics.* [En línea] 2009. [Citado el: 20 de Febrero de 2009.] http://www.datadynamics.com.

2. Agata Report. *Agata Report.* [En línea] Solis, 2009. [Citado el: 20 de Febrero de 2009.] http://www.agata.org.br.

3. SAP. *SAP.* [En línea] 2009. [Citado el: 20 de Febrero de 2009.] http://www.sap.com/solutions/sapbusinessobjects/sme/reporting/crystalreports/index.epx.

4. IdeaReports. *IdeaReports.* [En línea] InterSoft, 2008. [Citado el: 21 de Febrero de 2009.] http://www.intersoft.com.ar/ideareports.html.

5. JasperForge.org. *JasperForge.org.* [En línea] Jaspersoft Corporation, 2009. [Citado el: 21 de Febrero de 2009.] http://jasperforge.org/website/jasperreportswebsite/trunk/highlights.html?group_id=252.

6. DynamicJasper. *DynamicJasper.* [En línea] FDV Solutions , 2009. [Citado el: 22 de Febrero de 2009.] http://dynamicjasper.sourceforge.net/index.html.

7. López, Yadira Marrero y García, Adonis R. Rosales. Propuesta del diseño arquitectónico de la Plataforma bioGRATO. Ciudad de la Habana : s.n., 2008.

8. Jordana, Garcilaso**.** *Introducción a OpenUp.* [ppt] 2008.

9. OpenUp. *OpenUp.* [En línea] 2009. [Citado el: 18 de Marzo de 2009.] http://epf.eclipse.org/wikis/openup/.

10. Martínez, Llilian Durand y González, Osmany Becerra**.** *Análisis y Diseño de un Sistema para la*

Generación de Reportes. [pdf] Ciudad de la Habana : s.n., 2008.

11. Microsoft Developer Network. *Microsoft Developer Network.* [En línea] 2009. [Citado el: 5 de Abril de 2009.] http://msdn.microsoft.com/es-es/library/bb972251.aspx#M15

12. forge.morfeo-project.org. *forge.morfeo-project.org.* [En línea] 2009. [Citado el: 5 de Abril de 2009.] http://forge.morfeo-project.org/wiki/index.php/D.3.2_Documento_de_definici%C3%B3n_de_modelos_avanzados_de_comunicaci%C3%B3n_y_composici%C3%B3n_de_recursos#Definici.C3.B3n_de_Capas

13. Larman, Craig**.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall, 2004.

14. Sun Developer Network. *Sun Developer Network.* [En línea] Sun Microsystems, Inc, 2009. [Citado el: 25 de Abril de 2009.] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

15. Périssé, Marcelo Claudio. *Una Metodología Simplificada.* Buenos Aires : s.n., 2001. ISBN: 987-43-2947-5.

16. Visual Paradigm. *Visual Paradigm.* [En línea] 2009. [Citado el: 5 de Abril de 2009.] http://www.visual-paradigm.com/product/vpuml/modeleredition.jsp.

17. definición.org. *definición.org.* [En línea] 2009. [Citado el: 15 de Abril de 2009.] http://www.definicion.org/lenguaje-de-programacion.

18. Sun Microsystems. 2009. Sun Microsystems. *Sun Microsystems.* [En línea] Sun Microsystems, 2009. [Citado el: 15 de abril de 2009.] www.sun.com.

19. Retamar, Angel. AsturLinux. *AsturLinux.* [En línea] 2004. [Citado el: 18 de Abril de 2009.]

http://asturlinux.org/archivos/jornadas2004/ponencias/eclipse_ide.pdf.

20. Rational Software Corporation. 2003. *Rational Unified Process.* s.l. : Rational Software Corporation, 2003. Version 2003.06.00.65.

21. Bello, Dr. Rafael. 1998. *Métodos de Solución de Problemas para la Inteligencia Artificial.* s.l. : UCLV, 1998.

BIBLIOGRAFÍA

1. OpenUp. *OpenUp.* [En línea] 2009. [Citado el: 18 de Marzo de 2009.] http://epf.eclipse.org/wikis/openup/.

2. Larman, Craig**.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall, 2004.

3. Sun Developer Network. *Sun Developer Network.* [En línea] Sun Microsystems, Inc, 2009. [Citado el: 25 de Abril de 2009.] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

4. Visual Paradigm. *Visual Paradigm.* [En línea] 2009. [Citado el: 5 de Abril de 2009.] http://www.visual-paradigm.com/product/vpuml/modeleredition.jsp.

5. Eckstein, Robert. 1998. *JAVA Swing.* Sebastopol : O'Reilly & Associates, 1998.

6. Rational Software Corporation. 2003. *Rational Unified Process*. s.l. : Rational Software Corporation, 2003. Version 2003.06.00.65.

7. Bello, Dr. Rafael. 1998. *Métodos de Solución de Problemas para la Inteligencia Artificial.* s.l. : UCLV, 1998.

ANEXOS

Anexo 1: Interfaces

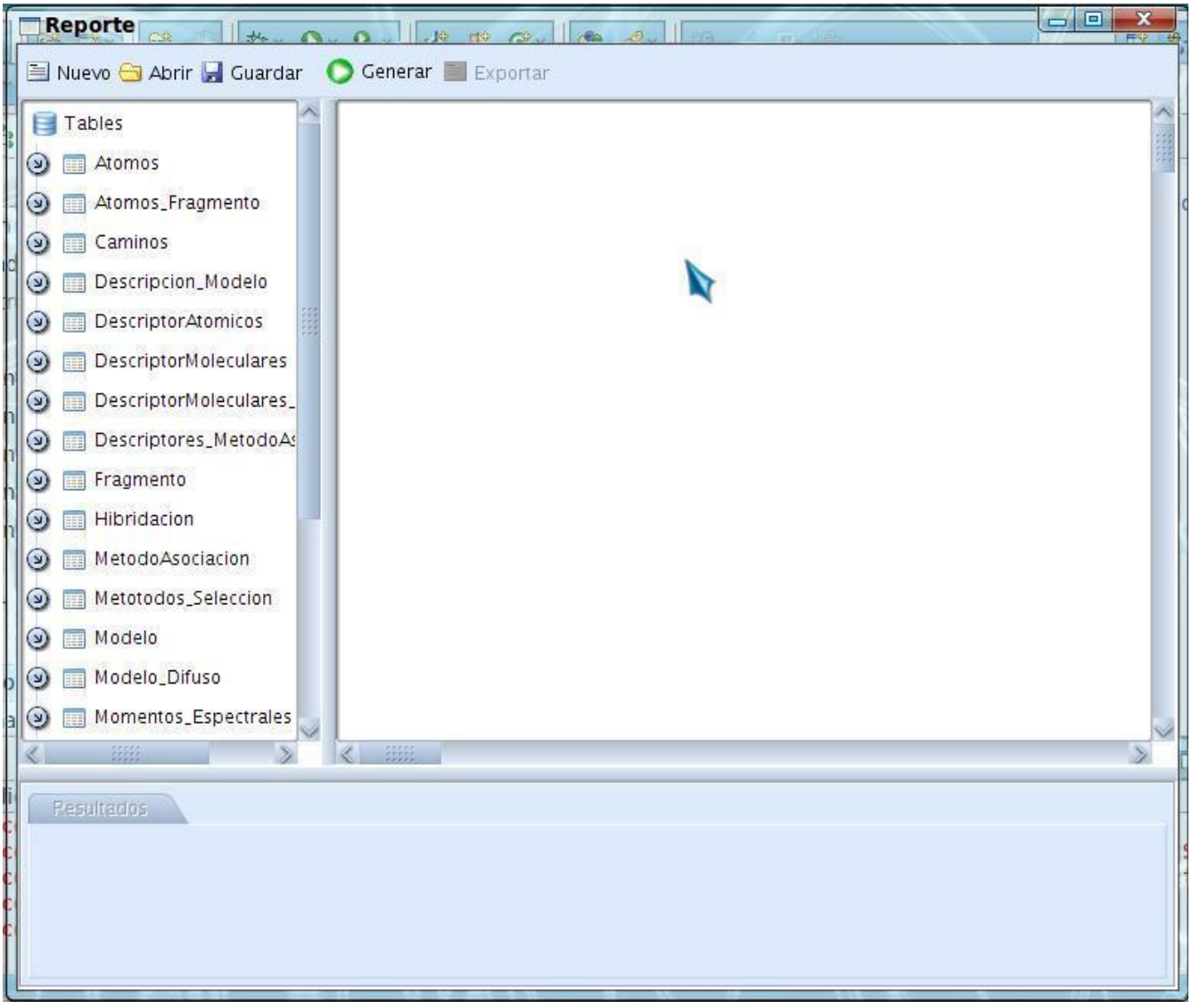


Figura 23: Interfaz. Ventana principal de la aplicación.

Figura 24: Interfaz. Ventana que se crea cuando se elige un atributo o una tabla.

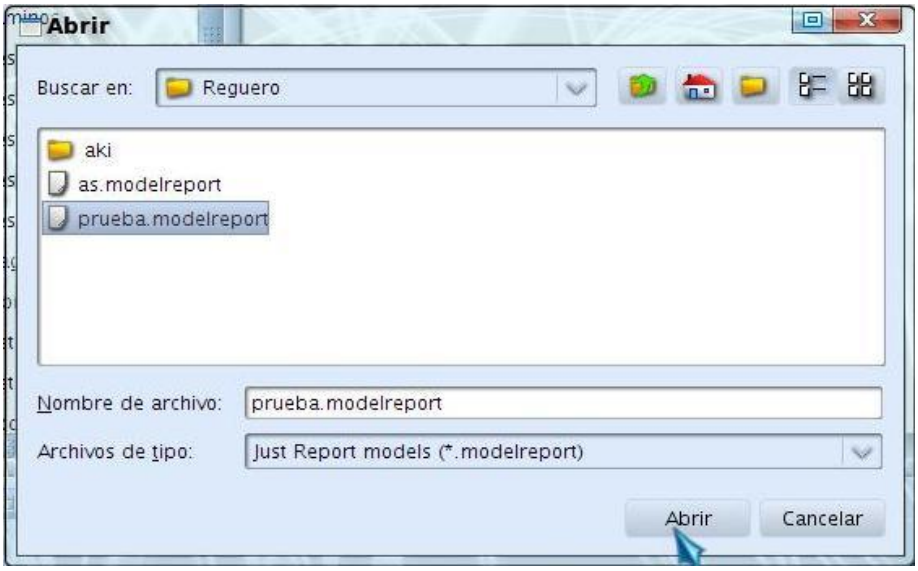


Figura 25: Interfaz. Ventana para abrir un modelo de reporte.

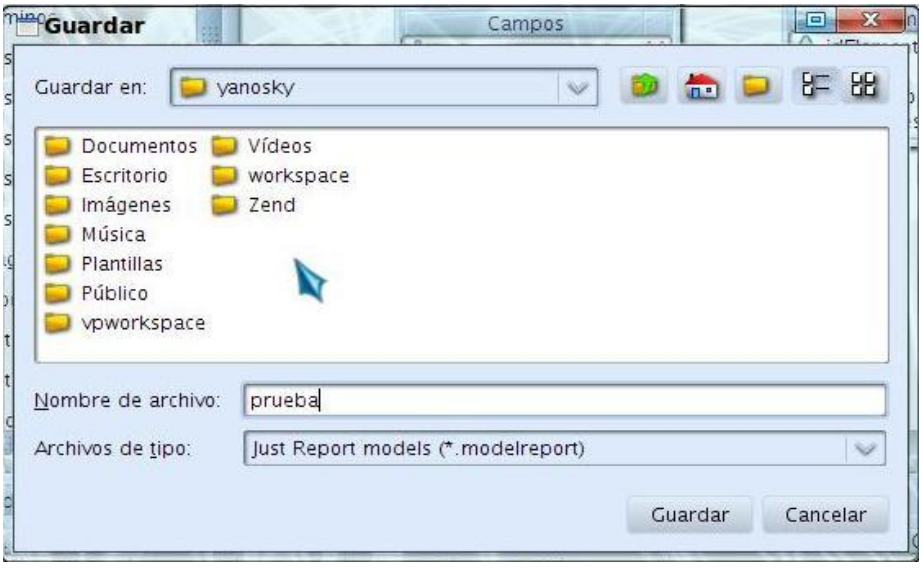


Figura 26: Interfaz. Ventana para guardar un modelo de reporte.

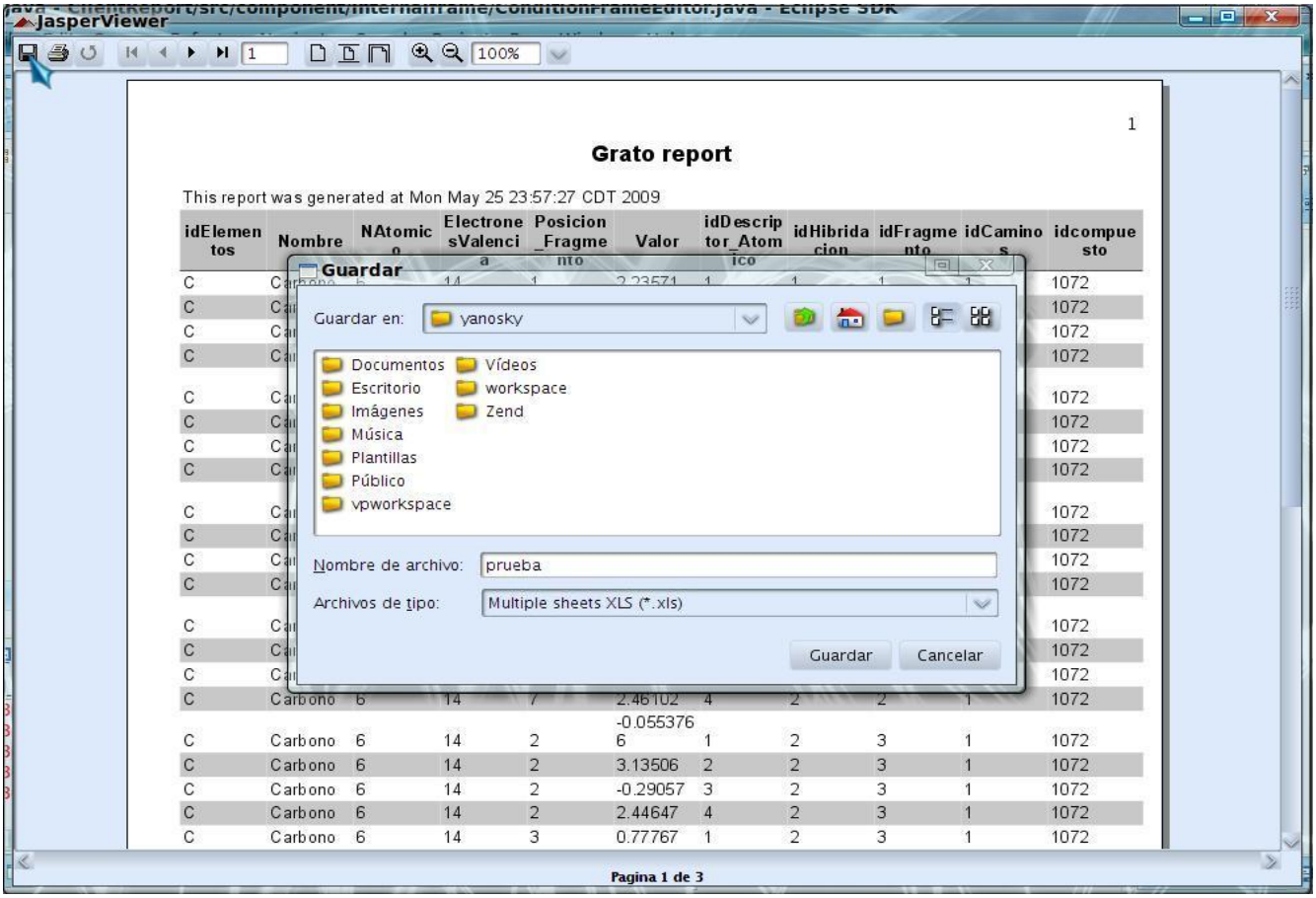


Figura 27: Interfaz. Ventana para exportar un reporte.

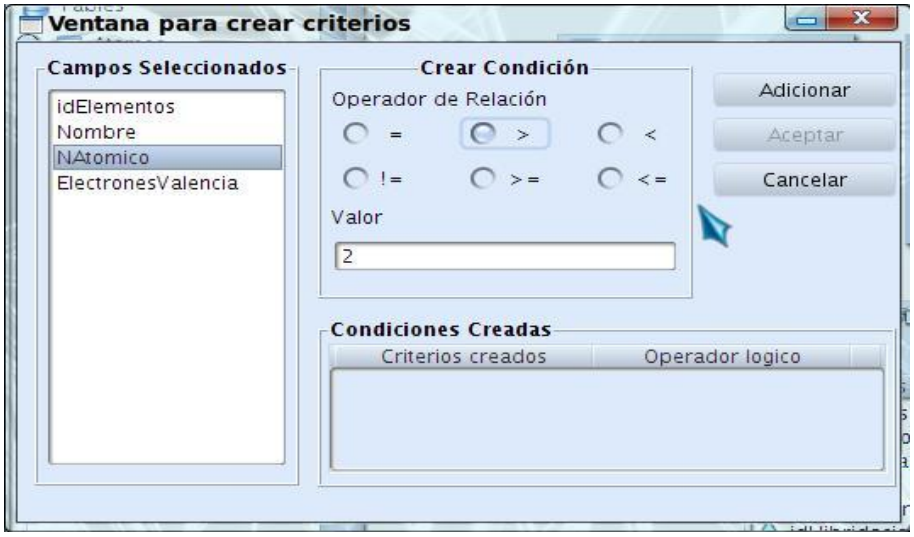


Figura 28: Interfaz. Ventana para edición de condiciones.

Anexo 2: Diagramas de Secuencia

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando se elige por parte del usuario una tabla de la cual desee que todos sus campos estén presentes en el reporte una vez generado.

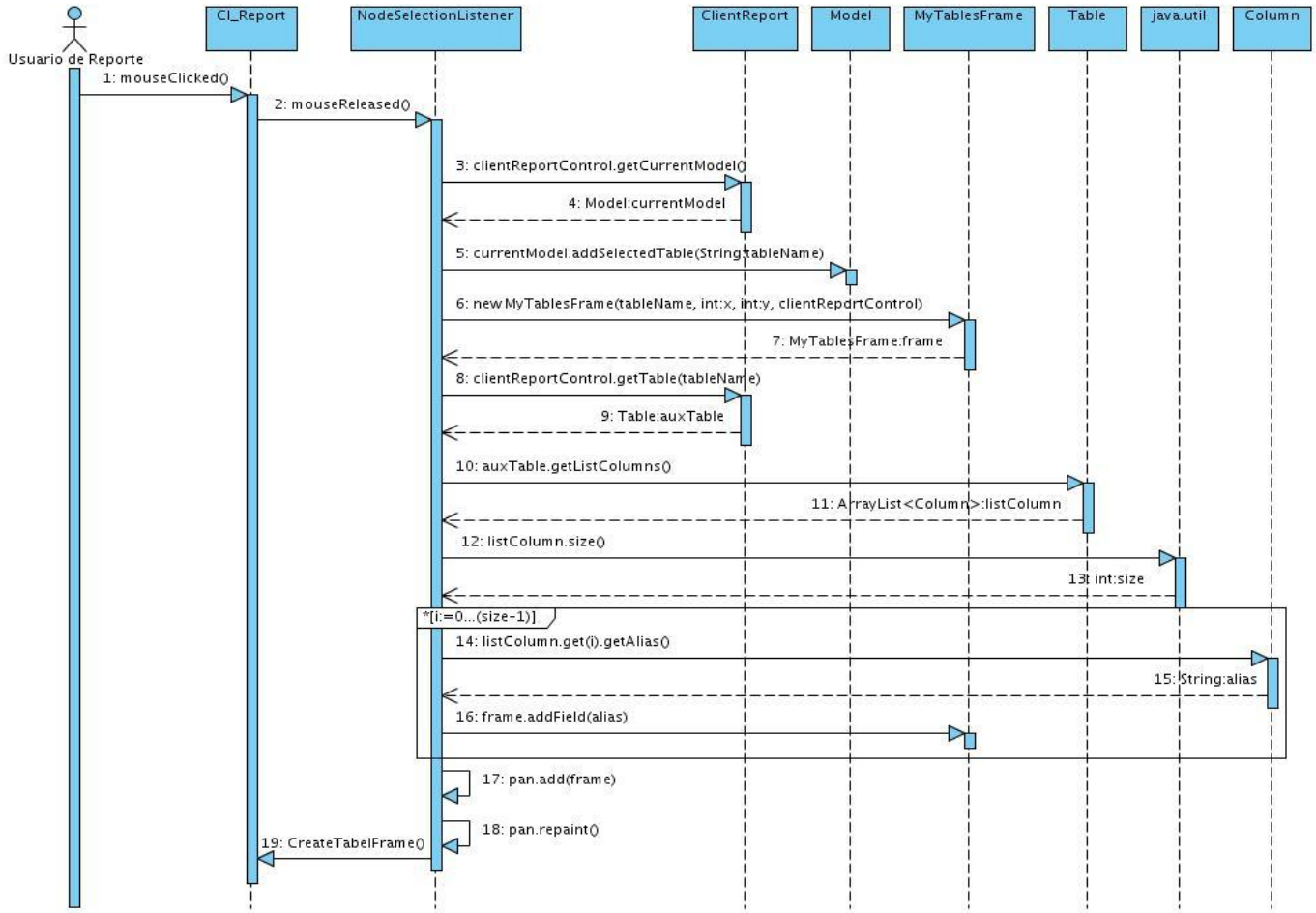


Figura 29: Diagrama de secuencia CU Elaborar Modelo de Reporte sección 2 “Elegir Tabla”.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando se elige por parte del usuario eliminar una tabla de la cual desee que ninguno de sus campos seleccionados esté presente en el modelo de reporte.

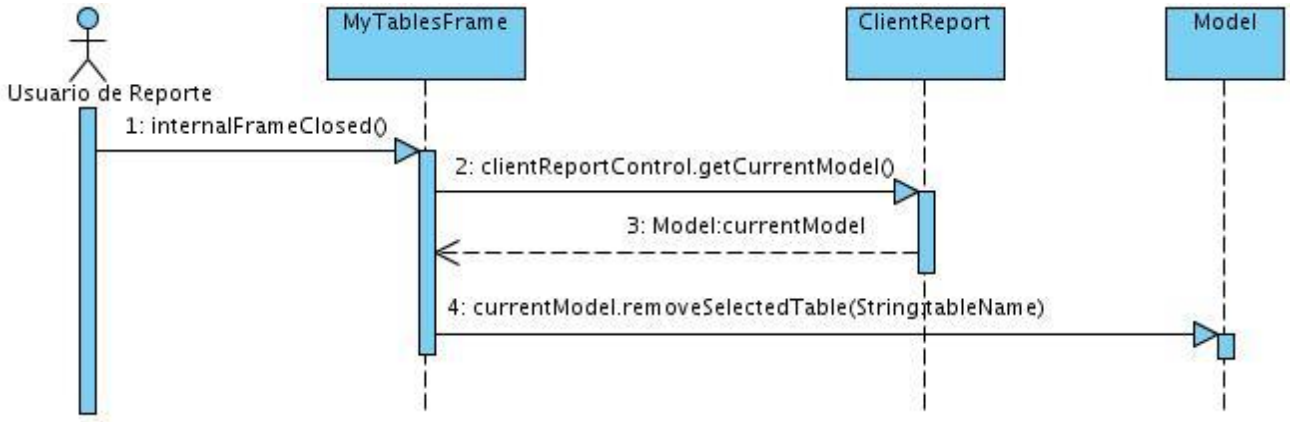


Figura 30: Diagrama de secuencia CU Elaborar Modelo de Reporte sección 3 “Cerrar ventana de la Tabla”.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando se elige por parte del usuario eliminar un atributo el cual no desee que esté presente en el modelo de reporte.

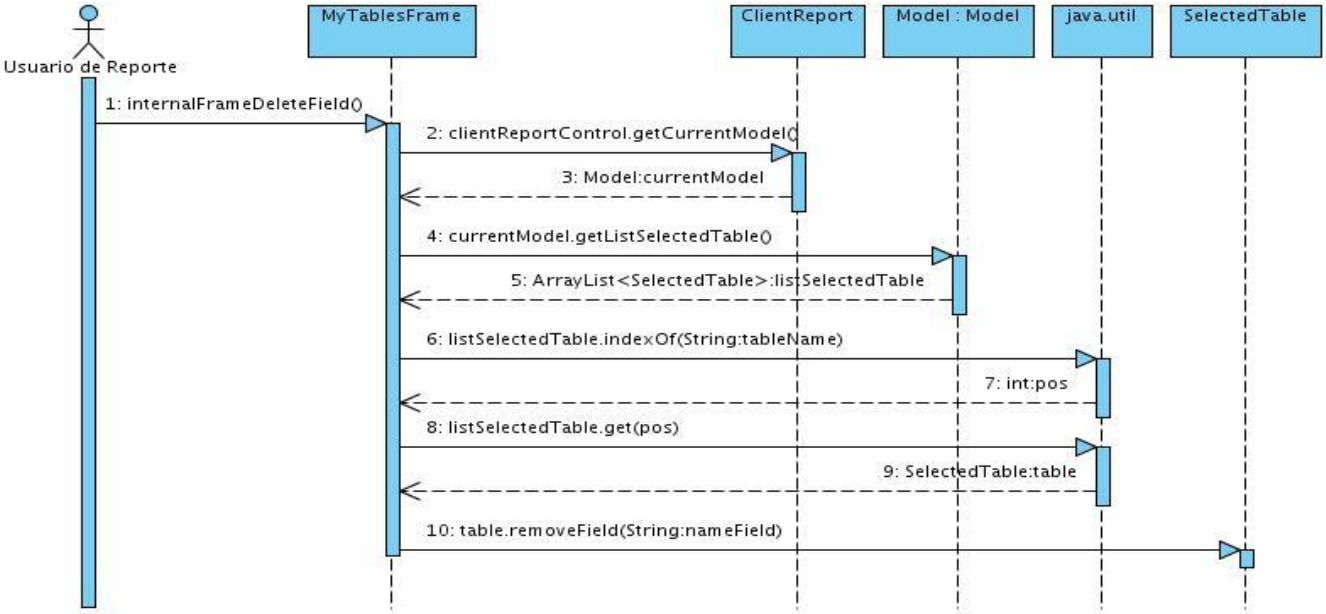


Figura 31: Diagrama de secuencia CU Elaborar Modelo de Reporte sección 4 “Eliminar Atributo”.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando el usuario se elige la acción “Editar Condiciones” para crear condiciones, sobre los atributos elegidos de una tabla determinada, a tener en cuenta en el modelo de reporte.

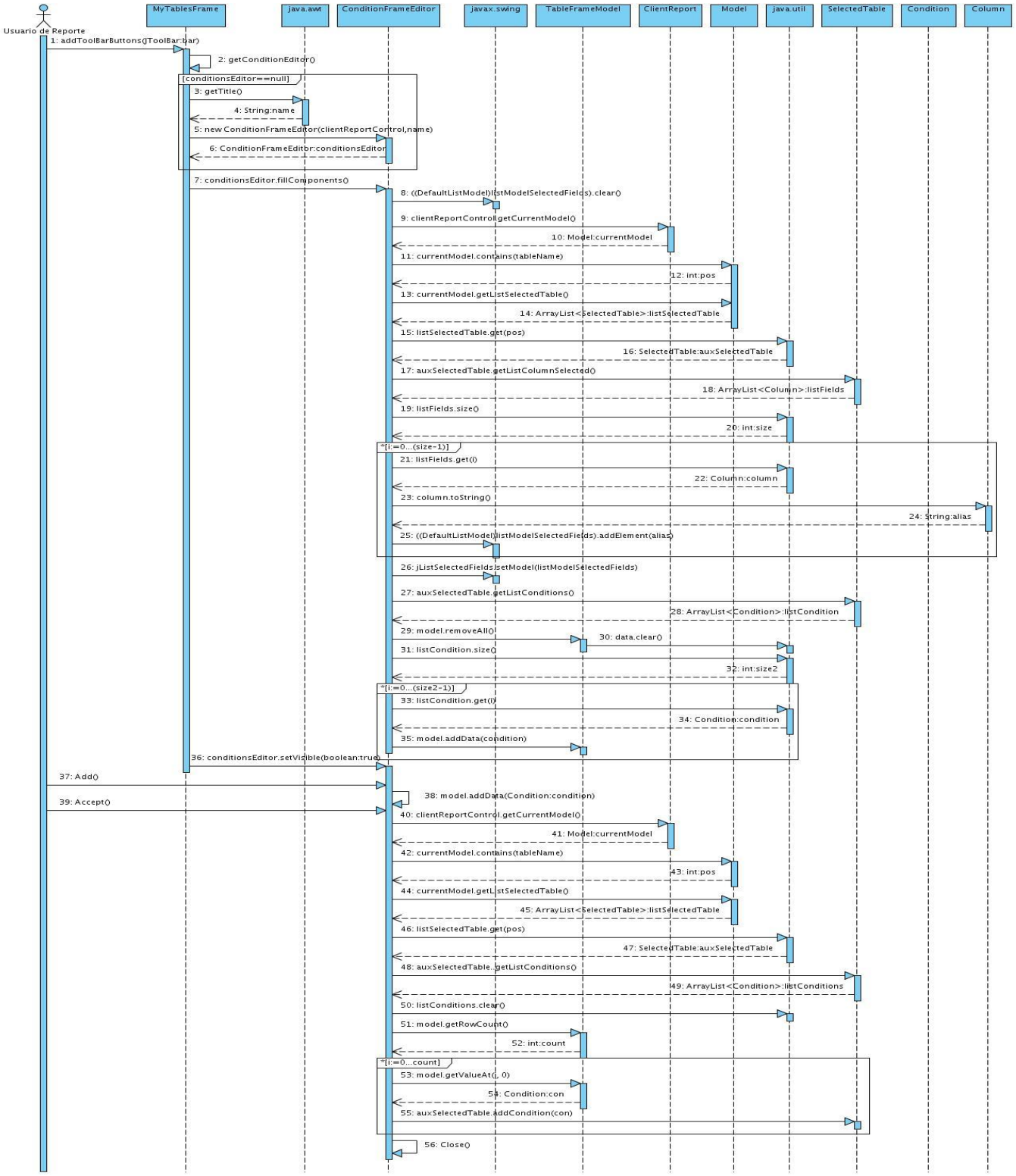


Figura 32: Diagrama de secuencia CU Editar Condiciones flujo central.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando el usuario se elige eliminar una condición creada del modelo de reporte.

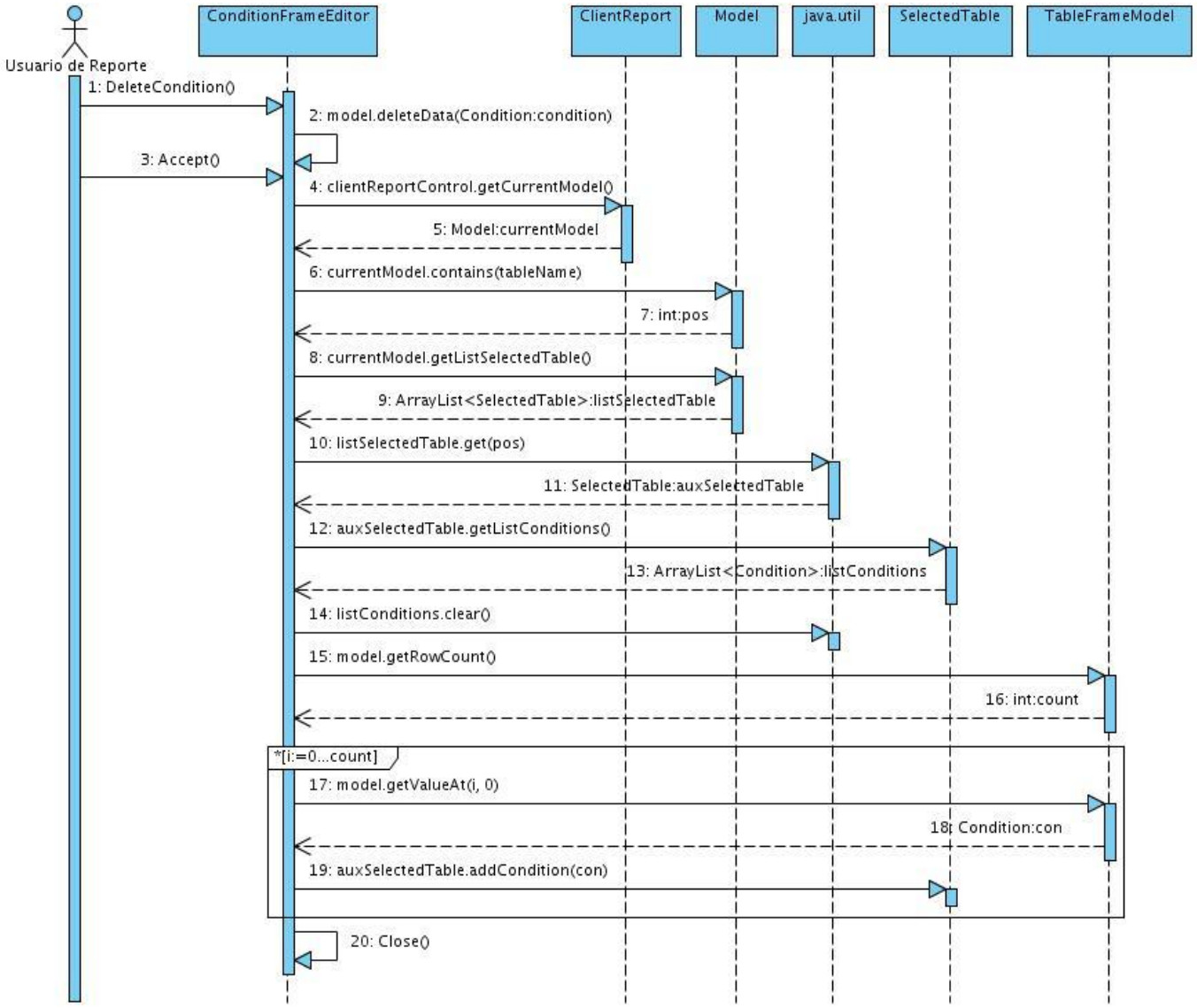


Figura 33: Diagrama de secuencia CU Editar Condiciones flujo alterno.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando se elige por parte del usuario la acción “Nuevo” para crear un nuevo modelo de reporte.

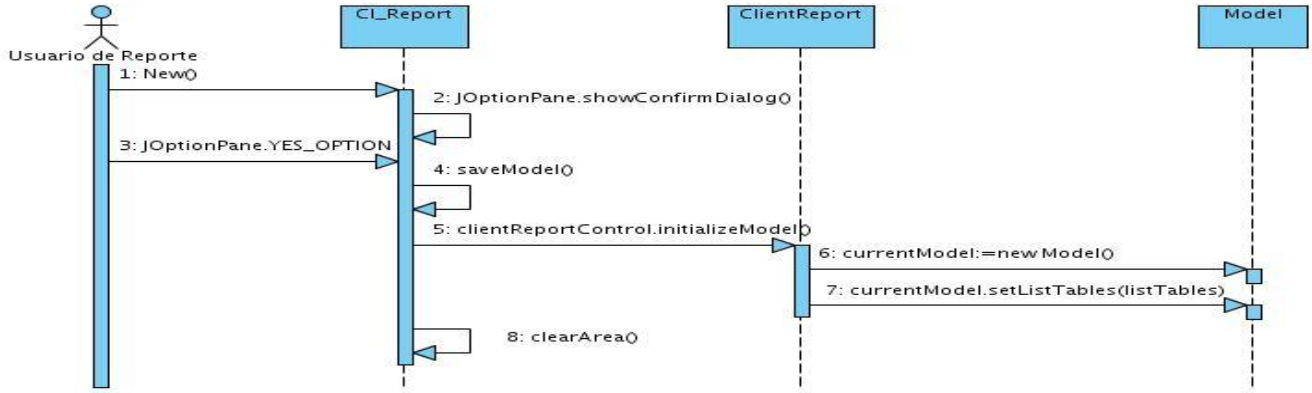


Figura 34: Diagrama de secuencia CU Gestionar Modelo de Reporte sección 1 “Nuevo”.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando el usuario se elige la acción “Abrir” para cargar un modelo de reporte guardado por el anteriormente.

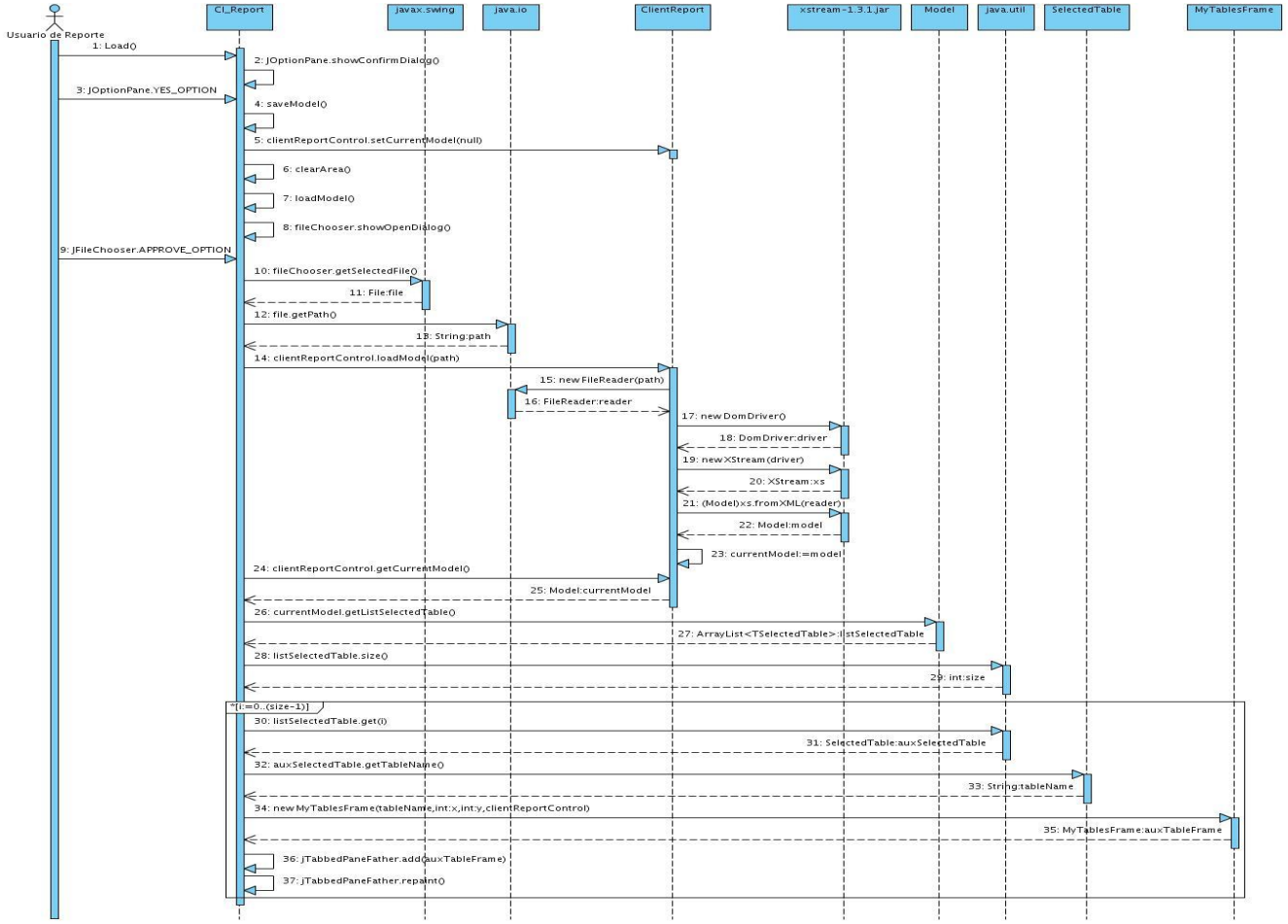


Figura 35: Diagrama de secuencia CU Gestionar Modelo de Reporte sección 2 “Abrir”.

En el diagrama a continuación se reflejan el flujo de eventos que ocurren en la aplicación cuando el usuario se elige la acción “Exportar” para exportar el reporte generado a un formato que desee.

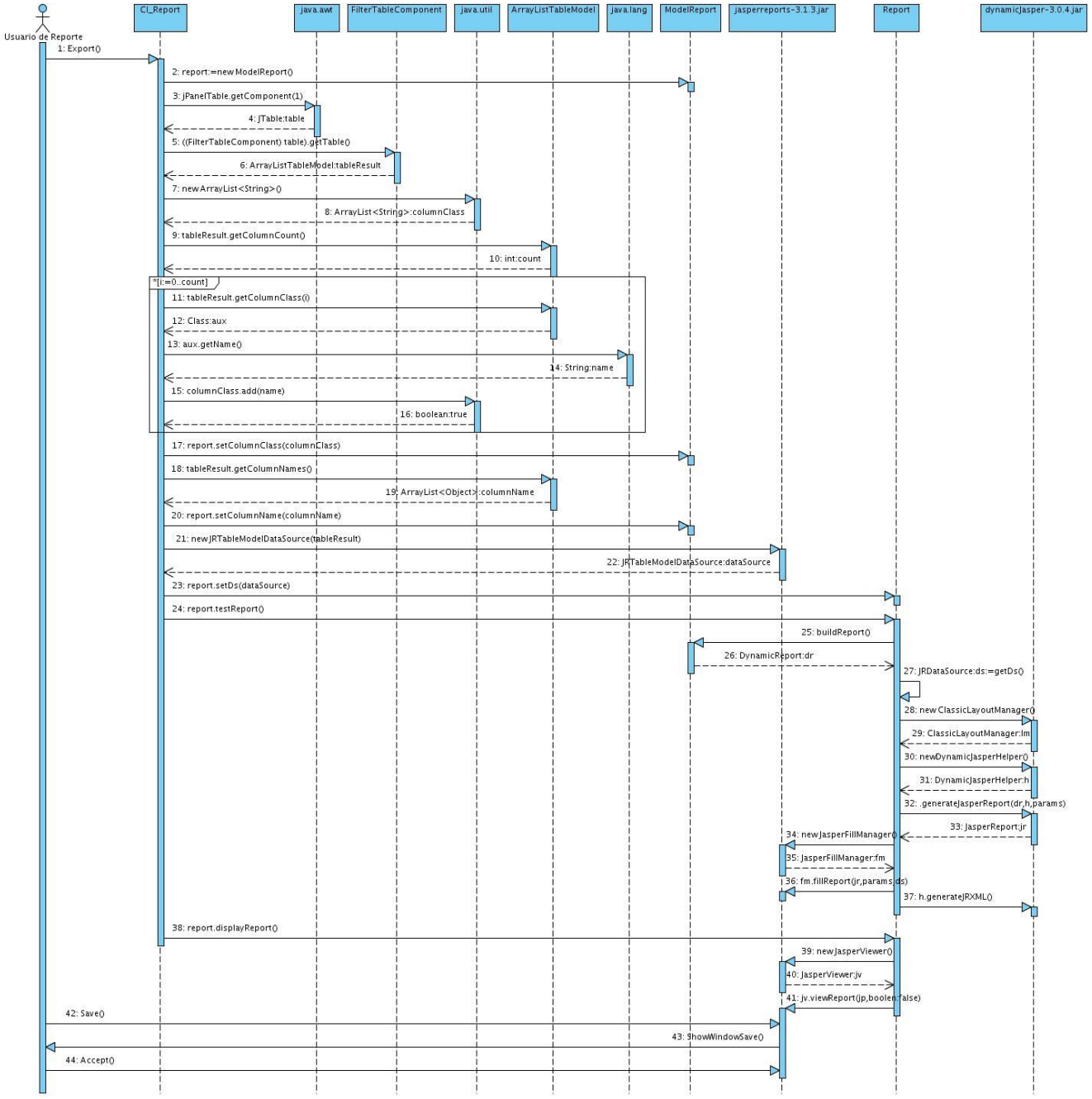


Figura 36: Diagrama de secuencia CU Procesar Reporte sección 2 “Exportar”.

Anexo 3: Descripción de clases del diseño

Nombre de la Clase:	Column
Descripción:	Modela un campo de una tabla en la base de datos.
Atributos	**Tipo**
- columnName	String
- alias	String
- dataType	String
- visible	int
Operaciones	**Descripción**
+ Column()	Constructor por defecto de la clase.
+ Column(columnName:String, alias:String, dataType:String, visible:int)	Constructor de la clase pasándole el nombre, el alias, el tipo de datos y si es visible.
+ getColumnName():String	Retorna el nombre del campo.
+ setColumnName(columnName:String): void	Permite cambiar el nombre del campo.
+ getAlias():String	Retorna el alias del campo.
+ setAlias(alias:String): void	Permite cambiar el alias del campo.
+ getDataType():String	Retorna el tipo de datos del campo.
+ setDataType(dataType:String):void	Permite cambiar el tipo de datos del campo.
+ getVisible():int	Retorna si el campo es visible.
+ setVisible(visible:int):void	Permite cambiar el estado de visibilidad del campo.
+ clone():Column	Retorna un clon de la clase.

Tabla 17: Descripción de la clase Column del CU Gestionar Reportes.

Nombre de la Clase:	KeyColumnUsage
Descripción:	Guarda las referencias de una columna de una tabla dada con otra.
Atributos	**Tipo**
- columnName	String
- referencedTableName	String
- referencedColumnName	String
Operaciones	**Descripción**

+ KeyColumnUsage()	Constructor por defecto de la clase.
+ KeyColumnUsage (columnName: String, referencedTableName:String, referencedColumnName:String)	Constructor de la clase pasándole el nombre del campo, el nombre de la tabla de la cual es llave primaria y el nombre del campo de dicha tabla al que hace referencia.
+ getColumnName():String	Retorna el nombre del campo.
+ setColumnName(columnName:String): void	Permite cambiar el nombre del campo.
+ getReferencedTableName():String	Retorna el nombre de la tabla.
+ setReferencedTableName (referencedTableName:String):void	Permite cambiar el nombre de la tabla.
+ getReferencedColumnName():String	Retorna el nombre del campo al que se hace referencia.
+ setReferencedColumnName (referencedColumnName:String):void	Permite cambiar el nombre del campo al que se hace referencia.

Tabla 18: Descripción de la clase KeyColumnUsage del CU Gestionar Reportes.

Nombre de la Clase:	Condition
Descripción:	Representa la estructura de las condiciones creadas por el usuario en el modelo de reporte.
Atributos	**Tipo**
- field	String
- operator	String
- value	String
- logic	String
Operaciones	**Descripción**
+ Condition ()	Constructor por defecto de la clase.
+ Condition (field:String, operator: String, value:String, logic:String)	Constructor de la clase pasándole el nombre del campo, el operador relacional, el valor a comparar y el operador lógico.
+ getField():String	Retorna el nombre del campo.
+ setField(field:String):void	Permite cambiar el nombre del campo.

+ getOperator():String	Retorna el operador relacional.
+ setOperator(operator:String):void	Permite cambiar el operador relacional.
+ getValue():String	Retorna el valor a comparar.
+ setValue(value:String):void	Permite cambiar el valor a comparar.
+ getLogic():String	Retorna el operador lógico.
+ setLogic(logic:String):void	Permite cambiar el operador lógico.
+ toString():String	Retorna la condición en una cadena.
+ clone():Condition	Retorna un clon de la clase.

Tabla 19: Descripción de la clase Condition del CU Gestionar Reportes.

Nombre de la Clase:	Algorithm
Descripción:	Contiene los métodos para buscar un camino entre dos tablas de la base de datos.
Atributos	**Tipo**
- tested	ArrayList<Table>
- way	ArrayList<Table>
Operaciones	**Descripción**
+ Algorithm()	Constructor por defecto de la clase.
+ getWay():ArrayList<Table>	Retorna el camino entre dos tablas.
+ clear():void	Elimina todos los elementos de las listas.
+ executeSearch(tabStart: Table, tabFinal: Table, general: ArrayList<Table>):boolean	Retorna verdadero en caso de que exista un camino entre dos tablas de la base de datos o falso en caso contrario.
+ search(tabStart: Table, tabFinal: Table, general: ArrayList<Table>):boolean	Crea el camino entre dos tablas en caso de que exista.

Tabla 20: Descripción de la clase Algorithm del CU Gestionar Reportes.

Nombre de la Clase:	ResultQuery
Descripción:	Almacena los datos resultantes generados por la consulta ejecutada sobre la base de datos.
Atributos	**Tipo**

- columnName	ArrayList<Object>
- data	ArrayList<Object>
Operaciones	**Descripción**
+ ResultQuery()	Constructor por defecto de la clase.
+ getColumnName():ArrayList<Object>	Retorna una lista de los nombres de los campos seleccionados para el reporte.
+ setColumnName(columnName: ArrayList<Object>):void	Permite cambiar la lista de nombres de los campos seleccionados.
+ getData():ArrayList<Object>	Retorna en una lista los datos generados por la consulta.
+ setData (data: ArrayList<Object>): void	Permite cambiar la lista de datos generados.

Tabla 21: Descripción de la clase ResultQuery del CU Gestionar Reportes.

Nombre de la Clase:	Report
Descripción:	Clase abstracta que contiene funcionalidades que permiten exportar el reporte a varios formatos.
Atributos	**Tipo**
~ jp	JasperPrint
~ jr	JasperReport
~ params	Map<Object, Object>
~ dr	DynamicReport
~ ds	JRDataSource
~ log	Log
Operaciones	**Descripción**
+ abstract buildReport(): DynamicReport	Método abstracto para construir el objeto DynamicReport.
+ testReport():void	Método que construye el reporte.
~ getLayoutManager():LayoutManager	Método que crea la plantilla para el reporte.
+ setDs(ds:JRDataSource):void	Método para cambiar la fuente de datos del reporte.
+ getDs():JRDataSource	Método que retorna la fuente de datos del reporte.
+ displayReport():void	Método que muestra una vista previa del reporte.

Tabla 22: Descripción de la clase Report del CU Gestionar Reportes.

Nombre de la Clase:	ModelReport
Descripción:	Clase que redefine la funcionalidad abstracta de la clase Report.
Atributos	**Tipo**
- columnName	ArrayList<Object>
- columnClass	ArrayList<String>
Operaciones	**Descripción**
+ setColumnName (arrayList: ArrayList<Object>):void	Permite modificar el nombre de los campos a mostrar en el reporte.
+ setColumnClass (columnClass: ArrayList<String>):void	Permite cambiar el tipo de datos que va a contener cada una de las columnas del reporte.
+ buildReport():DynamicReport	Método que crea el objeto DynamicReport.

Tabla 23: Descripción de la clase ModelReport del CU Gestionar Reportes.

Nombre de la Clase:	PostgresqlDAO
Descripción:	Clase que brinda funcionalidades para el trabajo sobre bases de datos implementadas en PostgreSql.
Atributos	**Tipo**
- connection	Connection
- config	Configuration
Operaciones	**Descripción**
+ MysqlDAO(xmlConfig: Configuration)	Constructor de la clase pasándole por parámetros la configuración de la pase de datos.
+ connect():void	Crea la conexión a la base de datos.
- getAllTablesName():ArrayList<Table>	Retorna todas las tablas contenidas en la base de datos.
+ getInformationFromInformationSchema(): ArrayList<Table>	Retorna toda la información del esquema de la base de datos.
+ close():void	Cierra la conexión a la base de datos.

- getReferencedColumnNamesFromInformationSchema (tableName:String, engine:String): ArrayList<KeyColumnUsage>	Retorna la lista de campos foráneos.
+ resultQuery(sql:String, begen:int, end:int):ResultQuery	Retorna los resultados de la consulta realizada sobre la base de datos.

Tabla 24: Descripción de la clase PostgresqlDAO del paquete WebService.jar del CU Gestionar Reportes.

GLOSARIO DE TÉRMINOS

Actividad Biológica: Actividad que caracteriza el comportamiento biológico en compuestos químicos.

B2B: es la abreviatura comercial de la expresión anglosajona *business to business:* (comunicaciones de comercio electrónico) de empresa a empresa.

HTML: siglas de *HyperText Markup Language* (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas web.

J2EE: (Plataforma Java 2, Edición Enterprise o J2EE hasta la versión 1.4), es una plataforma de programación -parte de la Plataforma Java- para desarrollar y ejecutar software de aplicaciones en lenguaje de programación con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

PDF: Formato de Documento Portable o en inglés *Portable Document Format* es el formato de archivos desarrollado por Adobe Systems y creado con los programas Adobe Acrobat Reader, Acrobat Capture, Adobe Distiller, Adobe Exchange, y el plugin Amber de Adobe Acrobat.

Polimorfismo: Capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

Reporte: Informe (Sinónimo), documento caracterizado por contener información u otra materia reflejando el resultado de una investigación adaptado al contexto de una situación y de una audiencia dadas.

RTF: Formato de Texto Enriquecido o en inglés *Rich Text Format*. Un método de codificar el formato de un texto y la estructura de un documento mediante el juego de caracteres ASCII. Por convención, los archivos RTF tienen una extensión ".rtf".

Script: es un guión o conjunto de instrucciones. Permiten la automatización de tareas creando pequeñas utilidades.

SGDB: Sistema Gestor de Bases de Datos relacional o RDBMS del inglés *Relational database management system* (Sistema Administrador de Bases de Datos Relacionales), software exclusivamente dedicado a tratar con bases de datos relacionales.

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