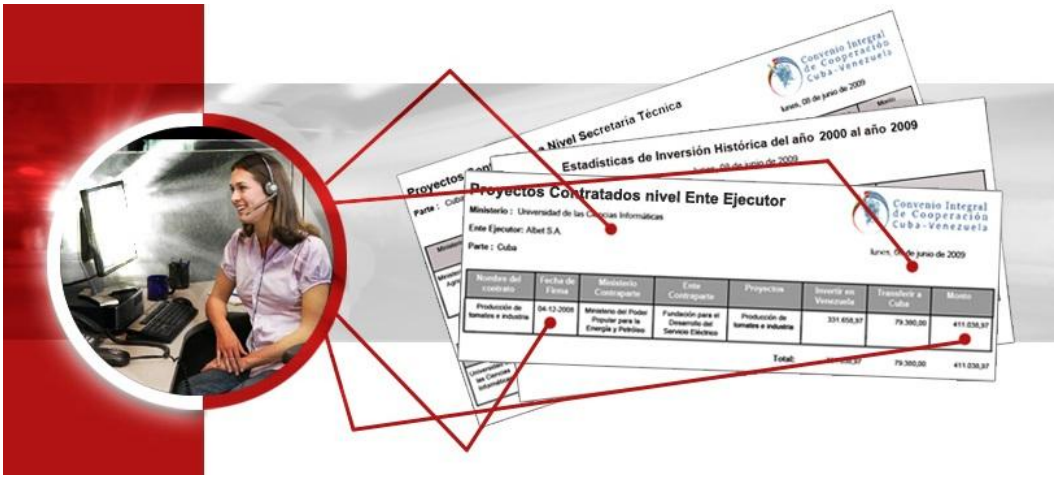
Universidad de las Ciencias Informáticas

Facultad 3



Diseño e Implementación del subsistema Reportes para el módulo Contratación del proyecto Informatización del Convenio Integral de Cooperación Cuba-Venezuela ICICCV.

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

Autor: Javier Ernesto Terrero Quevedo

Tutor: Ing. Amado Alburquerque Arias

Ciudad de la Habana, Cuba

Mayo, 2009

DECLARACIÓN DE AUTORÍA

Declaro ser el autor del presente trabajo de diploma y reconozco 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 _____

Javier Ernesto Terrero Quevedo

_____

Firma del Autor

Ing. Amado Alburquerque Arias

_____

Firma del Tutor

AGRADECIMIENTOS

A mi familia por contribuir a mi formación y por la confianza que depositaron en mí.

A mi tutor, por su abnegada y constante preocupación por este trabajo, por su experiencia y profesionalismo a la hora de inculcarme los conocimientos.

A las personas que crearon nuestra querida universidad, que se convirtió durante 5 años en nuestro hogar y por las incontables experiencias aquí vividas.

A los profesores, compañeros y amigos que ayudaron de una u otra forma al desarrollo de este trabajo.

A todos muchas gracias…

DEDICATORIA

A ustedes, sí, las tres personas más importantes de mi vida mis padres y mi hermano, porque son los inspiradores de este logro alcanzado, porque son ustedes la mejor escuela que he tenido, gracias por confiar en mí, mima eres la mejor madre y amiga del mundo, te amo, papi te quiero con la vida, estoy orgulloso de ti, tu veras que no habrá quien nos frene. Gusta gracias por pelearme y quererme tanto a la vez, de ti también aprendo, nunca te olvides de que siempre te voy a necesitar hermano, eres mi sangre. A los tres muchas gracias por existir.

A mi familia de manera general.

A mis amigos que son como mis hermanos: Leandro, hoy y siempre no habrá algo que no pueda hacer por ti, gracias por tu desinterés y por tu respeto mi hermano, Lisbet te quiero del tamaño del universo princesa linda, Julio, eres idéntico a mi hermano y te quiero como tal, gracias por todo tu apoyo, has estado conmigo desde el principio de estos 5 años, Nory, imposible de olvidarte chula, sin dudas eres lo mejor que me llevo de esta universidad, 10000000000000000 gracias por quererme tanto, te aseguro que nada será en vano. Roloooooooooooooo te quiero un mundo mi brother, que lejos vamos a estar, te voy a extrañar, gracias por estar conmigo en las buenas y en las malas. Yelanis, por ocupar tan rápido un lugar importante en mi vida, siento orgullo por ti, tus consejos son del tamaño del cielo, gracias. A Eric mi hermano sigues siendo mi guía, Ramón bro eres lo máximo. Andresito sin esta tesis no hubiera salido, nunca olvidaré eso, gracias por tu inteligencia y apoyo. Tengo la suerte de tener muchos amigos no quisiera que se quedara alguno, quiero agradecer al piquete de mi cuarto, Al Nino, Toledo, Maikel, Chung, Ornolis, Hussein, Alberto, al calvo, especialmente al Rega , a Tico a Juan, Armando, al Yoga , a Anita a Damaris,Tamara, Alionuska las quiero mucho a las 4, a todos mis amigos, no quiero que se me quede nadie, gracias a todos…..

RESUMEN

En la actualidad se manifiesta un incremento considerable en el desarrollo de aplicaciones web basadas en tecnologías de código abierto. La generación de reportes es un aspecto fundamental que se hace necesario en muchos de estos sistemas informáticos, pues mediante ellos, se materializa la gestión de la información y puede ser mostrada en formatos que hagan práctico su rendimiento de generación, su fácil entendimiento y su portabilidad.

A raíz de la decisión de desarrollar un sistema informático con altas prestaciones, que sea libre y con el objetivo de informatizar la actividad colaborativa entre los países de Cuba y Venezuela, el presente trabajo muestra el diseño e implementación de un subsistema de reportes encargado de generar parte de la información que se gestiona en el módulo Contratación de dicho sistema.

Al validar la solución propuesta, se demuestra como este módulo podrá contar con una forma eficiente de obtener reportes, en diferentes grados de complejidad, exactitud y detalle, según fue requerido por los clientes.

Palabras claves: Convenio de Cooperación Cuba-Venezuela, reportes, código abierto, JasperReport, Spring MVC framework.

ÍNDICE

DECLARACIÓN DE AUTORÍA ..... **I**

AGRADECIMIENTOS ..... **I**

DEDICATORIA ..... **II**

RESUMEN ..... **III**

INTRODUCCIÓN **1**

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA **5**

1.1 INTRODUCCIÓN 5

1.2 METODOLOGÍA DE DESARROLLO DE SOFTWARE 5

1.3 LENGUAJE DE PROGRAMACIÓN 9

1.4 PLATAFORMA DE PROGRAMACIÓN 10

1.5 ENTORNO DE DESARROLLO INTEGRADO 12

1.6 FRAMEWORK 13

1.6.1 MVC en Spring 14

1.7 PATRONES DE DISEÑO 17

1.8 MÉTRICAS PARA EL DISEÑO 19

1.8.1 Métricas de diseño arquitectónico 19

1.8.2 Métricas de diseño a nivel de componentes 20

1.8.2.1 Métricas de cohesión 21

1.8.2.2 Métricas de acoplamiento 22

1.8.2.3 Métricas de complejidad 23

1.8.3 Métricas orientadas a clases 23

1.8.3.1 Tamaño de clase (TC) 24

1.9 PRUEBAS DEL SOFTWARE 25

1.9.1.1 Pruebas de caja blanca 26

1.9.1.2 Pruebas de caja negra 26

1.9.1.3 Pruebas de unidad 27

1.9.1.4 Prueba de integración 28

1.9.1.5 Prueba del sistema ..... **¡Error! Marcador no definido.**

1.9.1.6 Pruebas de regresión 28

1.10 HERRAMIENTAS Y TECNOLOGÍAS PARA CREAR REPORTES 29

1.10.1 IText 29

1.10.2 FOP 30

1.10.3 JasperReports 31
1.10.3.1 Proceso de creación de un Reporte con JasperReport 33
1.10.3.2 Integración de JasperReports con Spring framework 34
1.10.3.3 Herramienta para el diseño visual de los reportes con JasperReports 35
1.10.4 Selección de la herramienta apropiada para la creación de reportes. 35
1.11 CONCLUSIONES DEL CAPÍTULO 36

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA 37

2.1. INTRODUCCIÓN 37
2.2. DISEÑO 38
2.2.1. Subsistema de diseño 38
2.2.2. Paquetes de diseño 40
2.2.3. Diagramas de clases de diseño. 40
2.2.4. Patrones de diseño empleados 41
2.2.5. Descripción de las clases 44
2.2.6. Representación de los diagramas de interacción del diseño (Diagramas de Secuencia) 48
2.3. IMPLEMENTACIÓN 49
2.3.1. Representación gráfica del diagrama de componentes. 49
2.3.2. Estructura para el Desarrollo del Sistema 51
2.3.3. Convenciones de Archivos y Paquetes 52
2.3.3.1. Ubicación del los ficheros de configuración del Spring 53
2.3.4. Estándar de Codificación 54
2.3.5. Representación del código fuente de los principales componentes. 57
2.2.5.1. Representación del código de clases diseñadas. 57
2.3.5.2. Representación del código de los archivos de configuración del Spring. 58
2.4. CONCLUSIONES DEL CAPÍTULO 63

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA. 64

3.1. INTRODUCCIÓN 64
3.1.1. Métricas de diseño a nivel de componentes 64
3.1.1.1. Métrica de cohesión 64
3.1.1.2. Métrica de acoplamiento 67
3.1.2. Métricas orientadas a clases. Tamaño de clase (TC) 69
3.1.3. Pruebas de unidad 70
3.1.4. Pruebas de caja negra 73
3.1.4.1. Diseño de Casos de Prueba 74

3.2. CONCLUSIONES DEL CAPITULO 81

CONCLUSIONES **82**

RECOMENDACIONES **83**

BIBLIOGRAFÍA **84**

ANEXOS **86**

ANEXO 1 - REPRESENTACIÓN DEL DIAGRAMA DE CASOS DE USO. **86**

ANEXO 2 - ESPECIFICACIÓN DE CASOS DE USO. **87**

ANEXO 3 - DESCRIPCIÓN DE LAS CLASES QUE CONFORMAN EL SUBSISTEMA REPORTES **103**

ANEXO 4- DIAGRAMAS DE CLASES DE DISEÑO POR CU. **114**

ANEXO 5- DIAGRAMAS DE SECUENCIA POR ESCENARIO DE CU **117**

ANEXO 6 - PRUEBAS DE CAJA BLANCA APLICADAS AL SUBSISTEMA REPORTES **120**

ANEXO 7 – IMÁGENES DEL DISEÑO DE LAS PLANTILLAS DE LOS REPORTES EN IREPORT. **123**

ANEXO 8 – REPRESENTACIÓN DEL CÓDIGO FUENTE DE LOS PRINCIPALES COMPONENTES. **124**

ANEXO 9: ACTA DE ACEPTACIÓN DEL MÓDULO CONTRATACIÓN **135**

ANEXO 10: ACTA DE FINIQUITO. **136**

GLOSARIO **138**

ÍNDICE DE IMÁGENES

FIGURA 1.1: MVC EN SPRING. 15

FIGURA 1.2: PROCESO DE CREACIÓN DE UN REPORTE CON JASPERREPORT. 33

FIGURA 2.1: OBJETIVO DEL MÓDULO DE CONTRATACIÓN DE PROYECTOS. 37

FIGURA 2.2: SUBSISTEMAS DE DISEÑO DEL MÓDULO CONTRATACIÓN DEL SISTEMA ICICCV. 39

FIGURA 2.3: PAQUETES DE DISEÑO DEL SUBSISTEMA REPORTES DEL MÓDULO CONTRATACIÓN. 40

FIGURA 2.4 : DIAGRAMA DE CLASES DE DISEÑO, CU GENERAR REPORTES DE PROYECTOS CONTRATADOS A NIVEL DE ENTE EJECUTOR 41

FIGURA 2.5: EJEMPLO DE PATRÓN EXPERTO 42

FIGURA 2. 6: EJEMPLO DE PATRÓN CREADOR. 43

FIGURA 2.7: DIAGRAMA DE SECUENCIA DEL CU GENERAR REPORTES DE PROYECTOS CONTRATADOS A NIVEL ENTE EJECUTOR. 49

FIGURA 2.8: DIAGRAMA DE COMPONENTES DEL SUBSISTEMA REPORTES 51

FIGURA 2.9: ORGANIZACIÓN DE PAQUETES Y SUBSISTEMAS (SUBSISTEMA REPORTES). 52

FIGURA 2.10 : ORGANIZACIÓN DE PAQUETES (INTEGRACIÓN DE SPRING CON JASPERREPORT). 54

FIGURA 2.11: DECLARACIÓN DE LAS CLASES CONTROLADORAS 55

FIGURA 2.12: DECLARACIÓN DE LAS CLASES ENTIDADES 55

FIGURA 2.13: DECLARACIÓN DE LAS CLASES DE APOYO. 56

FIGURA 2.14: ARCHIVO DE CONFIGURACIÓN DEL SPRING (WEB.XML (PARTE 1)) 58

FIGURA 2.15: ARCHIVO DE CONFIGURACIÓN DEL SPRING (WEB.XML (PARTE 2)) 59

FIGURA 2.16 : ARCHIVO DE CONFIGURACIÓN DEL SPRING (CONTRATACIÓN-APPLICATIONCONTEXT-REPORTE.XML) 60

FIGURA 2.17: ARCHIVO DE CONFIGURACIÓN DEL SPRING (COMMON-APPLICATIONCONTEXT-WEB.XML) 61

FIGURA 2.18 : ARCHIVO DE CONFIGURACIÓN DEL SPRING (REPORTES.VIEWS.XML) 62

FIGURA 3.1 RELACIONES DE USO DE LAS CLASES. 66

FIGURA 3.2 EXPANSIÓN DEL SUBSISTEMA REPORTES 68

FIGURA 3.3 RESULTADO DE LAS ITERACIONES DE PRUEBAS DE UNIDAD 71

FIGURA 3.4: PRUEBA REALIZADA A LA CLASE CREATEDATASOURCE (VISTA GENERAL, JUNIT INTEGRADO AL ECLIPSE).9 72

FIGURA 3.5: PRUEBA REALIZADA A LA CLASE CREATEDATASOURCE (VISTA ESPECIFICA DEL RESULTADO DE LOS MÉTODOS VALIDADOS). 73

FIGURA 3.6: NO CONFORMIDAD DETECTADA EN EL REPORTE PROYCONTRATADOSNIVELEE. 79

FIGURA 3.7: NO CONFORMIDAD DETECTADA EN EL REPORTE PROYCONTRATADOSNIVELST. 79

FIGURA 3.8: NO CONFORMIDADES DETECTADAS EN LAS ITERACIONES DE PRUEBA DE CAJA NEGRA 80

FIGURA A.1 : DIAGRAMA DE CASOS DE USO DEL SUBSISTEMA REPORTES 86

FIGURA A1 : DIAGRAMA DE CLASES DE DISEÑO, CU GENERAR REPORTES DE PROYECTOS CONTRATADOS A NIVEL DE MINISTERIO. ... 114

FIGURA A2 : DIAGRAMA DE CLASES DE DISEÑO, CU GENERAR REPORTES DE PROYECTOS CONTRATADOS A NIVEL DE SECRETARÍA TÉCNICA. 115

FIGURA A3 : DIAGRAMA DE CLASES DE DISEÑO, CU GENERAR REPORTES DE PROYECTOS NO CONTRATADOS A NIVEL DE ENTE EJECUTOR. 115

FIGURA A4 : DIAGRAMA DE CLASES DE DISEÑO, CU GENERAR REPORTES DE PROYECTOS NO CONTRATADOS A NIVEL DE MINISTERIO. 116

FIGURA A5 : DIAGRAMA DE CLASES DE DISEÑO, CU GENERAR REPORTES DE PROYECTOS NO CONTRATADOS A NIVEL DE SECRETARÍA TÉCNICA. 116

FIGURA A6: DIAGRAMA DE SECUENCIA DEL CU GENERAR REPORTES DE PROYECTOS CONTRATADOS A NIVEL DE MINISTERIO. 117

FIGURA A7: DIAGRAMA DE SECUENCIA DEL CU GENERAR REPORTES DE PROYECTOS CONTRATADOS A NIVEL DE SECRETARIA TÉCNICA. 118

FIGURA A8: DIAGRAMA DE SECUENCIA DEL CU GENERAR REPORTES DE PROYECTOS NO CONTRATADOS A NIVEL DE ENTE EJECUTOR. 118

FIGURA A9 : DIAGRAMA DE SECUENCIA DEL CU GENERAR REPORTES DE PROYECTOS NO CONTRATADOS A NIVEL DE MINISTERIO 119

FIGURA A10: DIAGRAMA DE SECUENCIA DEL CU GENERAR REPORTES DE PROYECTOS NO CONTRATADOS A NIVEL DE SECRETARÍA TÉCNICA. 119

FIGURA A11: ITERACIONES DE PRUEBA DE CAJA BLANCA HECHAS EN JUNIT ..... ¡ERROR! MARCADOR NO DEFINIDO.

FIGURA A12: PRUEBA REALIZADA A LA CLASE PROYCONTRATADOSNIVELEECONTROLLER. 120

FIGURA A13: PRUEBA REALIZADA A LA CLASE PROYCONTRATADOSNIVELMCONTROLLER. 120

FIGURA A14: PRUEBA REALIZADA A LA CLASE PROYCONTRATADOSNIVELSCONTROLLER. 121

FIGURA A15: PRUEBA REALIZADA A LA CLASE PROYNOCONTRATADOSNIVELEECONTROLLER. 121

FIGURA A16: PRUEBA REALIZADA A LA CLASE PROYNOCONTRATADOSNIVELMCONTROLLER. 121

FIGURA A17: PRUEBA REALIZADA A LA CLASE PROYCONTRATADOSNIVELSCONTROLLER. 122

FIGURA A18: DISEÑO DE LA PLANTILLA CORRESPONDIENTE AL REPORTE PROYECTOS NO CONTRATADOS A NIVEL DE ENTE EJECUTOR. 123

ÍNDICE DE TABLAS

TABLA A1- GENERAR REPORTE DE PROYECTOS CONTRATADOS A NIVEL DE EE 87

TABLA A2- GENERAR REPORTE DE PROYECTOS CONTRATADOS A NIVEL DE MINISTERIO 89

TABLA A3- GENERAR REPORTE DE PROYECTOS CONTRATADOS A NIVEL DE ST 93

TABLA A4- GENERAR REPORTE DE PROYECTOS NO CONTRATADOS A NIVEL DE EE 95

TABLA A5- GENERAR REPORTE DE PROYECTOS NO CONTRATADOS A NIVEL DE MINISTERIO 98

TABLA A6 - GENERAR REPORTE DE PROYECTOS NO CONTRATADOS A NIVEL DE ST 100

TABLA A7 - DESCRIPCIÓN DE LA CLASE PROYCONTRATADOSNIVELMCONTROLLER. 103

TABLA A8 - DESCRIPCIÓN DE LA CLASE PROYCONTRATADOSNIVELSCONTROLLER. 104

TABLA A9 - DESCRIPCIÓN DE LA CLASE PROYNOCONTRATADOSNIVELEECONTROLLER. 105

TABLA A10 - DESCRIPCIÓN DE LA CLASE PROYNOCONTRATADOSNIVELMCONTROLLER. 106

TABLA A11 - DESCRIPCIÓN DE LA CLASE PROYNOCONTRATADOSNIVELSCONTROLLER. 106

TABLA A12 - DESCRIPCIÓN DE LA CLASE REPORTEPCNIVELM 107

TABLA A13 - DESCRIPCIÓN DE LA CLASE REPORTEPCNIVELS 108

TABLA A14 - DESCRIPCIÓN DE LA CLASE REPORTEPNOCNIVELEE 110

TABLA A15 - DESCRIPCIÓN DE LA CLASE REPORTEPNOCNIVELM 110

TABLA A16 - DESCRIPCIÓN DE LA CLASE REPORTEPNOCNIVELS 111

TABLA A 17 - DESCRIPCIÓN DE LA CLASE CONTRATODATASOURCE_SECRETARIA 112

TABLA A18 - DESCRIPCIÓN DE LA CLASE PROYECTODATASOURCE_SECRETARIA 113

TABLA A19 - DESCRIPCIÓN DE LA CLASE PROYECTODATASOURCE 113

INTRODUCCIÓN

La clave para alcanzar y mantener el éxito en las organizaciones se ha venido redefiniendo para dar paso a la propuesta de lograr una colaboración estrecha entre la mayor parte de los Sistemas de Información de sus compañías, de modo que se aceleren los procesos administrativos que normalmente consumen mucho tiempo en tareas repetitivas, y a maximizar el flujo de información precisa y oportuna entre los empleados. El compartir información permite organizar y administrar proyectos o tareas de forma más eficiente. De este modo, los sistemas de Colaboración Empresarial no solo aumentan los niveles de productividad y competitividad de las empresas, sino que a través de las nuevas tecnologías pueden tener un mayor alcance de sus empleados y contar con una información más precisa, útil y siempre actualizada.

En el año 2000 se firma el Convenio Integral de Cooperación Cuba – Venezuela (CICCV), donde ambos gobiernos acuerdan elaborar programas y proyectos de cooperación, con el interés común de fomentar el progreso de sus respectivas economías y las ventajas recíprocas que resultan de una cooperación que tenga efectos positivos en el avance social de ambos países, y la integración con los países de América Latina y el Caribe.

Para la ejecución de estos proyectos, se cuenta con la participación de organismos públicos y privados de ambos países, universidades y organizaciones no gubernamentales. Cuba presta los servicios y tecnologías que estén a su alcance para apoyar el programa de desarrollo económico y social en Venezuela. Estos programas son definidos cada año precisando el monto monetario, las especificaciones, regulaciones y modalidades en las que son entregados, y a su vez estos bienes y servicios son pagados por Venezuela.

Una vez firmados por ambas naciones, estos proyectos comienzan a presentar una serie de dificultades en el seguimiento, revisión, y aprobación de los aspectos relacionados con los mismos. La información que se maneja, ya sea legal o financiera, presenta problemas en su control y gestión. No existe una herramienta informática que le permita a los funcionarios que manejan estos proyectos contar con las prestaciones necesarias para llevar un control adecuado de la evolución de los mismos.

Ante tal situación se puede apreciar la necesidad de emplear herramientas de gestión de proyectos especializados en relaciones colaborativas para efectuar las negociaciones enmarcadas en el Convenio de Colaboración, que amplíen la capacidad y rapidez de respuesta de las organizaciones e

instituciones representadas por las partes contractuales, que tienen bajo su responsabilidad la ejecución de los proyectos.

Después de un análisis minucioso de las características de las distintas herramientas colaborativas existentes, se determinó que sus funcionalidades no cubren todos los requerimientos exigidos por el Convenio Integral de Cooperación Cuba – Venezuela, dado que los procesos identificados en el marco del Convenio poseen especificidades propias que no coinciden con los procesos comunes de colaboración entre entidades.

En este sentido y a raíz de las dificultades en la gestión y seguimiento de los proyectos, anteriormente expuestas, surge la idea de desarrollar un sistema informático con altas prestaciones, que consiste principalmente en la informatización de la actividad colaborativa entre ambos países.

El desarrollo del sistema se dividió en siete módulos estructurados en dependencia de las diferentes funcionalidades identificadas, entre ellos se encuentra el de Contratación, el cual tiene como su principal objetivo gestionar el proceso de contratación de los proyectos que fueron aprobados en las Comisiones Mixtas y se les proporcionó financiamiento.

Para poder gestionar eficientemente este proceso de contratación, el módulo pretende resolver un conjunto de situaciones, dentro de ellas, la forma en que se reportan los resultados de la información que se gestiona de los proyectos. Este trabajo desde el surgimiento del Convenio de Cooperación Cuba-Venezuela se ha realizado de forma manual, lo que trajo aparejado:

- Demora en la generación de los reportes.
- Deterioro, pérdida o duplicidad de la información acerca del estado de los proyectos, entre los distintos niveles involucrados que forman parte del flujo informativo.
- Gasto innecesario de papel y otros recursos materiales.
- Demora en el momento de tomar decisiones para contratar un proyecto, lo que puede traer como consecuencia, inclusive, pérdida de dinero y que se dejen de realizar obras en beneficio de ambos pueblos.

Es por ello que se deberá contar con una forma más eficiente de obtener reportes que permita la validación y confiabilidad de la información económica extraída en este proceso, que ayude a tomar decisiones y que sirva para un mejor análisis de los resultados.

Analizados todos los factores anteriormente expuestos y partiendo del levantamiento de los requisitos funcionales y no funcionales por parte de los analistas que integran el proyecto ICICCV, la investigación se plantea el siguiente **problema científico**: ¿Cómo transformar los requisitos identificados en el proceso de contratación del Convenio Cuba-Venezuela en un subsistema de reportes para el módulo Contratación del proyecto ICICCV?

Dicho problema se enmarca dentro del **objeto de estudio**: Proceso de Gestión de Reportes, específicamente en el **campo de acción**: Diseño e implementación de reportes para Software Colaborativos.

Para dar cumplimiento al problema científico planteado anteriormente, el presente trabajo tiene como **objetivo general:** Diseñar e Implementar el subsistema Reportes del Módulo Contratación del proyecto ICICCV. Con **la idea a defender:** El diseño e implementación de un subsistema de reportes para el módulo Contratación del proyecto ICICCV, contribuirá a que se genere de una forma más eficiente los reportes en el proceso de contratación del Convenio Cuba-Venezuela.

Por último, con el propósito de alcanzar el objetivo general de esta investigación, se llevarán a cabo las siguientes **tareas:**

- Analizar la arquitectura de software, tecnologías, metodologías de desarrollo, estándares y técnicas de programación definidas por la dirección del proyecto ICICCV.
- Estudiar el estado del arte de las herramientas y tecnologías existentes para el diseño y la creación de reportes en una aplicación empresarial en Java, basada en tecnologías de código abierto. Selección de las apropiadas para el desarrollo de la investigación.
- Diseñar las plantillas de los reportes según la especificación de los casos de uso y prototipos no funcionales.
- Realizar el diseño y la implementación de las clases del subsistema Reportes del módulo Contratación.
- Validar la solución mediante: métricas de diseño empleadas, pruebas de caja blanca y pruebas de caja negra. Analizar los resultados.

Métodos Científicos de investigación utilizados:

Histórico-Lógico: Permitió conocer los antecedentes de las herramienta y tecnologías utilizadas para implementar reportes en sistemas con características similares y analizar sus ventajas y desventajas.

Conociendo su desarrollo histórico y las directrices, fue de gran ayuda para seleccionar la forma correcta de generar los reportes.

Analítico-Sintético: Fue útil para analizar teorías y documentos relacionados con la generación de reportes para aplicaciones empresariales, permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio. El gran volumen de información estudiado se desglosó mediante el pensamiento en conceptos abstractos y sus relaciones, a través del análisis, y luego mediante la síntesis se creó nueva información concreta, resumiendo en ella las características generales y sus relaciones.

Método de la Modelación: La investigación se auxilió de diferentes diagramas y modelos que permiten obtener distintas vistas que brindan un mayor entendimiento del cronograma a seguir. Este método además permitió la creación de modelos que representan lo que se quiere estudiar de una forma más simple, explicando lo que pasa de una manera lógica.

Estructura del trabajo de diploma:

Para el desarrollo del presente trabajo se proponen tres capítulos, los cuales están estructurados de la siguiente forma:

Capítulo 1: Fundamentación Teórica: El presente capítulo aborda el estudio de las principales tecnologías y otros elementos seleccionados por la dirección del proyecto ICICCV para la correcta construcción del software. A su vez se hace un estudio del estado del arte acerca de las diferentes herramientas que existen para la creación de reportes y se seleccionan las más adecuadas.

Capítulo 2: Diseño de la solución propuesta: Este capítulo expone la solución que se propone para el diseño y la implementación del subsistema Reportes. Para ello primeramente se analizan artefactos que propone la metodología de desarrollo de software RUP, los cuales son necesarios para obtener un adecuado modelo de diseño y guiar el proceso de desarrollo del subsistema. Posteriormente se muestra como fue implementado el mismo, teniendo en cuenta las clases que son significativas para la arquitectura y que fueron el resultado del diseño.

Capítulo 3: Implementación y validación de la solución: En este capítulo se muestra como fueron validadas las funcionalidades tantos internas como externas del subsistema Reportes y como se analizaron los resultados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

La calidad en el desarrollo y mantenimiento de software se ha convertido en uno de los principales objetivos del desarrollo de software a nivel mundial, la Universidad de las Ciencias Informáticas se hace partícipe de esta tendencia. Es por ello que en aras de construir una manera eficiente de generar los reportes para el módulo Contratación del sistema ICICCV con la calidad requerida, el presente capítulo abordará primeramente el estudio de las principales características de la metodología de desarrollo, lenguaje de programación, entorno de desarrollo integrado, patrones de diseño, framework y otros elementos que fueron seleccionados por la dirección del proyecto ICICCV como variante viable para lograr un desarrollo eficiente del sistema.

A su vez se hace un estudio del estado del arte acerca de las diferentes herramientas y tecnologías que existen para el diseño y la creación de reportes. De esta manera se podrán seleccionar las que mejor respondan a los requisitos funcionales y no funcionales identificados por los analistas principales del proyecto, para garantizar que el módulo Contratación cuente con una forma eficiente de generar sus reportes.

1.2 Metodología de desarrollo de software

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto software, definiendo una serie de pasos a seguir para obtener un software de calidad. Un proceso define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo. Las metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas abarcan procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software. (Jacaboson, et al., 2000).

La dirección del proyecto ICICCV seleccionó para la creación del producto el Proceso Unificado de Rational (**Rational Unified Process** en inglés, habitualmente resumido como RUP). A continuación se estudian sus principales características y se enfoca específicamente a los flujos de trabajo diseño e implementación que aportan elementos importantes en este estudio.

RUP

Este proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP es una infraestructura flexible de procesos de desarrollo de software que ayuda proveyendo guías consistentes y personalizadas de procesos para todo el equipo de un proyecto. Cuenta con prácticas recomendadas y probadas que dan guía para conducir las actividades de desarrollo de un equipo y que son adoptadas en cientos de proyectos mundialmente y enseñadas en cientos de universidades. Usando esta metodología se podrán alcanzar resultados predecibles unificando el equipo con procesos comunes que optimicen la comunicación y creen un entendimiento común para todas las tareas, responsabilidades y artefactos. (Jacaboson, y otros, 2000)

Como RUP es un proceso, en su modelación define como sus principales elementos:

- Trabajadores (“quién”): Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- Actividades (“cómo”): Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- Artefactos (”qué”): Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- Flujo de actividades (“Cuándo”) Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. En este caso solo se exponen los flujos de trabajos que son necesarios conocer para el desarrollo de la investigación:

- **Análisis y diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- **Implementación:** Se implementan las clases que son significativas para la arquitectura en términos de componentes.

Es importante destacar que aunque RUP contempla Análisis y Diseño en mismo flujo de trabajo por estar muy relacionadas son actividades diferentes con artefactos diferentes.

El diseño contribuye a una arquitectura estable y sólida, y crear un plano del modelo de implementación. Durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implementación.

Nos limitaremos a tratar brevemente algunos de los principales trabajadores que intervienen en los flujos de trabajo Diseño e Implementación para tener conocimiento de sus responsabilidades, así como los artefactos más significativos:

El artefacto más importante en el flujo de trabajo de Diseño es el **Modelo de Diseño** el mismo es un modelo de objetos que describe la realización de los casos de uso y al mismo tiempo constituye una abstracción del modelo de implementación y del código fuente, constituye una entrada esencial a las actividades de implementación y prueba. (Jacaboson, y otros, 2000)

RUP propone que el artefacto Modelo de Diseño básicamente contenga:

- **Introducción***:* Una descripción textual que sirve como breve introducción al modelo
- **Paquetes y Subsistemas de Diseño:** Los paquetes y subsistemas del diseño representados en una jerarquía y una breve descripción de ellos.
- **Diagramas:** los diagramas de clases del diseño y diagramas de interacción (colaboración y/o secuencia) del diseño. Estos últimos también llamados realización de casos de uso.
- **Clases, interfaces, relaciones, etc** contenidas en los paquetes y una breve descripción de ellos.

El arquitecto de software es responsable de la integridad del modelo y de que este se corresponda con la descripción de los casos de uso pero la responsabilidad de los paquetes, clases, relaciones, diagramas y realizaciones de casos se uso es del diseñador del sistema.

El **diseñador** es el responsable de diseñar una parte del sistema cumpliendo con las restricciones de los requerimientos, arquitectura y proceso de desarrollo del proyecto. Identifica y define las responsabilidades, operaciones, atributos y relaciones de los elementos de diseño.

El diseñador debe tener un sólido conocimiento de:

- Requerimientos del sistema

- Arquitectura del sistema
- Técnicas de diseño de software, incluyendo técnicas de análisis y diseño orientado a objetos y conocimiento de UML
- Tecnologías con la cuales se implementará el sistema
- Lineamientos del proyecto de como se relacionarán el diseño y la implementación, incluido el nivel de detalle del diseño antes de proceder a la implementación.

Por otra parte la implementación tiene entre sus propósitos definir la organización de código, en términos de sistemas de implementación organizados en capas. Implementar los elementos diseñados en términos de elementos de implementación y probar los componentes desarrollados como unidades. Por último integrar los resultados producidos por desarrolladores individuales o por equipos, en un sistema ejecutable. (Jacaboson, y otros, 2000)

El artefacto más importante de este flujo es el **Modelo de Implementación** el cual incluye los siguientes elementos:

- Subsistemas de implementación y sus dependencias, interfaces y contenidos.
- Componentes, incluyendo componentes fichero y ejecutables, y las dependencias entre ellos. Los componentes son sometidos a pruebas de unidad.
- La vista de arquitectura del modelo de implementación, incluyendo sus elementos arquitectónicamente significativos.

El arquitecto es el responsable de la integridad, corrección, completitud y legibilidad del modelo de implementación de acuerdo a lo descrito en el modelo de diseño, pero el encargado de implementar los elementos de este modelo es el implementador.

El **implementador** escribe código fuente, reutiliza código, compila, realiza pruebas unitarias. Si encuentra defectos en el diseño, regresa a la fase de análisis y diseño y los corrige, también arregla defectos de código y realiza pruebas unitarias para verificar los cambios. (Jacaboson, y otros, 2000)

Es importante resaltar que el flujo de implementación esta fuertemente determinado por el **lenguaje de programación.**

1.3 Lenguaje de Programación

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes. (Definición.org)

El lenguaje de programación que seleccionó el equipo de arquitectura del proyecto ICICCV para la construcción del software fue Java.

Java

Java es un lenguaje de alto nivel, de propósito general, y como tal es válido para realizar todo tipo de aplicaciones profesionales. Sus características más importantes son:

- Lenguaje orientado a objetos.
- Independiente de plataforma
- Brinda un gran nivel de seguridad
- Capacidad multihilo
- Gran rendimiento
- Creación de aplicaciones distribuidas

La combinación de características anteriormente mencionadas lo hace único y está siendo adoptado por multitud de fabricantes como herramienta básica para el desarrollo de aplicaciones comerciales y empresariales de gran repercusión.

Java resulta ser un lenguaje sencillo, no es difícil dominarlo si se tiene experiencia programando. Los desarrolladores de este lenguaje recibieron grandes influencias del paradigma orientado a objetos, lo que trae como resultado que tenga un modelo de objetos sencillo y de fácil ampliación. (Schildt, 2001)

Java está diseñado para que un programa escrito en este lenguaje sea ejecutado independientemente de la plataforma (hardware, software y sistema operativo) en la que se esté actuando. Esta portabilidad se consigue haciendo de Java un lenguaje medio interpretado medio compilado. Esto significa que se coge el código fuente, se compila a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo en que se ejecuta (llamado en el

mundo Java bytecodes). Finalmente, se interpreta ese lenguaje intermedio por medio de un programa denominado máquina virtual de Java (JVM), que sí depende de la plataforma.

Los java bytecodes permiten el ya conocido “write once, run anywhere” (compila una sola vez y ejecútalo donde quieras). Podemos compilar nuestros programas a bytecodes en cualquier **plataforma** que tenga el compilador Java. Los bytecodes luego pueden ejecutarse en cualquier implementación de la máquina virtual de Java (JVM). Esto significa que mientras la computadora tenga un JVM, el mismo programa escrito en Java puede ejecutarse en diferentes sistemas operativos. (Schildt, 2001)

1.4 Plataforma de programación

Con plataforma nos referimos al ambiente de hardware y software en donde un programa se ejecuta, por ejemplo, plataformas como Linux, Solaris, Windows 2003 y MacOS. (Ciberaula 2009). En este caso se estudia la plataforma Java, seleccionada por el equipo de arquitectura del proyecto ICICCV. Esta plataforma se diferencia en que es una plataforma sólo de software y se ejecuta sobre las otras plataformas de hardware. (Ciberaula 2009)

La plataforma Java, es un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode y un conjunto de herramientas de desarrollo. La misma está compuesta por dos componentes:

- La máquina virtual de Java (JVM)
- El Java API (Application Programming Interface)

En el epígrafe anterior se explicó el funcionamiento de la máquina virtual de Java (JVM), la cual es la base de la plataforma Java y es llevada a diferentes plataformas de hardware. El Java API es una gran colección de componentes de software que proporcionan muchas utilidades para el programador, por ejemplo, los API’s para las interfaces gráficas. Los API’s de Java están agrupados en librerías de ciertas clases e interfaces, estas librerías son conocidas como paquetes. El siguiente gráfico describe un programa que se está ejecutando sobre la plataforma Java. Como se muestra, el Java API y la máquina virtual aíslan al programa del hardware.

Figura 1. 1 Vista de un programa corriendo sobre la plataforma Java.

Esta plataforma antes era conocida como Plataforma Java 2 e incluye:

- Plataforma Java, Edición Estándar (Java Platform, Standard Edition), o **Java SE** (antes J2SE)
- Plataforma Java, Edición Empresa (Java Platform, Enterprise Edition), o **Java EE** (antes J2EE)
- Plataforma Java, Edición Micro (Java Platform, Micro Edition), o **Java ME** (antes J2ME)

El equipo de arquitectura del proyecto decidió nutrirse de las ventajas que trae como resultado utilizar la especificación **Java EE** la cual es sin dudas una gran opción para el desarrollo de aplicaciones empresariales, como lo fue para el desarrollo del software ICICCV.

Java Platform, Enterprise Edition

Java EE cuenta con una arquitectura de n niveles distribuida y basándose ampliamente en componentes de software modulares, ejecutándose sobre un servidor de aplicaciones. Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes al mismo.

Java EE configura algunas especificaciones únicas para componentes, estas incluyen Enterprise JavaBeans, servlets, JavaServer Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas, y a la vez que sea integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los

componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel. (Perrone, y otros, 2003)

1.5 Entorno de desarrollo integrado.

Un entorno de desarrollo integrado o, en inglés, Integrated Development Environment (**IDE**), es un programa compuesto por un conjunto de herramientas para un programador que puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. (SlideShare 2009)

El entorno de desarrollo integrado que fue seleccionado por el equipo de arquitectura para desarrollar el software mediante el lenguaje de programación Java, fue el Eclipse en su versión Europa 3.3.

Eclipse

Este IDE es muy apropiado y ha sido utilizado exitosamente para construir ambientes para un amplio rango de temas, como el desarrollo en lenguaje de programación Java, Servicios Web, certámenes de programación de juegos y otros.

Eclipse surge con la idea de crear una plataforma de desarrollo común para confeccionar sus productos, basados en el lenguaje de programación Java. Es un ambiente de desarrollo integrado (IDE) porque provee herramientas para administrar áreas de trabajo (o workspaces en inglés), para construir, lanzar y depurar aplicaciones, para compartir artefactos con un equipo y versionar el código fuente. Es multiplataforma, ya que se ejecuta en una gran cantidad de sistemas operativos incluyendo Windows y Linux y es abierto porque su diseño le permite ser extendido fácilmente por terceras partes. Es fácilmente integrable con la herramienta CASE Visual Paradigm y soporta perfectamente la plataforma de desarrollo **JEE**. Emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad, a diferencia de otros entornos donde las funcionalidades están todas incluidas, las necesite el usuario o no. (BEATON) .

Eclipse cuenta con un mecanismo de incorporación de plugins, como el Spring IDE, Hibernate y otros eficaces para el desarrollo sobre **frameworks** de desarrollo.

1.6 Framework

Un framework, en el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. En este caso el equipo de arquitectura seleccionó el framework Spring.

Spring's Web MVC framework

Es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Fue lanzado inicialmente bajo Apache 2.0 License en junio de 2003. El primer gran lanzamiento hito fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005.

Spring es un framework de aplicación desarrollado para aplicaciones escritas en el lenguaje de programación Java. Es que también es de código abierto, lo cual implica que no tiene ningún costo, ni se necesita una licencia para utilizarlo, por lo tanto da la libertad a muchas empresas y desarrolladores a incursionar en la utilización de esta aplicación. Spring intenta integrar las diferentes tecnologías existentes, en un sólo framework para el desarrollo más sencillo y eficaz de aplicaciones J2EE portables entre servidores de aplicación. Otro de los principales enfoques de Spring y por el cual está ganando dicha popularidad es que simplifica el desarrollo de aplicaciones J2EE, al intentar evitar el uso de Enterprise Java Beans, ya que como menciona Craig Walls en su libro Spring in Action, “En su estado actual, EJB es complicado porque EJB fue creado para resolver cosas complicadas, como objetos distribuidos y transacciones remotas. Y muchas veces aunque el proyecto no es lo suficientemente complejo, se utiliza EJB, contenedores de alto peso y otras herramientas que soportan un grado mayor de complejidad, como una solución a un proyecto. (C. Walls, 2005)

Spring fue creado basado en los siguientes principios:

- El buen diseño es más importante que la tecnología subyacente.
- Los JavaBeans ligados de una manera más libre entre interfaces es un buen modelo.
- El código debe ser fácil de probar.

Spring es un Framework para el desarrollo de aplicaciones para la plataforma Java, que interviene en todas las capas arquitectónicas de una aplicación J2EE, brinda soporte a varios frameworks de presentación, entre ellos Java Server Faces (JSF), brinda soporte a Hibernate integrándose con ellos para formar una poderosa herramienta para el desarrollo de software.

A pesar de que Spring no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar, prácticas comunes en la industria. La simplificación del desarrollo de aplicaciones y de sus respectivas pruebas es una de las claves de su éxito. Spring puede organizar de forma efectiva los objetos de la capa central y manejar sus conexiones. Además facilita unas buenas prácticas de programación orientada a objetos, por ejemplo utilizando interfaces. (C. Walls, 2005)

1.6.1 MVC en Spring

Spring brinda un MVC (Model View Controller, Modelo Vista Controlador) para web bastante flexible y altamente configurable, pero esta flexibilidad no le quita sencillez, ya que se pueden desarrollar aplicaciones sencillas sin tener que configurar muchas opciones.

El patrón arquitectónico **Modelo-Vista-Controlador (MVC)** separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista: Maneja la visualización de la información.

Controlador: Controla el flujo entre la vista y el modelo.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo, independientemente de la representación visual. MVC está demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan

presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones. (Burbeck, 2006)

El framework Spring es uno de los que implementa el patrón MVC. De esta manera el principio de funcionamiento sería el siguiente: Spring tiene implementado un servlet que realiza las tareas del patrón Front Controller, esto significa que cada uno de los request que son realizados por el usuario, pasan a través de este servlet, que se identifica con el nombre DispatcherServlet. Como la imagen lo indica el Request llega al DispatcherServlet el cual tiene la responsabilidad de delegar a otro componente el procesamiento del request. (C. Walls, 2005)

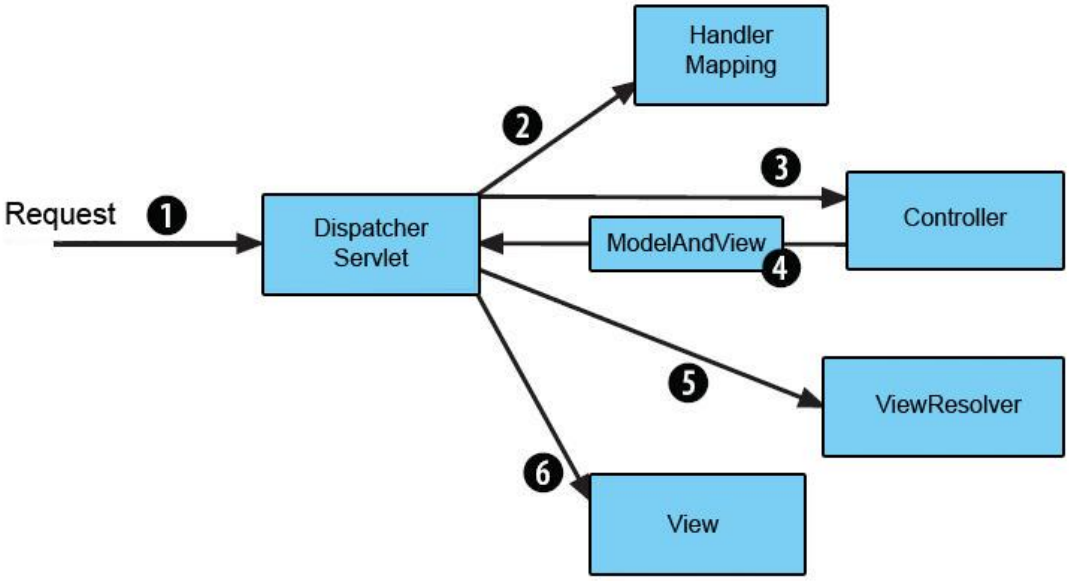


Figura 1.2: MVC en Spring.

Este componente que Spring utiliza lo que se denomina Handler Mapping, el cual tiene la función de determinar cuál será el Controller que recibirá el request. Dentro del Spring existe varios Handler Mapping los cuales tienen distintas capacidades para poder mapear a los controladores:

- **BeanNameUrlHandlerMapping**: Mapea controladores a URL basándose en el nombre del Bean
- **SimpleUrlHandlerMapping:** Mapea controladores a URL basándose en una colección de propiedades que se definen en el Spring Application Context.
- **ControllerClassNameHandlerMapping**: Mapea controladores a URL utilizando el Controller Class Name

Luego de que el Handler Mapping entrega el nombre del Controller que se hará cargo del Request, el DispatcherServlet le envía el request al Controller. Para poder implementar un Controller sobre Spring es necesario que se cree una clase que herede de los Controller que han sido implementados.

Luego que el Controller recibe el Request, se construye un Objeto que se denomina ModelAndView, este componente tiene como función:

- Entregar un nombre lógico a la vista que deberá realizar el despliegue del Model.
- Entregar un nombre lógico al Modelo que está asociado a este componente.
- Inyectar el objeto Model el cual tiene los datos que serán desplegados en la Vista.

Luego que el objeto ModelAndView es regresado al DispatcherServlet y este componente delega la responsabilidad de mapear el nombre lógico de la vista, con el componente a utilizar, el ViewResolver es el encargado de realizar el mapping entre el nombre lógico de la vista y el componente. Finalmente. Luego que la vista realiza el procesamiento, el DispatcherServlet envía el request de retorno al usuario. (C. Walls, 2005)

1.7 Herramienta CASE para el el diseño

Las **herramientas CASE** (**C**omputer **A**ided **S**oftware **E**ngineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. (Jacaboson, y otros, 2000)

Visual Paradigm

Es una herramienta CASE que utiliza UML como lenguaje de modelado. Soporta el ciclo de vida completo de desarrollo de software, ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste. Permite modelar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (Visual Paradigm 2005)

Dentro de las funcionalidades que soporta el Visual Paradigm están:

- Administración de requisitos.

- Modelado de procesos del negocio.
- Modelado de base de datos.
- Generación de código.
- Ingeniería inversa.

Se integra con herramientas Java como:

- Eclipse.
- JBuilder.
- NetBeans/sun ONE.
- Weblogic Workshop.
- JDeveloper.
- IntelliJ IDE.

Visual Paradigm brinda la posibilidad de lograr una mejor integración entre todos los involucrados en el desarrollo del software, brindándole la posibilidad de organizar los diagramas y documentación asociada al desarrollo del proyecto.

1.8 Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores, es decir que se haya comprobado que funcionan. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

En este caso se estudiaron las principales características del grupo GRASP de patrones de diseño, patrones que se aplicaron en la construcción del sistema.

GRASP

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Lo mismos constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

En cuanto a las responsabilidades UML define una responsabilidad como “un contrato u obligación de un clasificador”. (Saavedra)

Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento. Básicamente, estas responsabilidades son de los siguientes dos tipos:

Conocer: conocer los datos privados encapsulados, conocer los objetos relacionados, conocer las cosas que puede derivar o calcular.

Hacer: hacer algo él mismo, como crear un objeto o hacer un cálculo, iniciar una acción en otros objetos, controlar y coordinar actividades en otros objetos.

GRASP destaca 5 patrones principales que son: Experto, Creador, Alta cohesión, Bajo acoplamiento, Controlador**.** Y 4 patrones adicionales que son: Fabricación Pura, Polimorfismo, Indirección, No hables con extraños. A continuación se mencionan las principales características de cada uno de ellos:

Experto

- Expresa la intuición de que los Objetos hacen las cosas según la información que tienen.
- Refuerzan el encapsulamiento y esto redunda en bajo acoplamiento

Creador

- Consiste en asignar a un Objeto la responsabilidad de crear otro Objeto.
- Lo ideal es decidir estas cuestiones cuando se está diseñando y no cuando se está desarrollando y uno está inmerso entre miles de líneas de código.

Bajo Acoplamiento

- Una clase con alto acoplamiento depende de muchas otras Clases para llevar a cabo su trabajo
- Una Clase con bajo acoplamiento no depende de demasiadas Clases para hacer su trabajo.

Alta cohesión

- Las Clases se pueden reutilizar con mayor facilidad y flexibilidad
- Una Clase con baja cohesión hace muchas cosas no relacionadas

- Una Clase con alta cohesión hace lo que uno podría esperar que hiciera.
- Si el sistema fallara por alguna razón es mucho más fácil encontrar responsabilidades si las Clases del sistema son cohesivas

Controlador

- Un Controlador es un Objeto que no pertenece a la interfaz de usuario.
- Es aconsejable utilizar Controladores de Fachada cuando no existen demasiados eventos del sistema.
- Es aconsejable utilizar Controladores de Casos de uso cuando la aplicación es muy extensa. De esta forma en vez de tener un solo Controlador Fachada y saturarlo, tenemos Controladores más pequeños especializados en cada Caso de uso especificado por el analista. (Saavedra)

1.9 Métricas para el diseño

Una métrica puede ser cualquier medida o conjunto de medidas destinadas a conocer o estimar el tamaño u otra característica de un software. Estas ayudan a la evaluación de los modelos de análisis y diseño, proporcionan una indicación de la complejidad de los diseños procedimentales y del código fuente, y ayudan a realizar pruebas más efectivas. En las métricas para el modelo de diseño se genera la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del Sistema de Información. Proporcionan al diseñador una mejor visión interna y ayudan a que el diseño evolucione a un nivel superior de calidad. Se dividen en métricas de diseño arquitectónico, métricas de diseño a nivel de componente y métricas orientadas a clases.

1.9.1 Métricas de diseño arquitectónico

Éstas se concentran en las características de la estructura del programa dándole énfasis a la estructura arquitectónica y en la eficiencia de los módulos. Éstas métricas son llamadas de caja negra, ya que no requieren ningún conocimiento del trabajo interno de ningún modo en particular del sistema.

Card y Glass proponen tres medidas de complejidad del software:

- Complejidad estructural. S(i), de un módulo i se define de la siguiente manera:

$$S(i) = f^2_{out}(i)$$

Donde f out (i) es la expansión del módulo i.

- Complejidad de datos. D(i) proporciona una indicación de la complejidad en la interfaz interna de un módulo i y se define como:

$$D(i) = v(i) / [f_{out}(i) + 1]$$

Donde v (i) es el número de variables de entrada y salida del módulo i.

- Complejidad de sistema. C(i), se define como la suma de las complejidades estructural y de datos, y se define como:

$$C(i)=S(i)+D(i)$$

A medida que crecen los valores de complejidad, la complejidad arquitectónica o global del sistema también aumenta [21]

Por otra parte Henry y Kafura proponen la métrica de complejidad de expansión-concentración, que considera las estructuras de datos que recoge (concentra) o actualizan (expansión). También es representada una fórmula para definir esta métrica.

Fenton sugiere varias métricas de morfología simples las cuales permiten comparar diferentes arquitecturas de programa mediante un conjunto de dimensiones directas. (Pressman, 2005)

1.9.2 Métricas de diseño a nivel de componentes

Éstas se concentran en las características internas de los componentes del software, de ahí que son llamadas métricas de caja blanca, e incluyen medidas de la cohesión, acoplamiento y complejidad del módulo. Estás medidas ayudan al desarrollador de software a juzgar la calidad de un diseño a nivel de componentes. Puede aplicarse una vez que se haya desarrollado un diseño procedimental o pueden retrasarse hasta tener disponible el código fuente. (Pressman, 2005)

1.9.2.1 Métricas de cohesión

Bieman y Ott definen una colección de métricas que se definen con cinco conceptos y medidas:

- Porción de datos. Dicho simplemente, una porción de datos es una marcha atrás a través de un módulo que busca valores de datos que afectan a la localización del módulo en el que empezó la marcha atrás. Debería resaltarse que se pueden definir tanto porciones de programas (que se centran en enunciados y condiciones) como porciones de datos.
- Símbolos léxicos (tokens) de datos. Las variables definidas para un módulo pueden definirse como señales de datos para el módulo.
- Señales de unión. El conjunto de señales de datos que se encuentran en uno o más porciones de datos.
- Señales de super-unión. Las señales de datos comunes a todas las porciones de datos de un módulo.
- Cohesión. La cohesión relativa de una señal de unión es directamente proporcional al número de porciones de datos que liga. (Pressman, 2005)

Bieman y Ott desarrollaron métricas para las cohesiones funcionales fuertes, para las cohesiones funcionales débiles y pegajosidad. Éstas pueden ser interpretadas de la siguiente manera:

"Todas estas métricas de cohesión tienen valores que van desde 0 a 1. Tienen un valor de 0 cuando un procedimiento tiene más de una salida y no muestra ningún atributo de cohesión indicado por una métrica particular. Un procedimiento sin señales de super-unión, sin señales comunes a todas las porciones de datos, no tiene una cohesión funcional fuerte (no hay señales de datos que contribuyan a todas las salidas). Un procedimiento sin señales de unión, es decir, sin señales comunes a más de una porción de datos (en procedimientos con más de una porción de datos), no muestra una cohesión funcional débil y ninguna adhesividad (no hay señales de datos que contribuyan a más de una salida). La cohesión funcional fuerte y la pegajosidad se obtienen cuando las métricas de Bieman y Ott toman un valor máximo de 1." (UDELAB, 2009)

Para ilustrar el carácter de estas métricas, se debe la métrica para la cohesión funcional fuerte:

$$CFF(i) = SU(SA(i))/señales\ (i)$$

Donde SU(SA(i)) denota señales de super-unión (el conjunto de señales de datos que se encuentran en todas las porciones de datos de un módulo i). Como la relación de señales de super-unión con respecto al número total de señales en un módulo i aumenta hasta un valor máximo de 1, la cohesión funcional del módulo también aumenta 1.

1.9.2.2 Métricas de acoplamiento

El acoplamiento de módulo proporciona una indicación de la "conectividad" de un módulo con otros módulos, datos globales y entorno exterior. Dhama ha propuesto una métrica para el acoplamiento del módulo que combina el acoplamiento de flujo de datos y de control: acoplamiento global y acoplamiento de entorno. Las medidas necesarias para calcular el acoplamiento de módulo se definen en términos de cada uno de los tres tipos de acoplamiento apuntados anteriormente.

Para el acoplamiento de flujo de datos y de control:

d_i = número de parámetros de datos de entrada

c_i = número de parámetros de control de entrada

d_o = número de parámetros de datos de salida

c_o = número de parámetros de control de salida

Para el acoplamiento global

g_d = número de variables globales usadas como datos

g_c = número de variables globales usadas como control

Para el acoplamiento de entorno:

w = número de módulos llamados (expansión)

r = número de módulos que llaman al módulo en cuestión (concentración)

Usando estas medidas, se define un indicador de acoplamiento de módulo, m_c de la siguiente manera:

$$m_c = k/M$$

Donde k = 1 es una constante de proporcionalidad.

$$M = d_i + a * c_i + d_o + b * c_o + g_d + c * g_c + w + r$$

Donde:

$$a=b=c=2$$

Usando estas medidas, se define un indicador de acoplamiento de módulo, y cuanto mayor es el valor de este indicador, menor es el acoplamiento del módulo. (Pressman, 2005)

1.9.2.3 Métricas de complejidad

Se pueden calcular una variedad de métricas del software para determinar la complejidad del flujo de control del programa. Muchas de éstas se basan en una representación denominada grafo de flujo, un grafo es una representación compuesta de nodos y enlaces (también denominados filos). Cuando se dirigen los enlaces (aristas), el grafo de flujo es un grafo dirigido.

McCabe identifica un número importante de usos para las métricas de complejidad, donde pueden emplearse para predecir información sobre la fiabilidad y mantenimiento de sistemas software, también se alimentan la información durante el proyecto de software para ayudar a controlar la actividad de diseño, en las pruebas y mantenimiento, proporcionan información sobre los módulos del software para ayudar a resaltar las áreas de inestabilidad. (Pressman, 2005)

“La métrica de McCabe proporciona una medida cuantitativa para probar la dificultad y una indicación de la fiabilidad última. Estudios experimentales indican una fuerte correlación entre la métrica de McCabe y el número de errores que existen en el código fuente, así como el tiempo requerido para encontrar y corregir dichos errores. McCabe también defiende que la complejidad ciclomática puede emplearse para proporcionar una indicación cuantitativa del tamaño máximo del módulo.”

1.9.3 Métricas orientadas a clases

La clase es la unidad principal de cualquier sistema orientado a objeto. Esto dice, que tanto las medidas y métricas para una clase individual, la jerarquía de clases y las colaboraciones de las clases resultarán de gran utilidad al ingeniero que desee estimar la calidad de un diseño.

1.9.3.1 Tamaño de clase (TC)

El tamaño general de una clase se puede determinar siguiendo los planteamientos a continuación:

- El número total de operaciones (tanto operaciones heredadas como operaciones privadas de la instancia) que están encapsuladas dentro de la clase.
- El número de atributos (tanto atributos heredados como atributos privados de la instancia) que están encapsulados en la clase.

Si existen valores grandes de TC éstos estarán demostrando que una clase puede tener demasiada responsabilidad, lo cual reduciría la reutilización de la clase y hará complicada la implementación y la prueba. De forma contraria sucede si los valores TC son de menor valor.

Por otra parte es necesaria una evaluación concreta de las métricas mediante los umbrales. Estas medidas para los parámetros de calidad han sido una polémica a nivel mundial en el diseño de sistemas. Algunos especialistas plantean umbrales para estas métricas según se muestra en la tabla 1.

Tabla 1: Clasificación de las clases.

Clasificación	Valores de los umbrales
Pequeño	<= 20
Medio	>20 <= 30
Grande	>30

Finalmente se calcula los promedios correspondientes a los diferentes valores para tener una estimación general del sistema.

Según RUP los subsistemas de diseño deben de tener un alto grado de cohesión, un bajo acoplamiento, así como las clases que agrupan no deben tener muchas responsabilidades, de esta manera se puede garantizar la fácil reutilización de los mismos. Por esta razón se decide aplicar las métricas de cohesión, métricas de acoplamiento y métricas de tamaño de clases. Así se podrá medir el nivel de conectividad y dependencia del subsistema con otros subsistemas. Cabe señalar que el

diseño del subsistema Reportes será de mediana complejidad debido al aporte MVC que hace el Spring framework para el trabajo con reportes en su integración con JasperReport, temas que serán explicados más adelante.

1.10 Pruebas del software

La prueba del software es un elemento crítico para la garantía de la calidad del software. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar la integración adecuada de los componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.

Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que la prueba debe ocurrir durante todo el ciclo de vida: podemos probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, etc. Lo que conduce al principal beneficio de la prueba: proporcionar feedback mientras hay todavía tiempo y recursos para hacer algo. (Juristo, y otros, 2006)

La prueba es un proceso que se enfoca sobre la lógica interna del software y las funciones externas. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Un buen caso de prueba es aquel que tiene alta probabilidad de mostrar un error no descubierto hasta entonces. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

La prueba no puede asegurar la ausencia de defectos; sólo puede demostrar que existen defectos en el software.

Cualquier proceso de ingeniería puede ser probado de una de dos formas:

- Se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa.

- Se pueden desarrollar pruebas que aseguren que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada.

La primera aproximación se denomina prueba de la caja negra y la segunda prueba de la caja blanca.

1.10.1.1 Pruebas de caja blanca

Permiten examinar la estructura interna del programa. Se diseñan casos de prueba para examinar la lógica del programa (E.T.S.I) . Es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar casos de prueba que garanticen que:

- Se ejercitan todos los caminos independientes de cada módulo.
- Se ejercitan todas las decisiones lógicas.
- Se ejecutan todos los bucles.
- Se ejecutan las estructuras de datos internas

1.10.1.2 Pruebas de caja negra

Las pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa.

Los casos de prueba de la caja negra pretende demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta, y
- La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa. (E.T.S.I)

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.

- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Los casos de prueba deben satisfacer los siguientes criterios:

- Reducir, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales.
- Que digan algo sobre la presencia o ausencia de clases de errores.

1.10.1.3 Pruebas de unidad

La prueba de unidad se centra en el módulo. Usando la descripción del diseño detallado como guía, se prueban los caminos de control importantes con el fin de descubrir errores dentro del ámbito del módulo. La prueba de unidad hace uso intensivo de las técnicas de prueba de **caja blanca**. (E.T.S.I)

Una prueba de unidad examina el comportamiento individual de las unidades de trabajo. En una aplicación Java, una unidad de trabajo individual es a menudo un solo método, aunque no siempre. Una unidad de trabajo es una tarea que no es directamente dependiente del completamiento de otra tarea. (Massol, et al., 2004)

1.10.1.3.1 Herramienta para realizar pruebas de unidad

JUnit es framework escogido por la dirección del proyecto para realizar de unidad pruebas de caja blanca. Fue creado por Erich Gamma y Kent Beck, el mismo permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las **pruebas de regresión**, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación. (Usaola)

En la actualidad las herramientas de desarrollo, como Eclipse, cuentan con plugins que permiten que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se realice de manera automática, facilitando al programador enfocarse en la prueba y el resultado esperado, y dejando a la herramienta la creación de las clases que permiten coordinar las pruebas.

JUnit se integra perfectamente con Spring, este framework por sí mismo no soporta la inyección de dependencia, pero Spring soluciona este problema de forma muy fácil y sencilla. (Vicente)

1.10.1.4 Pruebas de regresión

Las pruebas de regresión son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad. El propósito de estas pruebas es asegurar que:

- Los defectos identificados en la ejecución anterior de la prueba se ha corregido.
- Los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores.

La prueba de regresión puede implicar la re-ejecución de cualquier tipo de prueba. Normalmente, las pruebas de regresión se llevan a cabo durante cada iteración, ejecutando otra vez las pruebas de la iteración anterior. (E.T.S.I)

1.10.1.5 Prueba de integración

El objetivo es coger los módulos probados en la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño.

Hay dos formas de integración:

- Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.
- Integración incremental: El programa se construye y se prueba en pequeños segmentos.

En la prueba de integración el foco de atención es el diseño y la construcción de la arquitectura del software.

Las técnicas que más prevalecen son las de diseño de casos de prueba de **caja negra**, aunque se pueden llevar a cabo unas pocas pruebas de **caja blanca**. (E.T.S.I)

Para la validación de la implementación a nivel de funcionalidades internas se decidió realizar pruebas unitarias mediante el framework JUnit el cual se basa en técnicas de prueba de caja blanca. Para la validación de las funcionalidades externas se utilizaron técnicas de caja negra basadas en casos de prueba.

1.11 Herramientas y tecnologías para crear Reportes

¿Qué es un reporte?

Un reporte o informe es una manera organizada de mostrar información procedente de una fuente de datos. Aunque esta información tradicionalmente ha sido entregada en reportes impresos en papel, en la actualidad, se hace evidente que los usuarios necesiten datos en otros formatos, más fáciles de usar, y que les puedan ofrecer un mayor grado de complejidad, exactitud y detalle.

Los reportes se relacionan, generalmente, directa o indirectamente con consultas a una base de datos. Los datos en la base de datos aunque pueden ser mostrados tal y como están, también pueden ser procesados mediante cálculos o representaciones gráficas para generar nueva información. De esta manera los usuarios pueden hacer un análisis más sofisticado y práctico de la información incluso en el momento de tomar decisiones. La generación de reportes sin dudas ayuda a disponer de la información de una manera más rápida y cuenta con medios más flexibles para distribuirla. A continuación se explican características de librerías que facilitan y posibilitan la generación de reportes, todas ellas escritas en Java y de código abierto, ya que el cliente requiere productos programados en lenguajes libres:

1.11.1 IText

IText es una librería Java gratuita para la generación de documentos PDF de forma dinámica. Fue escrita por Bruno Lowagie, Paulo Soares, y otros; está distribuida bajo la Mozilla Public License con la LGPL como licencia alternativa. (iText home page)

Es utilizada, habitualmente, para generar reportes de formatos planos, en general sin campos complejos, aunque cuenta con herramientas para mejorarlos. A parte del soporte para la generación

de documentos PDF, ofrece una serie de características adicionales, entre ellas, la posibilidad de firma digital de documentos PDF, corrección de colores, entre otras.

A su vez permite acceder al contenido del pdf a bajo nivel de una manera muy sencilla, el trabajo será esencialmente generar el contenido, porque todo lo demás ya nos los provee la biblioteca. También cuenta con un modelo de eventos para el documento, página y tabla. De esta manera podemos generar encabezados y pies de página dinámicos, centralizar los estáticos y realizar otras funciones.

Más recientemente, ha sido extendida a una biblioteca PDF de propósito general, capaz de rellenar formularios, mover páginas de un PDF a otro, entre otras cosas. De manera general la biblioteca está bien hecha y facilita mucho trabajo por su simplicidad, disponibilidad de ejemplos, documentación y funcionalidades. (iText home page)

1.11.2 FOP

Formatting Objects Processor fue desarrollado originalmente por James Tauber quien lo donó posteriormente a la Apache Software Foundation en 1999. Se trata de una aplicación Java basada en XLS Formating Objects (XLS-FO) que maneja diversos tipos de formatos incluso más complejos que librerías como IText.

Los XLS tal y como están definidos actualmente se divide en dos partes:

- **XSLT**: transformación de un documento de entrada XML en algún tipo de documento de salida, ya sea XML, HTML, PDF, etc.
- **Formatting Objects**: se encargan de definir la visualización final del resultado de la transformación.

El único uso que se ha hecho en la actualidad de Formatting Objects, siempre ha estado enfocado a la generación de documentos de calidad destinados a la impresión. Este es el caso de los documentos PDF (Apache FOP home page). Es por eso que si queremos generar un PDF partir de un documento en formato FO, podemos utilizar Apache **FOP**, contando este con un conjunto de utilidades libres de distribución.

Dentro de las ventajas que brinda FOP haciendo uso de FO para la generación de documentos se encuentran:

1. Sencillo manejo de ciertas características de la generación de documentos como:

- Paginación automática.
- Definición de márgenes para el documento.
- Definición de patrones distintos de presentación para cada una de las hojas.
- Control "al milímetro" de la presentación de los elementos dentro del PDF.
- Definición simplificada de cabeceras y pies de página.
- Permite la inserción de diversos elementos gráficos como: imágenes, tablas, etc.

2. Permite definir la presentación de cada elemento del documento en base a atributos muy similares, en la mayoría de los casos, a los atributos de una hoja de estilos CSS.

3. Permite una gran integración con otras APIs del proyecto Apache:

- FOP es una parte constituyente de Cocoon (Framework para la publicación de documentos XML a través de la web).
- FOP permite embeber gráficos SVG, renderizándolos a PNGs e insertando el resultado dentro del propio PDF.

4. Permite acometer varios aspectos en la generación de documentos con una alta calidad de impresión:

- Definición del tamaño físico de la página que se creará (A4, Letter, etc).
- Control sobre propiedades de la página como los márgenes, cabeceras, pies, etc.
- Uso de elementos clásicos dentro del documento como son los párrafos, tablas, etc.
- Posibilidad de dar formato de presentación al texto generado, cambiando el tipo de fuente, su tamaño, su color y demás recursos gráficos. (Apache FOP home page)

1.11.3 JasperReports

JasperReports es una librería de clases 100% Java de código abierto desarrollada por Teodor Danciu que está diseñada para agregar capacidades de reporte a las aplicaciones Java. La misma permite organizar la información obtenida desde una base de datos relacional, mediante conectores JDBC y organizarla en diseños de reportes previamente definidos en un formato XML. Es fácil de integrar y los usuarios pueden customizar y redefinir las facilidades que brinda implementando algunas de sus interfaces. No es una herramienta por sí sola por lo que no se puede instalar. Aún cuando JasperReports fue hecho con el propósito principal de añadir características de generación de reportes a aplicaciones web desarrolladas bajo Java, ésta no tiene ningún tipo de dependencia con las librerías

de Java asociada a las aplicaciones web, por lo que es posible utilizar JasperReports para aplicaciones Java de escritorios. (Danciu, 2004)

Además de los datos en texto, JasperReports permite incluir en los reportes imágenes, gráficos y otros elementos, para que los mismo tengan un aspecto profesional. Algunas de las características que provee JasperReports son las siguientes:

- **Permite una diagramación flexible de los reportes:** Los reportes se pueden dividir en secciones opcionales que son: titulo del reporte, el encabezado de página, una sección para los detalles del reporte, el pie de página y una sección de resumen que aparece al final del reporte.

- **Permite que los desarrolladores le surtan datos en varias formas:** esto es que los desarrolladores pueden pasar datos a los reportes por medio del paso de parámetros. Estos parámetros de reportes pueden ser instancia de cualquier clase de Java.

- **Puede generar sub-reportes:** JasperReports permite la creación de reportes dentro de reportes lo que facilita bastante el diseño porque es posible usar estos sub-reportes en otros reportes.

- **Los reportes son capaces de presentar los datos de manera textual o a través de gráficos:** no sólo son capaces de mostrar los datos que le son pasados sino que pueden generar o calcular con esos datos otros datos de forma dinámica y mostrarlos.

- **Puede generar marcas de agua:** JasperReports permite generar textos o imágenes de fondo para utilizarlo como marcas de agua con el propósito de identificar el reporte o simplemente por motivos de seguridad.

- **Se pueden exportar los reportes a una multitud de formatos:** Los reportes generados con JasperReports pueden ser exportados a una multitud de formatos como PDF, XLS, RTF, HTML, XML, CVS (valores separados por coma) y texto plano.

- **Cuenta con herramientas para el diseño de los reportes:** Se usa comúnmente con **iReports**, una herramienta gráfica de código abierto para la edición de informes que evita todo el trabajo del diseño de los reportes a nivel de código. (Soluciones Informáticas 2009)

1.11.3.1 Proceso de creación de un Reporte con JasperReport

Cuando se trabaja con JasperReports los pasos en el proceso de creación de un reporte son los siguientes:

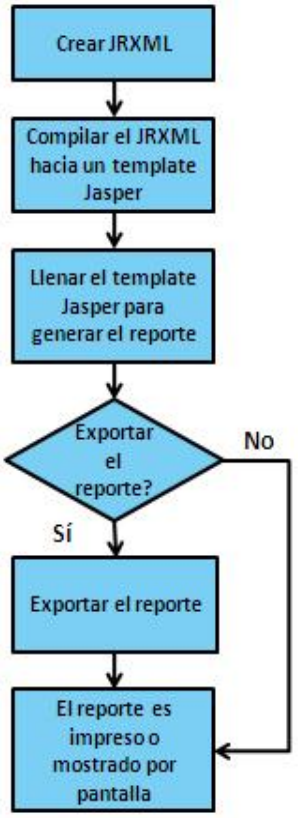


Figura 1.3: Proceso de creación de un reporte con JasperReport.

El primer paso es la creación de un template (plantilla) del reporte mediante el uso de un archivo XML. Estos reportes pueden ser codificados a mano o pueden utilizarse herramientas gráficas que permiten su desarrollo, como por ejemplo **iReport**. Aún cuando estos archivos son archivos xml, a ellos se la da una extensión diferente y específica para los template de JasperReports que es .jrxml. (jasperforge.org)

Luego los archivos .jrxml son compilados a un template binario nativo de JasperReports, ya sea por medio de programación llamando a los métodos apropiados de la librería de clases de JasperReports o mediante el uso de una tarea personalizada ANT. El resultado de la compilación es un archivo conocido comúnmente como archivo Jasper y se guarda al disco con una extensión .jasper. El archivo Jasper es entonces utilizado para generar el reporte proveyéndole los datos que necesita. Este proceso es conocido como "llenar el reporte". El archivo jrxml es compilado una sola vez, pero el archivo Jasper puede ser llenado tantas veces sean necesarias para mostrar o crear el reporte. Los reportes llenos se almacenan en el disco en un formato de archivo nativo de JasperReports conocidos como archivos JasperPrint y que tienen extensión .jrprint. Estos archivos nada más pueden visualizarse en un visor propio de JasperReports conocido como JasperReport Viewer, pero puede exportase a otro múltiples tipos de formatos como por ejemplo pdf. (jasperforge.org)

1.11.3.2 Integración de JasperReports con Spring framework

Para un uso más dinámico y configurable de la librería que implementa JasperReports, dentro de una aplicación basada en la arquitectura que Spring ofrece, han sido implementas un conjunto de clases en este framework que permiten una integración relativamente fácil con dicha librería. De esta manera el proceso de creación de los reportes se realiza de una manera mucho más sencilla.

Con estas implementaciones y su integración a Spring se pueden lograr los siguientes resultados:

- El programador no necesita procesar la gestión y obtención de un recurso (*jrxml, o *.jasper). Spring hace configurable el manejo de estos.
- Reduce el código por parte del programador, así como el tiempo de desarrollo.
- Abstrae al programador del diseño del reporte y evita la dependencia entre el código y el diseño. Aporta un diseño MVC para el trabajo con reportes.
- Cualquier cambio en el diseño o disposición de recursos, no implica un cambio en el código. (C. Walls, 2005)

1.11.3.3 Herramienta para el diseño visual de los reportes con JasperReports.

IReport

La herramienta iReport es un constructor/diseñador de informes, visual, intuitivo y fácil de usar para JasperReports. Este instrumento permite que los usuarios elaboren visualmente informes complejos, ahorrando código en lo que es el tradicional funcionamiento de generación de reportes usando JasperReport. iReport está además integrado con JFreeChart, una de las bibliotecas gráficas de código abierto más difundida para Java. Los datos para imprimir pueden ser recuperados por varias vías, incluso múltiples uniones JDBC, TableModels, JavaBeans, XML, y otros. (jasperforge.org)

Las características más relevantes de iReport son:

- 100% escrito en Java y además de código abierto y gratuito. Maneja el 98% de las etiquetas de JasperReports
- Permite diseñar con sus propias herramientas: rectángulos, líneas, elipses, campos de los textfields, cartas, y subreportes.
- Soporta internacionalización nativamente.
- Navegador de la estructura del documento.
- Recopilador y exportador integrados. Soporta JDBC.
- Soporta JavaBeans como orígenes de datos (éstos deben implementar la interfaz JRDataSource). Incluye Wizards (asistentes) para crear automáticamente informes.
- Tiene asistentes para generar los subreportes.
- Tiene asistentes para las plantillasy posee gran facilidad de instalación.

1.11.4 Selección de la herramienta apropiada para la creación de reportes.

Después de haber hecho el estudio de estas 3 librerías, especializadas en la generación de reportes, se seleccionó JasperReport . Es importante señalar que es la que mejor se integra con el framework Spring y combinada con la herramienta de diseño visual Ireport, constituyen una solución concreta y confiable para facilitar la generación, la previsualización y la impresión de los reportes de una manera más eficiente, cubriendo las necesidades del cliente.

1.12 Conclusiones del capítulo

Conociendo las principales características de las herramientas, técnicas y tecnologías de desarrollo que fueron seleccionadas por la dirección del proyecto para la construcción del sistema, se realizó un estudio del estado del arte de las diferentes herramientas y tecnologías que existen para el diseño y la creación de reportes en una aplicación empresarial en Java, basada en tecnologías de código abierto. De esta manera se seleccionó JasperReport como las más indicada para generar los reportes, apoyandose en Ireport para el diseño visual y en la ventajas que trae como resultado su integración con el framework Spring y el modelo MVC que el mismo aporta. Se estudiaron métricas y patrones de diseño que servirán para tener una mejor visión interna del diseño y ayudará a que el mismo tenga buena calidad. Por último también se estudiaron diferentes pruebas que servirán para probar las funcionalidades tanto internas como externas del subsistema Reportes después de haber sido implementado, de esta manera se podrán corregir los posibles errores detectados, garantizando la eficiencia del mismo y la satisfacción plena de los clientes.

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

2.1. Introducción

El módulo Contratación, tiene como principal objetivo elaborar el contrato de aquellos proyectos que tienen financiamiento y que fueron concebidos para ser desarrollados en el marco del Convenio Integral Cuba-Venezuela.

El mismo permite que de forma automatizada, los Entes Ejecutores, los Ministerios y las Secretarías Técnicas tanto de la parte cubana como venezolana lleguen a un consenso en cada una de las propuestas de Contratos que son situadas en el sistema. Las Secretarías Técnicas son las encargadas de dar la aprobación final a los Contratos que se llevan a cabo entre ambos países, evento que consiste en la firma y aprobación oficial.

El control se manifiesta a partir del preforma del contrato. Al finalizar el proceso se obtiene como resultado el contrato firmado y el Plan de Ejecución Financiera para cada uno de los proyectos incluidos en el contrato.

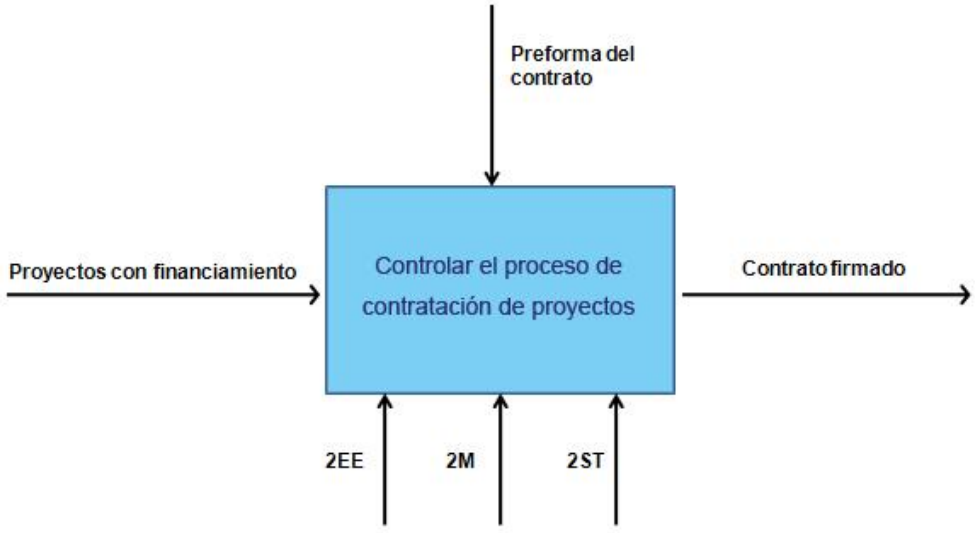


Figura 2.1: Objetivo del módulo de Contratación de proyectos.

La investigación pretende que el módulo cuente con una forma eficiente de obtener reportes que permita la validación y confiabilidad de la información económica extraída en este proceso. Después de haber seleccionado las herramientas y tecnologías más adecuadas para facilitar tal objetivo, se procede a realizar el diseño y la implementación.

2.2. Diseño

RUP define el diseño como un flujo de trabajo en el que sus actividades se realizan desde la fase de Inicio. Durante esta fase se analiza si es posible dar una solución que satisfaga a los requerimientos significativos de la arquitectura. El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida, y a crear un plano del modelo de implementación.

Una entrada esencial en el diseño es el resultado del análisis, o sea el modelo de análisis, que proporciona una comprensión detallada de los requisitos. Además impone una estructura del sistema que debemos esforzarnos por conservar lo más fielmente posible cuando demos forma al sistema. En el **Anexo 1** se muestra representación del Diagrama de Casos de Uso que responde a los seis requisitos funcionales que fueron identificados a partir de las necesidades reales de los usuarios y de las demandas del cliente. En el **Anexo 2** se muestra la especificación de cada uno de ellos. Estas especificaciones son necesarias entre otras cuestiones para darle cumplimiento a la tarea de diseñar las plantillas en iReport (ver **Anexo 7**) guiándose por los prototipos no funcionales descritos en las mismas, prototipos que servirán también para validar si el resultado visual de los reportes es el deseado por los clientes. Estos artefactos son resultado del análisis y sirven de partida para el diseño.

El propósito principal que se plantea la investigación con el diseño de la solución, es crear una entrada apropiada y un punto de partida para las actividades de implementación del subsistema Reportes, capturando las clases y funcionalidades que intervienen.

2.2.1. Subsistema de diseño

Los subsistemas de diseño constituyen una forma de estructurar los artefactos que conforman el modelo de diseño en estructuras más independientes. Los elementos del subsistema deben tener en su conjunto un alto grado de cohesión y bajo acoplamiento con respecto a otros subsistemas, lo cual facilita su reutilización, pudiéndose convertir en componentes genéricos. (Universidad P. Valencia)

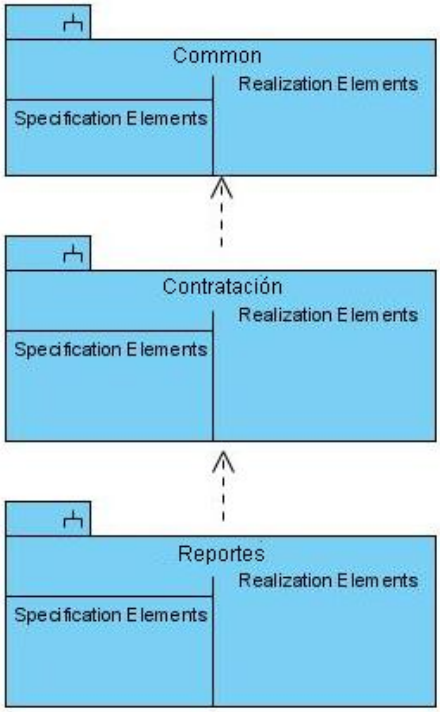


Figura 2.2: Subsistemas de diseño del Módulo Contratación del sistema ICICCV.

Con el fin de separar los aspectos de diseño y lograr una mayor independencia en cuanto a desarrollo y a reutilización de funcionalidades, se identificaron tres subsistemas relacionados con el módulo Contratación del sistema ICICCV.

Dentro de ellos se identifica el **subsistema Reportes**. El mismo es una estructura organizativa que surge con la idea de encapsular en solo paquete, clases y otros elementos interrelacionados, que tendrán la responsabilidad de generar los reportes identificados en los procesos del módulo Contratación. Reportes se nutrirá de las funcionalidades que brinda la capa lógica del negocio del subsistema Contratación. A su vez Contratación se nutrirá de todas las funcionalidades que brinda el subsistema Common y que son generales para todo el sistema.

2.2.2. Paquetes de diseño

El diseño de paquetes facilita la agrupación de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes relacionados de alguna manera. Constituye una división del subsistema en partes más manejables, las cuales no necesariamente tienen una interfaz definida. Con el objetivo de lograr una mejor organización y especialización de los elementos del diseño se identificaron y definieron los siguientes paquetes para el subsistema Reportes del módulo Contratación del sistema ICICCV.

Figura 2.3: Paquetes de diseño del subsistema Reportes del Módulo Contratación.

2.2.3. Diagramas de clases de diseño.

Una clase de diseño es una construcción similar en la implementación del sistema. Los diagramas de clases del diseño son los encargados de representar las relaciones entre clases, interfaces así como la colaboración entre ellos. Constituyen la vista del diseño estático de un sistema. Contienen las definiciones de las entidades de software y además de visualizar, estructurar y documentar los modelos, ayudan a construir el sistema a través de la ingeniería directa e inversa de código.

El sistema ICICCV por sus fines de uso se definió que debía ser una aplicación web, por tal razón en los diagramas de clases de diseño elaborados se hicieron uso de los estereotipos web, con el objetivo de ilustrar la colaboración real y la representación de todos los elementos que interactúan en la ejecución de las funcionalidades que debe brindar el subsistema Reportes.

A continuación se muestra el diagrama de clases de diseño del caso de uso “Generar Reportes de Proyectos Contratados a nivel de Ente ejecutor”, el resto de los diagramas de clases de diseño se pueden consultar en el **Anexo 4**. Es conveniente señalar que el resto de los casos de uso presentan

un diseño similar, esto es debido al modelo vista-controlador que aporta el Spring para el trabajo con los reportes integrado a JasperReport.

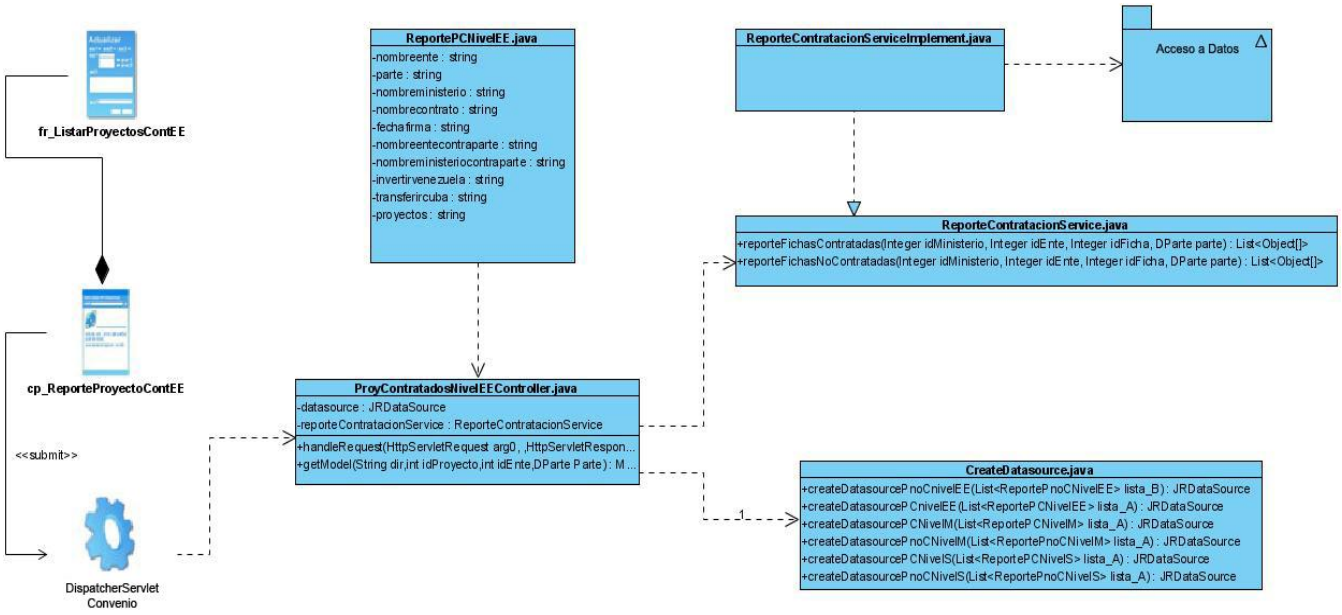


Figura 2.4 : Diagrama de clases de diseño, CU Generar Reportes de Proyectos Contratados a nivel de Ente Ejecutor

El diseño realizado se representa con estereotipos web debido a que el software es una aplicación web. Se representa el DispatcherServlet Convenio como el encargado de manejar todas las peticiones que llegan en forma de url desde la página cliente cp_ReporteProyectoContEE y de darle la vista final al reporte después que las clases controladoras le devuelvan al mismo el ModelAndView. La clase controladora que se representa en el diseño implementa la clase Controller del Spring y contiene los dos métodos representados encargados de construir el ModelAndView. Se representan también en el diseño la clase CreateDatasource encargada de crear los objetos JRDataSource, así como las clases que forman parte de la capa lógica del negocio y de la cual se nutre la clase controladora.

2.2.4. Patrones de diseño empleados

El diseño fue elaborado siguiendo patrones basados en la experiencia, que de manera general constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. En este caso se emplearon los patrones GRASP (General Responsability Assignment

Software Patterns), los que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que se utilizaron son los siguientes:

Experto: En el subsistema se evidencia este patrón con el uso de clases que poseen responsabilidades específicas a cumplir, por ejemplo las clases controladoras poseen la responsabilidad de construir el modelo con los datos que provienen de las capa lógica del negocio, definiendo el nombre de la vista y devolviendo el ModelAndView que necesita posteriormente el framework Spring integrado a JasperReport para generar los reportes. En la figura 2.5, se muestra un fragmento del diagrama de secuencia correspondiente al caso de uso "Generar Reportes de Proyectos Contratados a nivel de Ente Ejecutor", en el mismo se refleja claramente como la clase controladora ProyContratadosNivelEE.java se responsabiliza de implementar todas las funcionalidades necesarias para construir el ModelAndView.

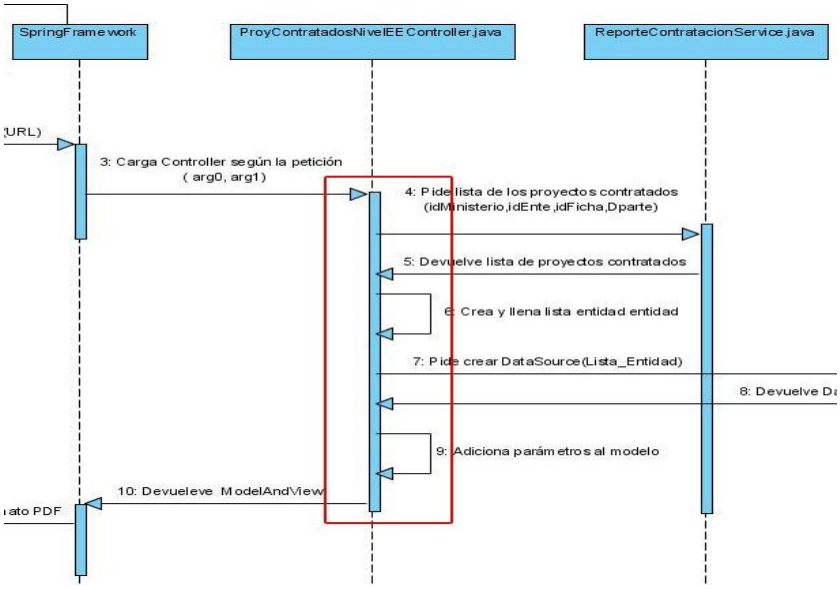


Figura 2.5: Ejemplo de patrón experto

Creador: Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien estas responsabilidades, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización. Ejemplo: En la figura 2.6, se muestra un fragmento del diagrama de secuencia

correspondiente al caso de uso “Generar Reportes de Proyectos Contratados a nivel de Ente Ejecutor”, en el mismo se refleja claramente como la clase CreateDataSource es la responsable de crear el objeto de tipo JRDataSource que se adicionará posteriormente al modelo desde la clase controladora ProyContratadosNivelEEController.java:

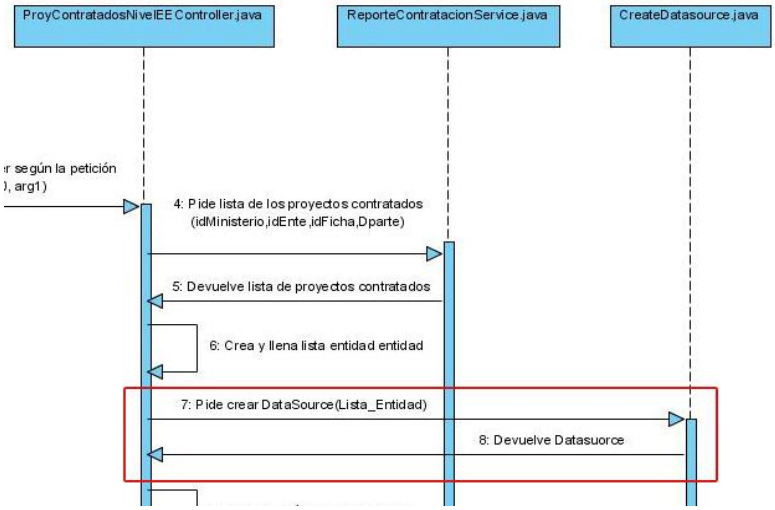


Figura 2. 6: Ejemplo de patrón creador.

De igual forma otro ejemplo que refleja el uso de este patrón es cuando en cada una de las clases controladoras se crea el objeto reporteContrataciónService con el objetivo de acceder a los métodos definidos en la interfaz ReporteContratacionService.java, clase que pertenece a la capa lógica del negocio y es la encargada de devolver todos los datos necesarios que provienen de la capa de acceso a datos y son los necesarios para generar los reportes.

Alta Cohesión: La cohesión es la medida de la fuerza que une a las responsabilidades de una clase. Una clase con baja cohesión es aquella que hace muchas cosas no afines o muchas tareas, lo que trae como consecuencias dificultades para entender, reutilizar y conservarla. Son delicadas y las afectan constantemente los cambios. Una clase con alta cohesión mejora la claridad y la facilidad de su uso, su mantenimiento se simplifica y es fácil de reutilizar. A menudo se genera un bajo acoplamiento.

El subsistema se diseñó asignando responsabilidades de modo que la cohesión sea alta, por lo que los componentes del mismo tienen el mínimo de tareas en el sistema, ejemplo de esto lo representan los métodos de las clases controladoras, ya que tienen bien delimitadas sus responsabilidades, no

recargando en ningún caso sus funcionalidades, permitiendo mayor eficiencia y que el tiempo de respuesta y ejecución no exceda lo estimado, generando por ende **bajo acoplamiento**.

2.2.5. Descripción de las clases

Para darle cumplimiento a los requisitos funcionales asociados a los reportes del módulo Contratación y garantizar que el sistema funcione correctamente, fue necesario diseñar 6 clases controladoras como parte del Modelo Vista-Controlador que implementa el Spring, las cuales se encargan de devolver el modelo con los datos correspondientes a cada reporte y el respectivo nombre de sus vistas. Se implementó una clase controladora para cada reporte. A continuación se muestra la descripción de una de ellas:

Tabla 2.1 - Descripción de la Clase ProyContratadosNivelEEController.

Nombre:	ProyContratadosNivelEEController	
Tipo de clase	Controladora	
Atributo		**Tipo**
datasource reporteContratacionService		JRDataSourcefichaService ReporteContratacionService
Responsabilidades		
Nombre:	**Descripción:**	
getReporteContratacionService()	Método que devuelve el objeto reporteContratacionService	
setReporteContratacionService(ReporteContratacionService reporteContratacionService)	Método que setea el objeto reporteContratacionService	
handleRequest(HttpServletRequest arg0,HttpServletResponse arg1)	Método que devuelve el ModelAndView que requiere la petición. El ModelAndView contiene el modelo con los datos y el nombre de la vista.	
getModel(dir,idProyecto,idEnte,Parte)	Método que construye y devuelve el modelo, adicionándole al mismo la dirección física donde se encuentra la imagen del logotipo del reporte, la fecha actual en el momento de la generación del reporte, el DataSource con los datos de los proyectos contratados a nivel de Ente ejecutor y los parámetros: invertirtotal, transferirtotal y	

	montototal, es decir, toda la información necesaria que será distribuida posteriormente en el diseño de la plantilla del reporte

A su vez se diseñaron 6 clases entidades para organizar de una manera más sencilla los datos que devuelve la capa lógica del negocio, a continuación se representa la descripción de una de ellas:

Tabla 2.2 - Descripción de la Clase ReportePCNivelEE

Nombre:	**ReportePCNivelEE**
Tipo de clase	Entidad
Atributo	**Tipo**
nombreente parte nombreministerio nombrecontrato fechafirma nombreentecontraparte nombreministeriocontraparte invertirvenezuela transferircuba monto proyectos	String String String String String String String String String String String
Responsabilidades	
Nombre:	**Descripción:**
getNombreente() setNombreente(String nombreente) String getNombrecontrato() setNombrecontrato(String nombrecontrato) setNombreentecontraparte(String nombreentecontraparte) getNombreministeriocontraparte() setNombreministeriocontraparte(String nombreministeriocontraparte) getInvertirvenezuela() setInvertirvenezuela(String invertirvenezuela) getTransferircuba() setTransferircuba(String transferircuba) getMonto() setMonto(String monto) getProyectos() setProyectos(String proyectos) getNombreministerio()	Métodos para retornar y setear todos los atributos de la clase.

setNombreministerio(String nombreministerio) String getFechafirma() setFechafirma(String fechafirma) getParte() setParte(String parte)	

Los datos que se mostraran en los reportes provienen en este caso de una base de datos, pero en otras ocasiones pueden provenir por ejemplo de archivos XML u otros ficheros de almacenamiento, estas formas diferentes de obtener datos son las llamadas fuentes de datos Data Source. Las fuentes de datos son representadas en JasperReport por una interfaz llamada JRDataSource, el motor ya provee muchas implementaciones de esta interfaz para poder utilizar datos provenientes de Arreglos, Vectores, Collections, Modelos de Tablas, Consultas SQL, Consultas Hibernate, Archivos CVS, Documentos XML, por nombrar algunos.

Es por eso que fue diseñada una clase llamada CreateDataSource encargada de crear 6 objetos (uno por cada reporte) de tipo JRDataSource que contendrán los datos provenientes de las listas entidades declaradas en las clases controladoras. Estos objetos una vez que tengan distribuida la información, serán adicionados a los modelos que devuelven sus respectivas clases controladoras junto con otros elementos como parámetros y direcciones físicas de imágenes, elementos necesarios para completar toda la información que necesitan los diseños de los reportes previamente definidos en el archivo de relleno .jasper.

Tabla 2.3 - Descripción de la Clase CreateDatasource

Nombre:	**CreateDatasource**	
Tipo de clase	Control	
Atributo		**Tipo**
Responsabilidades		
Nombre:	**Descripción:**	
createDatasourcePnoCnivelEE(List<ReportePnoCNivelEE> lista_B) **createDatasourcePCnivelEE**(List<ReportePCNivelEE> lista_A)	Métodos encargados de organizar la información que viene en las listas entidades de manera tal que el resultado final de la generación de los reportes coincida con los prototipos no funcionales del diseño de los reportes.	

createDatasourcePCNivelM(List<ReportePCNivelM> lista_A) **createDatasourcePnoCNivelM**(List<ReportePnoCNivelM> lista_A) **createDatasourcePCNivelS**(List<ReportePCNivelS> lista_A) **createDatasourcePnoCNivelS**(List<ReportePnoCNivelS> lista_A)	

Se diseñaron también 4 clases que heredan de la clase AbstractCollectionDataSource, clase que define una implementación abstracta de JRDatasource con el propósito de contener los elementos con los que se implementará la estrategia de iteración sobre las estructuras de datos en el momento de rellenar los campos definidos en el diseño.

Tabla 2.4 - Descripción de la Clase ContratoDataSource

Nombre:	**ContratoDataSource**	
Tipo de clase	Clase Auxiliar	
Atributo		**Tipo**
ministerio		ReportePCNivelM
Responsabilidades		
Nombre:	**Descripción:**	
next()	Este método retorna true si quedan más elementos por iterar, y false en caso contrario.	
getFieldValue(JRField field)	En este método se define el modo en que serán obtenidos los campos necesarios, según la estrategia de iteración y de obtención de los mismos, sobre los elementos en cuestión.	

En el **Anexo 3** se describen el resto de las clases utilizadas.

2.2.6. Representación de los diagramas de interacción del diseño (Diagramas de Secuencia)

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva a modelar instancias concretas de clases, componentes y nodos, todo en el contexto de un escenario que ilustra un comportamiento. En cada caso de uso se puede definir diferentes escenarios que posibilitan la identificación de objetos, clases e interacciones entre objetos necesarios para llevar a cabo la parte de funcionalidad que especifica el caso de uso. Los escenarios documentan el reparto de las responsabilidades que se especifican en el caso de uso. (Avilas)

Dentro de diagramas de interacción se encuentran los diagramas de secuencia y los de colaboración, que se utilizan para describir los eventos del sistema. En este caso se utilizaron los diagramas de secuencia por ser uno de los más efectivos para modelar la interacción entre los objetos del sistema a través del tiempo, además, se destaca la ordenación temporal de los mensajes pasados entre ellos.

RUP plantea que se debe modelar un diagrama de secuencia por cada escenario de caso de uso, en este caso en particular, los casos de uso que se identificaron contienen un solo escenario, por lo que se modelaron seis diagramas de secuencia. A continuación se representa el correspondiente al caso de uso “Generar Reportes de Proyectos Contratados a nivel de Ente Ejecutor”, el resto se pueden consultar en el **Anexo 5** del documento. Es conveniente señalar que el resto de los diagramas presentan una secuencia similar, esto es debido al modelo vista-controlador que aporta el Spring para el trabajo con los reportes integrado a JasperReport.

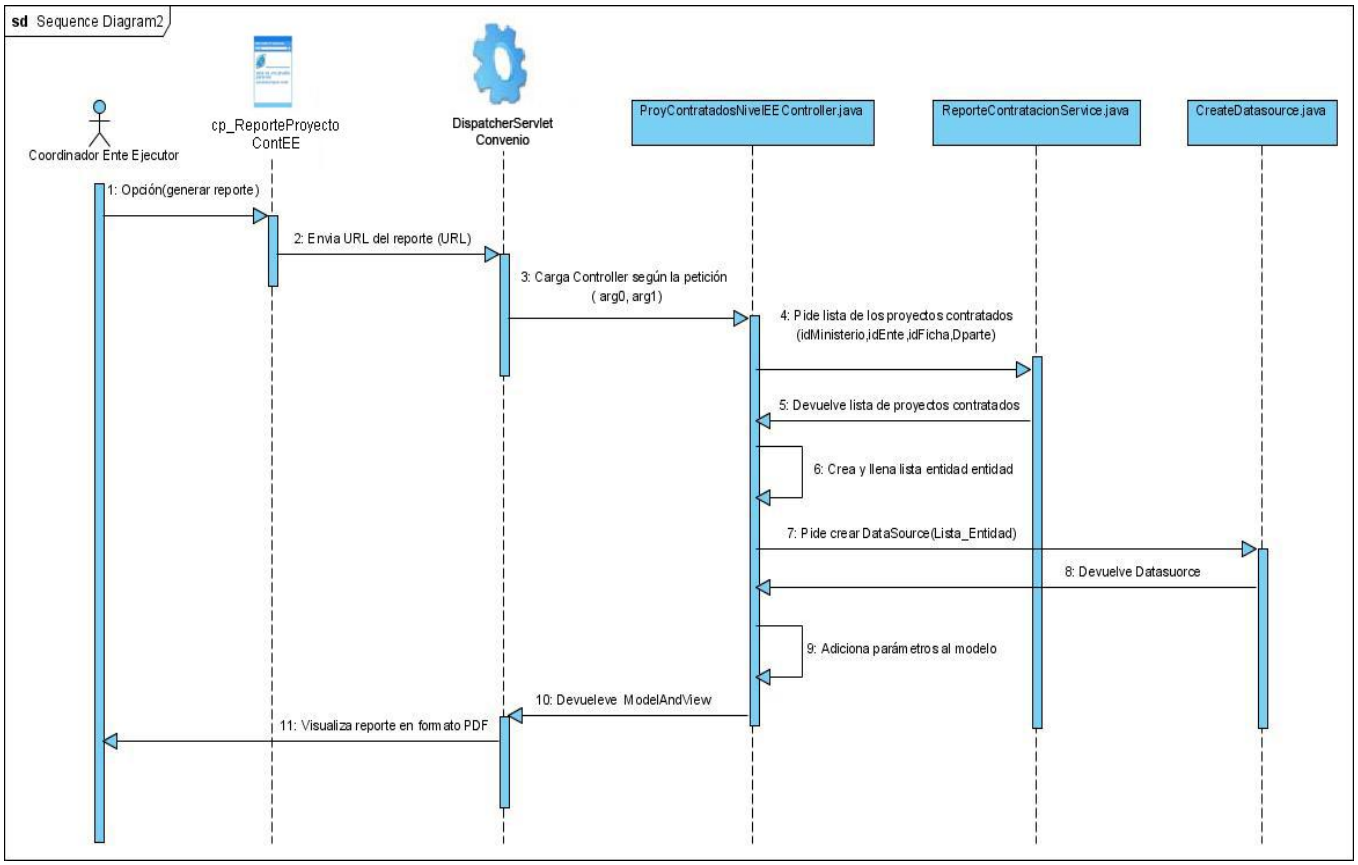


Figura 2.7: Diagrama de secuencia del CU Generar Reportes de Proyectos Contratados a nivel Ente Ejecutor.

2.3. Implementación

El modelo de implementación, principal artefacto que se genera en este flujo, según RUP, describe como los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, ficheros de código fuente, ejecutables etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración en el entorno de implementación y del lenguaje de programación utilizado. Además describe como dependen los componentes unos de otros.

2.3.1. Representación gráfica del diagrama de componentes.

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos. Los componentes físicos incluyen archivos, cabeceras, librerías

compartidas, módulos, ejecutables, o paquetes. Los diagramas de componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. (Hommel, 1999)

Debido a que estos son más parecidos a los diagramas de casos de usos, son utilizados para modelar la vista estática de un sistema, que representa la organización y dependencia que habría entre los componentes físicos que se necesitan para ejecutar la aplicación. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. (Kruchten, 200)

A continuación se representa el diagrama de componentes referente al subsistema Reportes, en el mismo se representa además de las clases identificadas en el diseño, los archivos de extensión .jasper, que son los utilizados para generar los reportes, proveyéndoles de los datos que necesitan . Se representa además un paquete que agrupa las librerías necesarias para garantizar la implementación del subsistema y el correcto funcionamiento de JasperReport. Es necesario señalar que las siguientes librerías junto con las JasperReport deben incluirse en el proyecto en que se desee usar esta herramienta para generar reportes:

- Jasperreports-3.0.0.jar
- Jasperreports-extensions-1.3.1.jar
- Itext-1.02b.jar
- Commons-collections.jar
- Commons-beanutils-1.7.jar
- Jasperreports-extensions-1.3.1.jar
- Jfreechart-1.0.3.jar
- poi-3.0-Final.jar
- Commons-digester.jar
- Commons-logging.jar

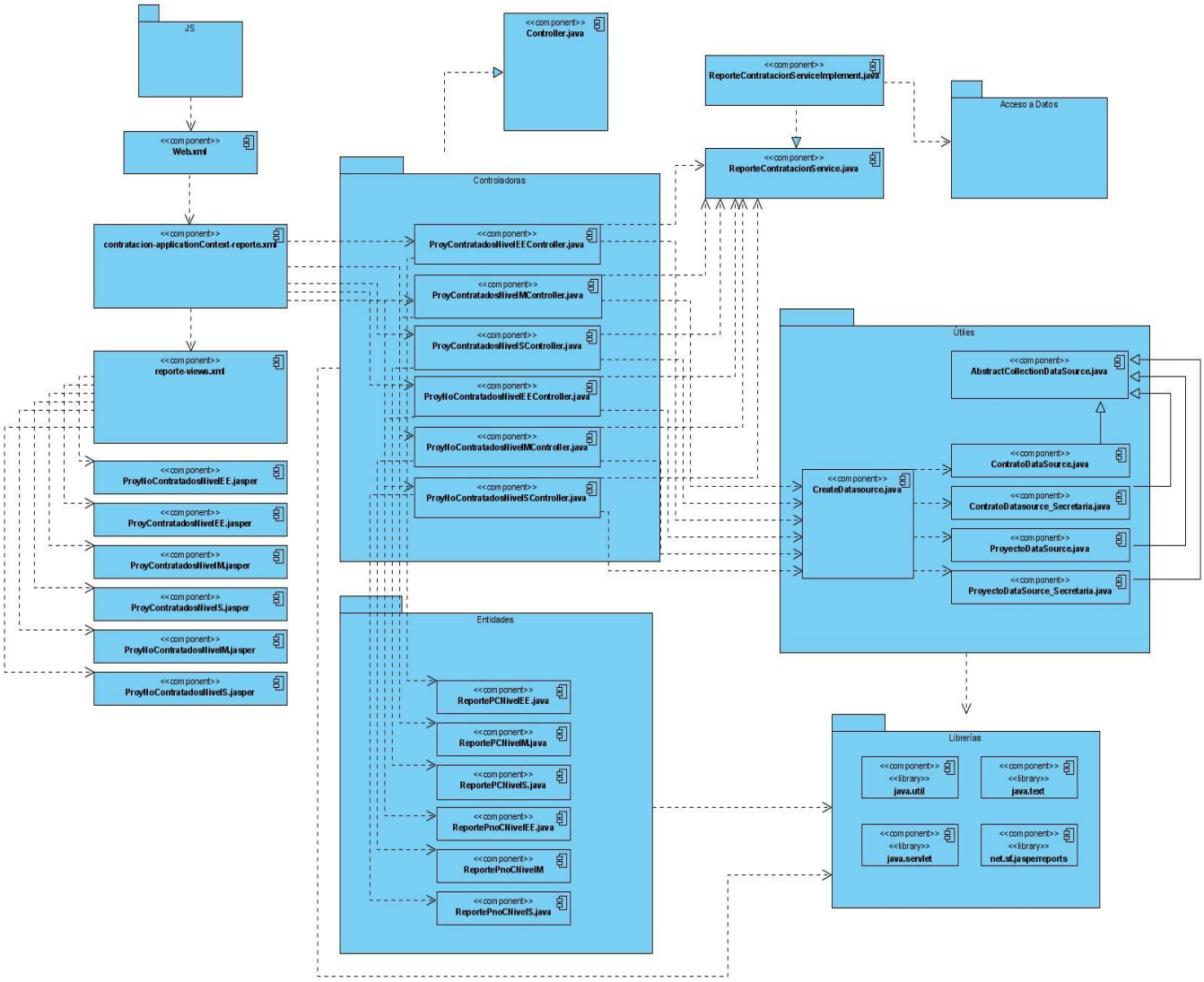


Figura 2.8: Diagrama de componentes del subsistema Reportes.

2.3.2. Estructura para el Desarrollo del Sistema

La arquitectura de software tiene que ver con el diseño y la implementación de estructuras de software de alto nivel, la cual establece los fundamentos para que los analistas, diseñadores y programadores trabajen en una línea común que permita alcanzar los objetivos del sistema, cubriendo todas las necesidades. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales como la confiabilidad, escalabilidad, portabilidad, y disponibilidad.

2.3.3. Convenciones de Archivos y Paquetes

El proyecto Web que se creará en el Eclipse para el desarrollo del sistema será "CCV", al igual que la URL para acceder al mismo. Todas las clases, ficheros XML, páginas JSP y otros ficheros del sistema estarán agrupados en una estructura de carpetas y paquetes. Las clases y ficheros de recursos se encontrarán dentro de la carpeta **src**, haciendo uso de las convenciones de nombrado de paquetes y adaptándolas al sistema quedaría como paquete principal: ***cu.uci.ccv.***

A partir de este nivel se agregaría la separación por subsistemas o módulos:

cu.uci.ccv.administracion (Subsistema Configuración y Administración), ***cu.uci.ccv.common*** (Subsistema que contendría las clases y componentes comunes para el resto de los subsistemas), ***cu.uci.ccv.financiamiento*** (Subsistema Financiamiento de Proyectos), ***cu.uci.ccv.misiones*** (Subsistema Misiones),***cu.uci.ccv.presentacion*** (Subsistema Presentación de Proyectos), ***cu.uci.ccv.seguimiento*** (Subsistema Seguimiento de Proyectos), ***cu.uci.ccv.contratacion*** (Subsistema Contratación de Proyectos), ***cu.uci.ccv.reporte*** (Subsistema Reportes para Contratación)

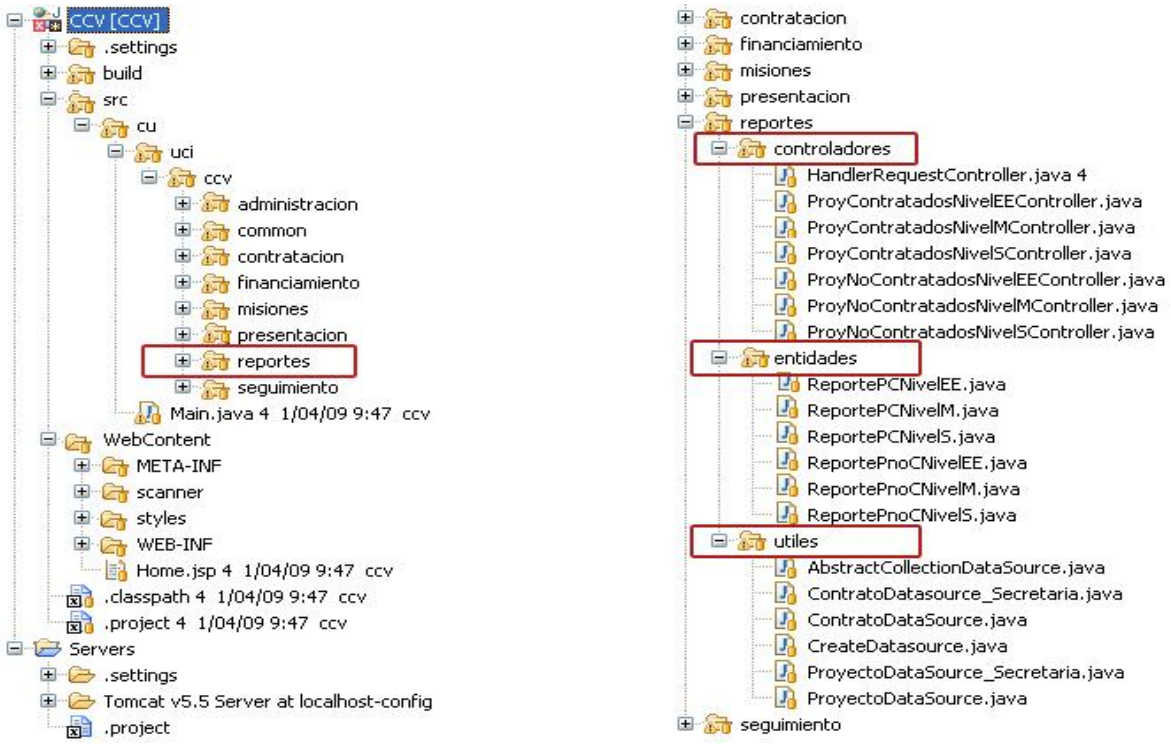


Figura 2.9: Organización de paquetes y subsistemas (subsistema Reportes).

Dentro de cada subsistema se agregó la separación de las clases por capas, en el caso del subsistema Reportes solo se agregaron los tres paquetes identificados en el diagrama de paquetes del diseño:

cu.uci.ccv.reportes.controladores (Controladoras de la capa Presentación, donde se exponen cada uno de los métodos de esta capa)

cu.uci.ccv.reportes.entidades (Clases entidades)

cu.uci.ccv.reportes.utiles (Clases útiles)

2.3.3.1. Ubicación del los ficheros de configuración del Spring

Los ficheros de configuración de Spring que se integrarán a JasperReports serán situados en la carpeta *WEB-INF* dentro de *WebContent*, en este nivel se organizarán en la carpeta reportes dentro de la carpeta *contratación* todos los ficheros .jasper que necesita Spring conjuntamente integrado con JasperReports, para distribuir la información solicitada y darle la vista final al reporte que será generado.

A su vez en la subcarpeta *contratación* que se encuentra en *conf*, se encuentra el fichero **contratación-aplicationContex-reportes.xml**, en el mismo se configuran los beans para atender los diferentes request y direccionarlos a los Controller en dependencia de la URL.

Otro fichero de configuración de Spring que se integra con JasperReports es **reporte-views.xml** donde se configura la vista final del reporte.

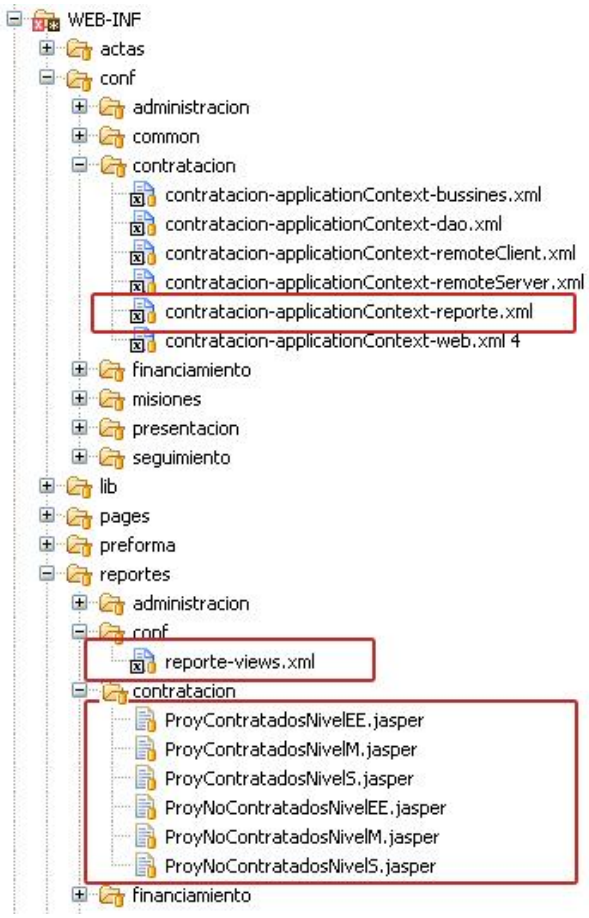


Figura 2.10 : Organización de paquetes (Integración de Spring con JasperReport).

2.3.4. Estándar de Codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico [15]. El estándar de codificación es muy importante para los programadores por muchas razones, pues el producto final (software) no es mantenido por ellos toda la vida del software, el estándar de codificación permite mejorar la lectura del software, permitiendo que el equipo de trabajo que se haga cargo del mantenimiento y soporte al producto entender el código nuevo mucho más rápidamente y más a fondo. El código fuente le debe resultar un entorno familiar para cada miembro del equipo, como si hubiese sido escrito por el mismo. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas y su mantenimiento.

Para la construcción del software, se siguió el estándar de codificación definido para el lenguaje de programación Java. (Hommel, 1999)

Nombres de ficheros

Los nombres de las clases deben terminar en Controller para el caso de las clases controladoras que implementan la interfaz Controller del Spring. En el caso de las clases entidades el nombre empezará por la palabra Reporte y terminará con la abreviatura del nivel correspondiente, nivel Ente Ejecutor (EE), nivel ministerio (M), nivel Secretaria Técnica (S). En el caso de las clases responsables de definir el criterio de iteración y como serán llenados los campos definidos en las plantillas de los reportes, empezarán para los proyectos contratados con la palabra ContratoDataSource y para los no contratados ProyectoDataSource.

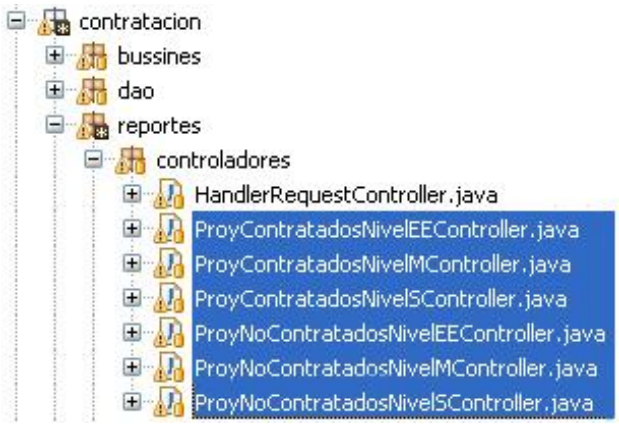


Figura 2.11: Declaración de las clases controladoras

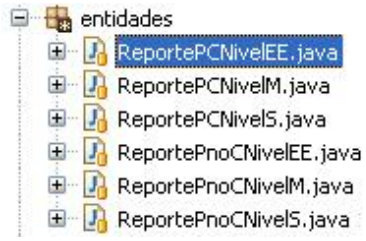


Figura 2.12: Declaración de las clases entidades

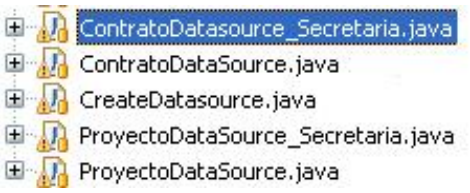


Figura 2.13: Declaración de las clases de apoyo.

Organización de los ficheros

Los ficheros se agruparan por paquetes, cada paquete se corresponderá con cada una de las capas a implementar. La capa de presentación estará agrupada en la capeta web, en el caso de la capa de lógica de negocio, las interfaces estarán agrupadas en la carpeta bussines y las clases que implementan estas interfaces estarán en la subcarpeta impl, las clases de apoyo serán ubicadas en la carpeta util.

Indentación

Para el salto de líneas de una condición se seguirá la regla de los ocho espacios, en los demás casos se seguirá la regla convencional de los cuatros espacios.

Comentarios

En nuestro código se utilizaran los comentarios de implementación /*…*/ y //, el uso de estos comentarios son con el objetivo de dar más información acerca del código. Se evitando realizar comentarios triviales, que pueden a llevar a un mal entendimiento del código, así como realizar dibujos con el uso de los asteriscos.

Los comentarios en bloque deberán realizarse de la siguiente forma:

```
/*
 * bloque de código
 */
```

Los comentarios de una línea se realizarán de la siguiente manera:

```
/* Línea de código */
```

Los comentarios de remolque se realizarán siguiendo la siguiente regla:

```
if (condición) {
    return a; /* caso especial */
}
else {
        return b; /* caso general */
       }
```

Declaraciones

Las funcionalidades deben comenzar con letra inicial minúscula, donde la primera palabra debe ser alusiva a la función que se va a realizar y la segunda palabra debe comenzar con mayúscula y debe ser alusiva al tipo de proceso que se va a realizar, ejemplo:

createDatasourcePCNivelM(List<ReportePCNivelM> lista_A). No se deberá dejar ningún espacio en blanco entre la declaración de los métodos y los paréntesis, la llave de apertura debe encontrarse al final línea de declaración del método, la llave de cierre comienza en una nueva línea indentada excepto en el caso que no exista sentencias entre ambas, donde deberá aparecer a continuación de la llave de apertura. Solamente se podrá realizar una sola declaración de los atributos por líneas. Estas son las principales reglas a seguir para lograr un código de mayor calidad y que este pueda resultarle de fácil entendimiento para las personas que nos sustituyan en la labor de mantenimiento y soporte al software desplegado.

2.3.5. Representación del código fuente de los principales componentes.

A continuación se presenta el código fuente de las principales clases que fueron implementadas y de los archivos de configuración del Spring, incluyendo su integración con JasperReport.

2.2.5.1. Representación del código de clases diseñadas.

En el **Anexo 8** se representan fragmentos del código fuente de las principales clases descritas en el diseño, específicamente las correspondientes al CU Generar Reportes de Proyectos Contratados a

nivel de Ministerio. La implementación se realizó cumpliendo los patrones, la arquitectura y los estándares de codificación definidos previamente en este capítulo.

2.3.5.2. Representación del código de los archivos de configuración del Spring.

A continuación se muestran imágenes de cómo se configuraron los archivos del framework Spring para garantizar una manera sencilla de generar los reportes utilizando JasperReports:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>Convenio</display-name>
<servlet>
    <servlet-name>Convenio</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <!--  <init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>Convenio</servlet-name>
        <url-pattern>/reportes/*</url-pattern>
    </servlet-mapping>
```

Figura 2.14: Archivo de configuración del Spring (web.xml (parte 1))

Spring está diseñado alrededor de un servlet central (DispatcherServlet) que deriva los requests a los diferentes handlers. El DispatcherServlet define un modelo único componente que se encarga de la aplicación de procesamiento de solicitudes. Una interfaz de controlador centraliza funciones tales como ver la selección, la seguridad, y de plantillas, y se aplica de manera coherente en todas las páginas o puntos de vista. El dispatcherServlet, al ser un servlet, debe ser declarado en el web.xml de la aplicación para determinar los request que pretendemos que maneje, en este caso la dirección del proyecto ICICCV le identificó con el nombre Convenio como se muestra en la figura anterior.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        <!--   Cuba    -->
        <!--   -->
        <!-- Venezuela -->

        /WEB-INF/conf/contratacion/contratacion-applicationContext-dao.xml
        /WEB-INF/conf/contratacion/contratacion-applicationContext-bussines.xml
        /WEB-INF/conf/contratacion/contratacion-applicationContext-reporte.xml
        /WEB-INF/conf/contratacion/contratacion-applicationContext-remoteServer.xml


        /WEB-INF/conf/common/common-applicationContext-properties.xml
        /WEB-INF/conf/common/common-applicationContext-dataSource.xml
        /WEB-INF/conf/common/common-applicationContext-dao.xml
        /WEB-INF/conf/common/common-applicationContext-security.xml
        /WEB-INF/conf/common/common-applicationContext-bussines.xml
        /WEB-INF/conf/common/common-applicationContext-captcha.xml
```

Figura 2.15: Archivo de configuración del Spring (web.xml (parte 2))

Después que se tiene configurado el acceso al servlet central, se debe configurar los beans para atender los diferentes request. Spring, al inicializar el servlet DispatcherServlet, buscara un archivo con el nombre [servlet-name]-servlet.xml para localizar la información de configuración de nuestros beans. El nombre del archivo de configuración de los beans puede ser cambiado mediante el parámetro de inicialización del servlet, contextConfigLocation, indicando la ubicación del mismo. En este caso el archivo que contiene los beans de los reportes lleva como nombre contratación-applicationContext-reporte.xml como se indica en la imagen anterior.

```
    http://www.springframework.org/schema/osgi http://www.springframework.org/schema/osgi/spring-osgi.xsd
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.1.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.0.xsd">

<bean id="publicUrlMappingContratacion" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="pcnivelee">ProyContratadosNivelEEController</prop>
            <prop key="pnocnivelee">ProyNoContratadosNivelEEController</prop>
            <prop key="pcnivels">ProyContratadosNivelSController</prop>
            <prop key="pcnivelm">ProyContratadosNivelMController</prop>
            <prop key="pnocnivelm">ProyNoContratadosNivelMController</prop>
            <prop key="pnocnivels">ProyNoContratadosNivelSController</prop>
        </props>
    </property>
</bean>

<bean id="ProyContratadosNivelEEController" class="cu.uci.ccv.reportes.controladores.
         ProyContratadosNivelEEController">
    <property name="reporteContratacionService" ref="ReporteContratacionService"></property>
</bean>
```

Figura 2.16 : Archivo de configuración del Spring (contratación-applicationContext-reporte.xml)

El DispatcherServlet tiene un conjunto de beans especiales que utiliza para poder procesar los request y resolver la vista apropiada a mostrar. Estos beans están definidos por Spring y pueden ser configurados como cualquier otro bean en el WebApplicationContext.

- **Controllers:** Son los encargados de procesar el request.
- **Handler mapping**: Determinan una lista de pre y pos procesos y controllers que deberán ser ejecutados para determinado request
- **View resolvers:** Permiten asignar un nombre lógico a las vistas de la aplicación

Controller

Al recibir un request el DispatcherServlet mapea dicho request a una clase que implementa la interfaz Controller y llama al método handleRequest definida en ella. Precisamente estas son las clases Controladoras que se diseñaron en el capítulo anterior como parte del subsistema Reportes y posteriormente fueron implementadas.

Handler Mappings

Para especificar a que Controller llamar se utilizan los Handler Mappings. La funcionalidad básica que proveen los mismos es determinar, según el contenido del request que acciones deben llevarse a cabo. Puede definir una cadena de ejecución a aplicar sobre el request. Este concepto de poder actuar sobre el request antes o después de invocar al Controller permite, por ejemplo, seleccionar al Controller por la URL en el request.

Como siempre Spring provee varias clases que podemos utilizar sin realizar ninguna implementación. Como se aprecia en el fragmento de código de la imagen anterior, SimpleUrlHandlerMapping permite mapear las direcciones URL de los reportes a los beans donde se especifican las ubicaciones de las clases controladoras.

```
 version="1.0" encoding="UTF-8"?>
ns xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:lang="http://www.springframework.org/schema/lang"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:util="http://www.springframework.org/schema/util"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/bear
    http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.1.xsc
    http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-2.0.
    http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.1.xsd
    http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.0.
```

```
<bean id="viewResolverXML"
    class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location">
        <value>/WEB-INF/reportes/conf/reporte-views.xml</value>
    </property>
    <property name="order" value="1" />
</bean>
```

Figura 2.17: Archivo de configuración del Spring (common-applicationContext-web.xml)

View Resolver

Las vistas en Spring son referenciadas por un nombre y localizadas por una clase que implemente la interfaz ViewResolver, es el caso de la clase representada en la imagen anterior.

```
ml version="1.0" encoding="UTF-8"?>
eans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/

- Configuraciones del modulo contratacion -->

 <bean id="pcnivelee" class="org.springframework.web.servlet.view.jasperreports.JasperReportsPdfView">
     <property name="url">
         <value>/WEB-INF/reportes/contratacion/ProyContratadosNivelEE.jasper</value>
     </property>
     <property name="reportDataKey">
         <value>jrdatasource</value>
     </property>
 </bean>

 <bean id="pcnivels" class="org.springframework.web.servlet.view.jasperreports.JasperReportsPdfView">
     <property name="url">
         <value>/WEB-INF/reportes/contratacion/ProyContratadosNivelS.jasper</value>
     </property>
```

Figura 2.18 : Archivo de configuración del Spring (reportes.views.xml)

En este caso Spring se integra con JasperReport para un mejor uso de esta librería generadora de reportes. Se implementaron clases que determinan la vista final de los reportes especificando el formato en que serán generados. En este caso como se aprecia en el fragmento de la imagen anterior, se usa la clase JasperReportsPdfView para generar los reportes en formato PDF y se especifica la ubicación de los archivos *.jasper de cada uno de ellos.

2.4. Conclusiones del capítulo

En este capítulo se realizó el diseño y la implementación del subsistema Reportes. Se describió como fue realizado el diseño a partir de las funcionalidades previstas y las restricciones impuestas (requerimientos), creando de esta manera un punto de partida para las posteriores actividades de implementación. Para ello como punto de partida se hizo referencia a algunos artefactos del flujo de análisis, esto fue necesario para darle cumplimiento a tareas como la de poder diseñar las plantillas de los reportes. Posteriormente se generaron los artefactos que componen el modelo del diseño que propone RUP en el flujo de trabajo de Diseño, gracias a esto se pudo guiar de una manera más organizada el proceso de desarrollo para este flujo. Además se justificó el uso de los patrones de diseño empleados como parte de la propuesta de solución.

Posteriormente se prosiguió a presentar los resultados obtenidos después de implementar todas las funcionalidades identificadas en el diseño. Como parte de esos resultados se representó la vista estática del subsistema Reportes mediante los componentes físicos que se necesitaron para garantizar el buen funcionamiento del mismo. Se hizo un estudio de la estructura arquitectónica propuesta para el sistema ICICCV, adaptando la implementación del subsistema Reportes a esta estructura. De esta manera y aplicando estándares de codificación que propone el lenguaje Java en la implementación del código, se pudo trabajar en una línea común permitiendo alcanzar los objetivos y cubriendo todas las necesidades referentes a los reportes.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

3.1. Introducción

Una vez terminado los flujos de diseño e implementación del subsistema Reportes y para validar la calidad y correspondencia con los requisitos funcionales y no funcionales para los cuales fue desarrollado, se presentan diferentes métodos y procedimientos que se emplearon para lograr este objetivo. Garantizar la calidad de los artefactos obtenidos en cada fase ayuda a minimizar los errores y obtener un producto con mayor calidad, que cumpla con todas las expectativas del cliente. He ahí la importancia del análisis de los resultados obtenidos en cada etapa del proceso de desarrollo de software.

A continuación se muestran los procedimientos y métodos utilizados para medir la factibilidad de los artefactos obtenidos en el diseño a través de métricas de diseño de clases. Además son validadas las funcionalidades que fueron implementadas para el subsistema Reportes, mediante pruebas de caja blanca y pruebas de caja negra.

3.1.1. Métricas de diseño a nivel de componentes

Estas métricas incluyen medidas de la cohesión, acoplamiento y complejidad del subsistema que ayudarán a juzgar la calidad del software.

3.1.1.1. Métrica de cohesión

Una buena práctica para el diseño lo constituye el lograr una alta cohesión, como resultado de aplicar los patrones GRASP en la construcción del mismo. Para medir el índice de cohesión de los elementos que conforman el diseño del subsistema Reportes se aplicó la métrica de Bieman y Ott, la cual se detalla en el **Capítulo 1.**

Para ello se analizaron elementos pertenecientes a los principales conceptos que plantea la métrica, dígase Porción de datos, Símbolos léxicos (tokens) de datos, Señales de unión, Señales de súper-unión y Cohesión. Para proceder al análisis de cada uno de estos conceptos se recogieron los datos que ilustran el nivel de uso de las principales clases diseñadas para el subsistema Reportes (ver **Tabla 3.1**).

Tabla 3.1 Usabilidad de las clases.

	Usabilidad
1.CreateDatasource	4
2. ProyContratadosNivelEEController	3
3. ProyContratadosNivelMController	3
4. ProyContratadosNivelSController	3
5. ProyNoContratadosNivelEEController	3
6. ProyNoContratadosNivelMController	3
7. ProyNoContratadosNivelSController	3
8. ReportePCNivelS	0
9.ReportePCNivelM	0
10. ReportePCNivelEE	0
11. ReportePnoCNivelEE	0
12. ReportePnoCNivelM	0
13. ReportePnoCNivelS	0
14. ContratoDataSource.java	1
15.ContratoDatasource_Secretaria.java	1
16.ProyectoDataSource.java	1
17.ProyectoDataSource_Secretaria.java	1

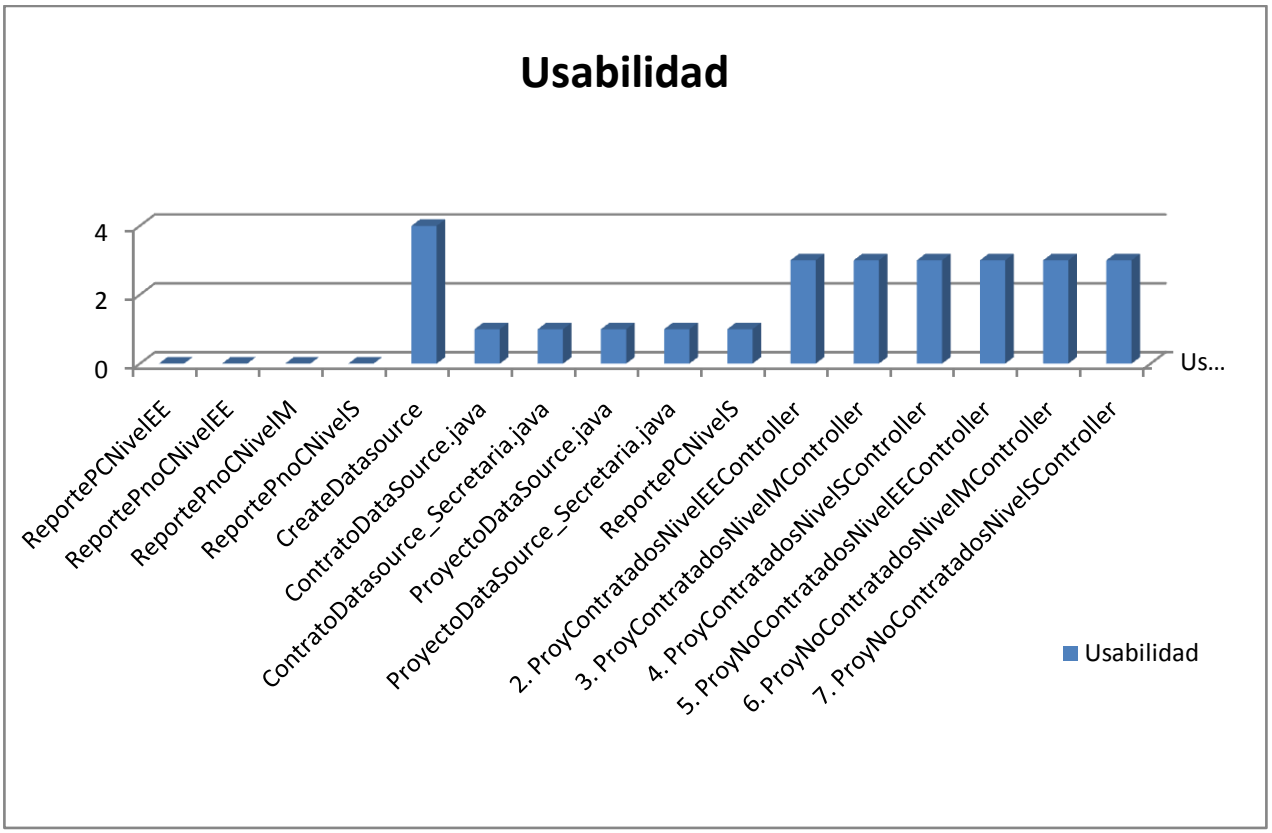


Figura 3.1 Relaciones de uso de las clases.

La cohesión funcional se determina con dos enfoques:

1. Determinando la cohesión funcional fuerte (CFF) y la pegajosidad: se obtienen cuando el resultado de la métrica es de 1. Se define como:

   CFF = número de súper adhesivos (i) / número de elementos (i)

2. Determinando la cohesión funcional débil(CFD): Se define como:

   CFD = número de adhesivos (i)/ número de elementos (i)

Adhesivo. Se le llamará adhesivo a un elemento que aparece en dos o más rebanadas.

Súper adhesivo. Se denomina súper adhesivo a un elemento que está en todos los elementos de un módulo.

(i) Se define como la muestra.

Según los datos de las clases analizadas se tiene que:

número de elementos = 17 **número de súper adhesivos(i) = 0** **número de adhesivos(i) = 11**

CFD = número de adhesivos (i) / número de elementos (i)

CFD = 11 / 17 **CFD =** 0.64

CFF = número de súper adhesivos (i) / número de elementos (i)

CFF = 0 / 17 **CFF =** 0

La métrica de Bieman y Ott plantea que mientras más cerca están los valores de CFF y CFD de 1 mayor será la cohesión del módulo. Los resultados demuestran que no hay una cohesión funcional fuerte, pero la relación del número de clases adhesivas con el número total de elementos de la muestra, determinados por la CFD = 0.64, está cercana a 1, lo que demuestra que el diseño de las clases del subsistema Reportes posee una cohesión funcional con un 64% de fortaleza.

3.1.1.2. Métrica de acoplamiento

Para medir el nivel de conectividad y dependencia del subsistema con otros subsistemas se aplica la medida propuesta por Dhama (ver Capítulo 1), con el fin de determinar el nivel de independencia y reutilización que posee el subsistema según el diseño propuesto.

Definiendo:

Acoplamiento de flujos de datos de control:

d_i = 2 (número de parámetros de datos de entrada)

c_i = 1 (número de parámetros de control de entrada)

d_0= 1 (número de parámetros de datos de salida)

c_0 = 1 (número de parámetros de control de salida)

Acoplamiento global:

g_d = 0 (número de variables globales usadas como datos)

g_c = 0 (número de variables globales usadas como control)

Acoplamiento de entorno:

w = 1 (número de subsistemas llamados **Figura 3.12**)

r = 0 (número de subsistemas que llaman al subsistema Reportes)

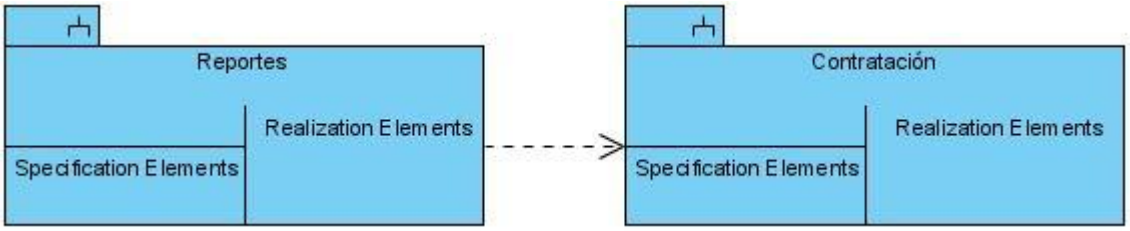


Figura 3.2 Expansión del subsistema Reportes.

Indicador de acoplamiento del subsistema:

m_c = K / M

Donde K = 1, constante de proporcionalidad según el autor

M = d_i + aXc_i + d_0 + bXc_0 + g_d + cXg_c+ w + r

a=b=c=2

m_c = 1 / (2 + 2X1+ 1 + 2X1 + 0 + 2X0+ 1 + 0)

m_c= 0.13

El valor de 0.13 del indicador de acoplamiento del subsistema nos sugiere el grado de acoplamiento del mismo, lo que demuestra que existe dependencia con el subsistema Contratación, pero que la misma es mínima y no repercute por el hecho de depender de módulos básicos para el funcionamiento del sistema ICICCV.

3.1.2. Métricas orientadas a clases. Tamaño de clase (TC)

Con el objetivo de comprobar el adecuado diseño de las clases y el nivel de reutilización de las mismas se aplicó la métrica del TC (**ver Capítulo 1**). Se aplicará esta métrica para las principales clases definidas en el subsistema Reportes.

Tabla 3.2 Atributos y operaciones de las clases.

Clases	N°. de atributos	N°. de operaciones
1.CreateDatasource	0	6
2. ProyContratadosNivelEEController	2	2
3. ProyContratadosNivelMController	2	2
4. ProyContratadosNivelSController	2	2
5. ProyNoContratadosNivelEEController	2	2
6. ProyNoContratadosNivelMController	2	2
7. ProyNoContratadosNivelSController	2	2

Se presentó un **total de 7 clases** para un **promedio de atributos de 1.71** y **un promedio de operaciones de 2.57** De esta forma el umbral queda con los datos mostrados a continuación:

Tabla 3.3 Umbral de tamaño de clase.

Umbral	Tamaño	Cantidad de clases
<=20	Pequeño	7
>20<=30	Medio	0
>30	Grande	0

Por lo que se concluye que las clases son relativamente pequeñas y no tienen grandes responsabilidades, lo cual aumenta la reutilización de estas y no será difícil su implementación y prueba.

3.1.3. Pruebas de unidad

Uno de los pasos importantes que debe tener en cuenta el rol de programador es evaluar o probar los componentes resultantes de implementar los elementos de diseño. Las pruebas surgen con la necesidad de prevenir y detectar los errores en la implementación del código. Además las pruebas ayudan al programador a evitar escribir código incorrecto permitiendo implementar componentes con calidad.

La validación de las funcionalidades que fueron implementadas se realizó a través de diferentes pruebas de unidades, estas pruebas hacen uso intensivo de las técnicas de prueba de **caja blanca**. Las pruebas de unidad experimentan si un método cumple los términos de su API contrato.

Una API contrato es un acuerdo hecho por la interfaz del método. El acuerdo se realiza entre los valores de los objetos entrada del método y los valores de los objetos que retorna el método. De esta forma, si la conducta del método no es la esperada entonces podemos decir que el método rompió el contrato.

Es necesario conocer cómo aplicar las pruebas y que herramientas utilizar debido a que una mala selección puede causar pérdida de tiempo en la realización.

En Java la implementación de las pruebas de unidad puede realizarse utilizando clases de pruebas improvisadas que dado algunos valores de entrada comprueban los valores esperados por una unidad de trabajo. La dificultad de estas clases está dada porque no cumplen con los siguientes aspectos:

- Cada unidad de trabajo debe correr independientemente de todas las otras unidades de trabajo.
- Los errores deben ser detectados y reportados pruebas por pruebas.
- Debe ser fácil de definir cual prueba de unidad correr en el momento que se desee.

Con el objetivo de solucionar estos problemas han sido creadas herramientas muy potentes para desarrollar pruebas de unidad. Entre ellos el framework JUnit (ver capitulo 1), que fue el seleccionado por el equipo de arquitectura para validar internamente la funcionalidad del subsistema Reportes.

A continuación se muestra una gráfica con los resultados de las pruebas de caja blanca después de haber pasado por varias iteraciones. La clase que presentó mayor cantidad de errores debido a su complejidad fue la clase CreateDatasource, aunque la mayoría fueron problemas con el mal tratamiento de excepciones. Los resultados demuestran como la integración de JasperReport con el Spring al reducir la cantidad y complejidad de líneas de código propician una menor cantidad de errores.

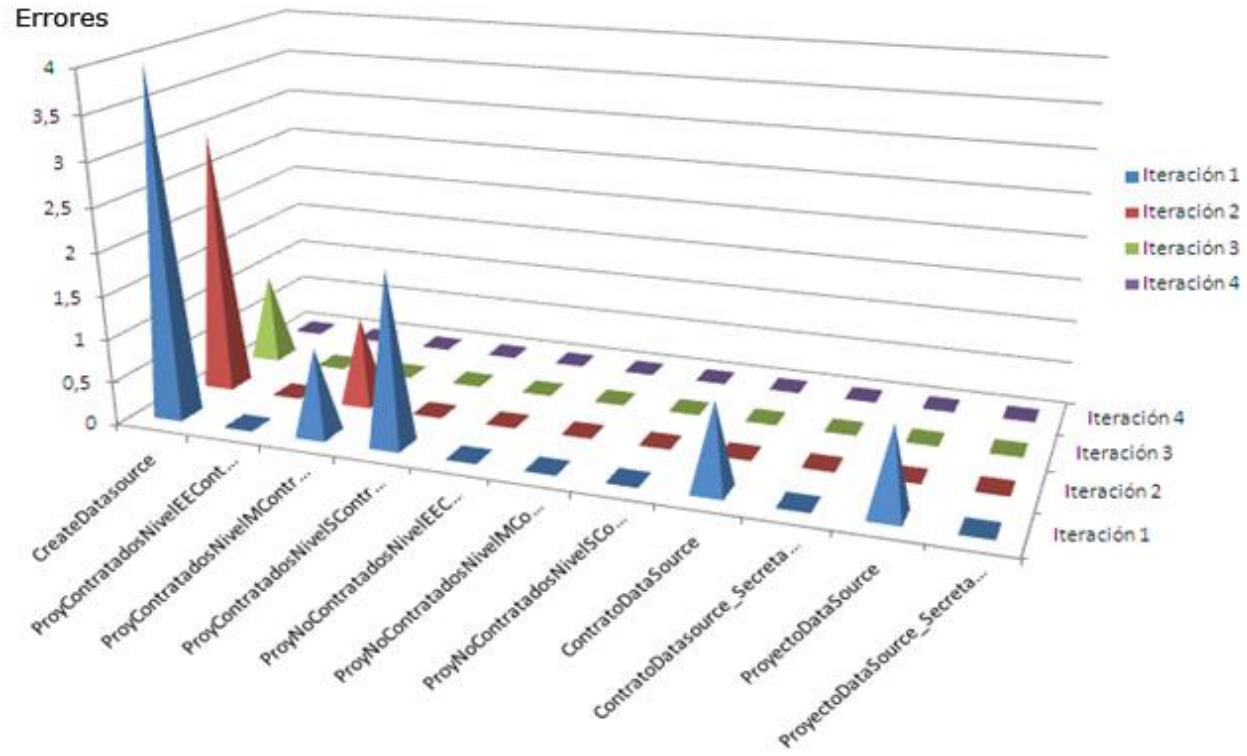


Figura 3.3 Resultado de las iteraciones de pruebas de unidad.

A continuación se muestran 2 imágenes que representa la realización de las pruebas de unidad aplicada a la clase CreateDatasource en la última iteración, el resto de las pruebas aplicadas en las demás clases se encuentran en el **Anexo 5** del documento :

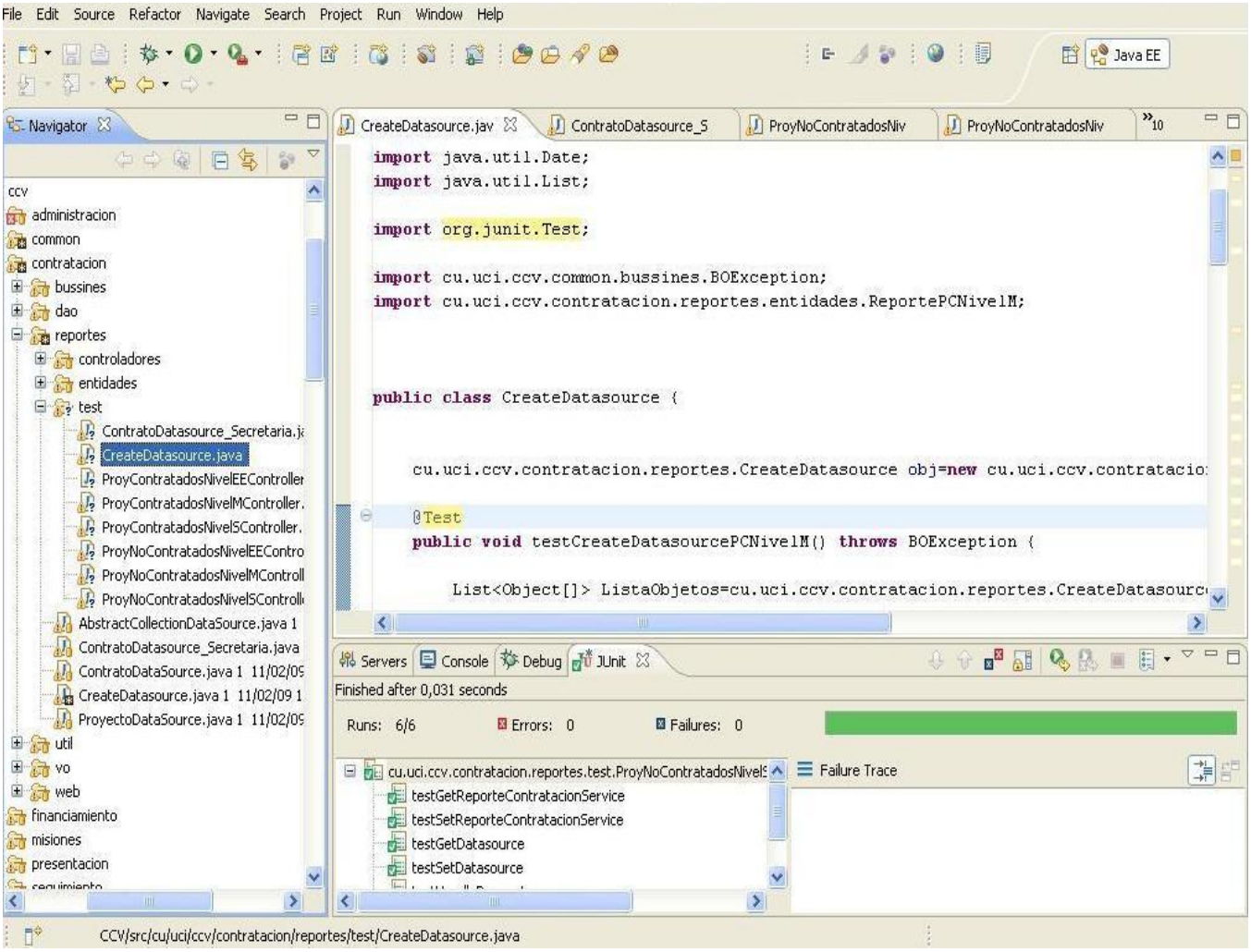


Figura 3.4: Prueba realizada a la clase CreateDatasource (Vista general, JUnit integrado al Eclipse).9

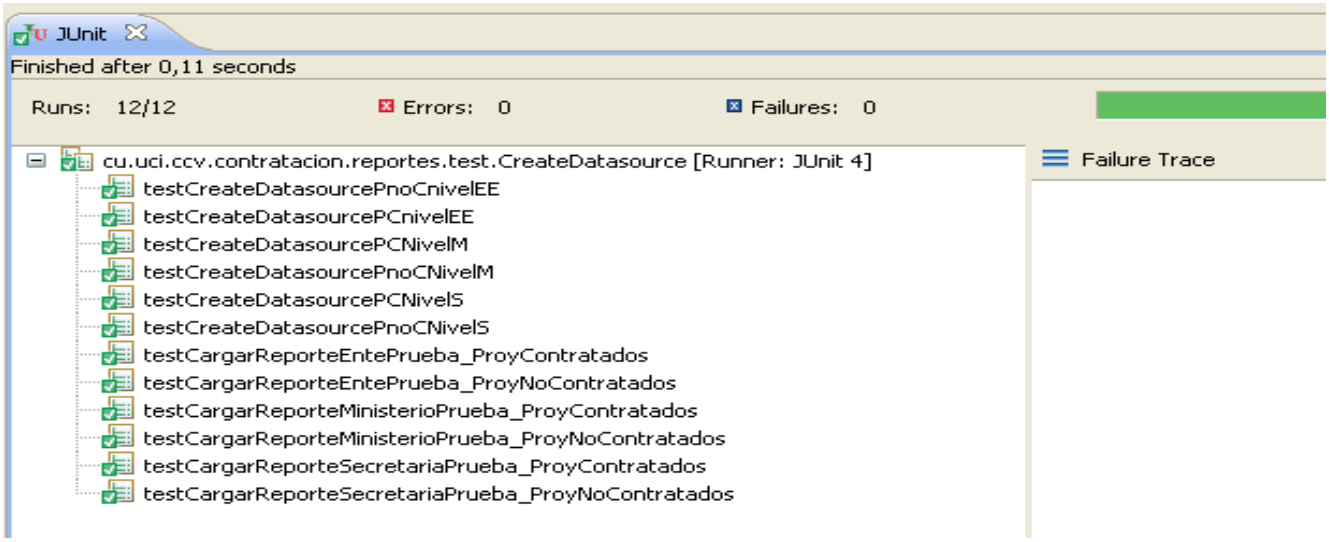


Figura 3.5: Prueba realizada a la clase CreateDatasource (Vista especifica del resultado de los métodos validados).

La figura anterior muestra la prueba realizada a la clase que contiene los métodos responsables de crear los objetos de tipo JRDatasource, cuyo resultado fue satisfactorio.

3.1.4. Pruebas de caja negra

El proceso de pruebas de **caja negra** se llevó a cabo sobre la interfaz del software y se centró principalmente en los requisitos funcionales para verificar el comportamiento del subsistema externamente, es decir, la calidad funcional y la eficiencia de los reportes. De esta forma se pudo detectar si existía un incorrecto o incompleto funcionamiento de los reportes, así como los errores en la vista final de los mismos y su rendimiento.

Para realizar este procedimiento y poder determinar si los requisitos funcionales eran satisfactorios, se realizaron casos de prueba.

Lo que caracteriza un escrito formal de caso de prueba es que hay una *entrada conocida* y una *salida esperada*, los cuales son formulados antes de que se ejecute la prueba. La *entrada conocida* debe probar una precondición y la *salida esperada* debe probar una postcondición. Bajo circunstancias especiales, podría haber la necesidad de ejecutar la prueba, producir resultados, y luego un equipo de expertos evaluaría si los resultados se pueden considerar como "Correctos" [Pressman, 2002]

A continuación se muestran los casos de prueba realizados para los reportes de los proyectos contratados en los distintos niveles que forman parte del flujo informativo. Estos casos de prueba se hicieron uno por cada caso de uso y son el resultado de la última iteración.

3.1.4.1. Diseño de Casos de Prueba

Nombre del Caso: CP Generar reporte de proyectos contratados a nivel de Ente Ejecutor

Descripción General

El caso de uso inicia cuando un coordinador de ente necesita obtener reportes con los datos correspondientes a los proyectos contratados con los que está asociado.

Condiciones de Ejecución:

El sistema debe estar instalado y ejecutado correctamente.

El usuario debe estar autenticado con los permisos necesarios.

Tabla 3.4 - Secciones a probar en el Caso de Uso

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC1: Reportar proyectos contratados a nivel de EE.	• EC 1.1: El actor selecciona la opción de generar reporte de los proyectos contratados.	El sistema muestra la pantalla un reporte de los datos correspondientes al proyecto contratado seleccionado. El sistema ejecuta la acción de impresión de la información del reporte, dando el formato requerido para una buena impresión y termina el caso de uso.	El actor solicita la búsqueda de los proyectos a través del siguiente filtro: Nombre del proyecto Y presiona el botón "Generar"

Tabla 3.5 - SC 1: Reportar proyectos contratados nivel EE.

Id del escenario	Escenario	Nombre del proyecto	Respuesta del Sistema	Resultado de la Prueba
EC 1	El actor selecciona la opción de generar reporte de los proyectos contratados.	Se selecciona un proyecto Se seleccionan todos los proyectos	El sistema genera un reporte en formato PDF	Tiene un resultado satisfactorio
		Se seleccionan todos los proyectos	El sistema muestra un mensaje si no hay proyectos contratados.	

Nombre del Caso: CP Generar reporte de proyectos contratados a nivel de Ministerio

Descripción General

El caso de uso inicia cuando un coordinador de ministerio necesita obtener reportes con los datos correspondientes a los proyectos contratados con los que está asociado.

Condiciones de Ejecución:

El sistema debe estar instalado y ejecutado correctamente.

El usuario debe estar autenticado con los permisos necesarios.

Tabla 3.6 - Secciones a probar en el Caso de Uso

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC1: Reportar proyectos contratados a nivel de EE.	• EC 1.1: El actor selecciona la opción de generar reporte de los proyectos contratados a nivel de ministerio.	El sistema muestra la pantalla un reporte de los datos correspondientes al proyecto contratado seleccionado. El sistema ejecuta la acción de impresión de la información del reporte, dando el formato requerido para una buena impresión y termina el caso de uso.	El actor solicita la búsqueda de los proyectos a través del siguiente filtro: Nombre del proyecto Nombre del ente al que pertenece Y presiona el botón "Generar"

Tabla 3.7 - SC 1: Reportar proyectos contratados nivel Ministerio.

Id del escenario	Escenario	Nombre del proyecto	Nombre del ente	Respuesta del Sistema	Resultado de la Prueba
EC 1	El actor selecciona la opción de generar reporte de los proyectos contratados.	Se selecciona todos los proyectos de ese ente Se selecciona un proyecto Se selecciona todos los proyectos de ese ente	Se selecciona un ente Se selecciona un ente Se selecciona todos los entes	El sistema genera un reporte en formato PDF	Tiene un resultado satisfactorio
		Se selecciona todos los proyectos de ese ente Se selecciona todos los proyectos de ese ente	Se selecciona todos los entes Se selecciona un ente	El sistema muestra un mensaje si no hay proyectos contratados.	

Nombre del Caso: CP Generar reporte de proyectos contratados a nivel de Secretaria

Descripción General

El caso de uso inicia cuando un coordinador de secretaria necesita obtener reportes con los datos correspondientes a los proyectos contratados con los que está asociado.

Condiciones de Ejecución:

El sistema debe estar instalado y ejecutado correctamente.

El usuario debe estar autenticado con los permisos necesarios.

Tabla 3.8 - Secciones a probar en el Caso de Uso

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC1: Reportar proyectos contratados a nivel de EE.	• EC 1.1: El actor selecciona la opción de generar reporte de los proyectos contratados.	El sistema muestra la pantalla un reporte de los datos correspondientes al proyecto contratado seleccionado. El sistema ejecuta la acción de impresión de la información del reporte, dando el formato requerido para una buena impresión y termina el caso de uso.	El actor solicita la búsqueda de los proyectos a través del siguiente filtro: Nombre del proyecto Nombre del ente al que pertenece Nombre del ministerio al que pertenece Y presiona el botón "Generar"

Tabla 3.9 - SC 1: Reportar proyectos contratados nivel Secretaria Técnica.

Id del escenario	Escenario	Nombre del proyecto	Nombre del ente	Nombre del ministerio	Respuesta del Sistema	Resultado de la Prueba
EC 1	El actor selecciona la opción de generar reporte de los proyectos contratados.	Se selecciona todos los proyectos de ese ente	Se selecciona un ente	Se selecciona un ministerio	El sistema genera un reporte en formato PDF	Tiene un resultado satisfactorio
		Se selecciona un proyecto	Se selecciona un ente	Se selecciona un ministerio		
		Se selecciona todos los proyectos de ese ente	Se selecciona todos los entes	Se selecciona todos los ministerio		
		Se selecciona todos los proyectos de ese ente	Se selecciona todos los entes	Se selecciona un ministerio		
		Se selecciona todos los proyectos de ese ente	Se selecciona todos los entes	Se selecciona un ministerio	El sistema muestra un mensaje si no hay proyectos contratados.	
		Se selecciona todos los proyectos de ese ente	Se selecciona un ente	Se selecciona un ministerio		
		Se selecciona todos los proyectos de ese ente	Se selecciona todos los entes	Se selecciona todos los ministerio		

En el caso del resultado visual de los reportes, se comprobó comparando con los prototipos no funcionales descritos en la especificación de los casos de uso (ver Anexo 2). Durante las iteraciones de la revisión se identificaron un total de 15 no conformidades, la mayoría precisamente por problemas de diseño. A continuación se presentan ejemplo de algunas de ellas, identificadas en la

primera iteración de pruebas que se hizo a los reportes ProyContratadosNivelEE y ProyContratadosNivelST:

Proyectos Contratados a Nivel de Ente Ejecutor

Convenio Integral de Cooperación Cuba-Venezuela

Ministerio : Universidad de las Ciencias Informáticas

Ente Ejecutor: Albet S.A.

Parte : Cuba

miércoles, 03 de junio de 2009

Nombre del contrato	Fecha de	Ministerio Contraparte	Ente Contraparte	Proyectos	Invertir en	Transferir a Cuba	Monto
Producción de tomates e industria	04-12-2008	Ministerio del Poder Popular para la Energía y Petróleo	Fundación para el Desarrollo del Servicio Eléctrico	Producción de tomates e industria	331.658,97	79.380,00	411.038,97
				Total:	331.658,97	79.380,00	411.038,97

Figura 3.6: No conformidad detectada en el reporte ProyContratadosNivelEE.

En este caso se requería que el formato de las cifras que representan el monto total de los proyectos contratados separara con una coma los dos últimos lugares. Se detectó también que los encabezados de dos columnas estaban incompletos.

Proyectos Contratados a Nivel Secretaría Técnica

Convenio Integral de Cooperación Cuba-Venezuela

Parte : Cuba

Monday, september 13 , 2008

Ministerio	Ente Ejecutor	Nombre del Contrato	Fecha de Firma	Ministerio Contraparte	Ente Contraparte	Proyectos	Invertir en Venezuela	Transferir a Cuba	Monto
Ministerio de la Agricultura	Empresa de Informática de la Agricultura	Red Nacional de Comunicación Agrícola	04-12-2008	Ministerio del Poder Popular para la Agricultura y Tierras	Instituto Socialista de la Pesca	Red Nacional de Comunicación Agrícola	452.817,00	0.00	452.817,00
						Total:	452.817,00	0,00	452.817,00

Ministerio	Ente Ejecutor	Nombre del Contrato	Fecha de Firma	Ministerio Contraparte	Ente Contraparte	Proyectos	Invertir en Venezuela	Transferir a Cuba	Monto
Universidad de las Ciencias Informáticas	Albet S.A.	Producción de tomates e industria	04-12-2008	Ministerio del Poder Popular para la Energía y Petróleo	Fundación para el Desarrollo del Servicio Eléctrico	Producción de tomates e industria	331.658,97	79.380,00	411.038,97
						Total:	331.658,97	79.380,00	411.038,97
								Monto Total :	63.855,97

Figura 3.7: No conformidad detectada en el reporte ProyContratadosNivelST.

En el caso anterior se detectó que la fecha estaba en inglés y que el monto total no estaba sumando correctamente.

A continuación se muestra un gráfico que representa la cantidad de no conformidades que se detectaron por cada iteración y como estas fueron disminuyendo a medida que avanzaba el proceso de desarrollo y se iban perfeccionando los reportes:

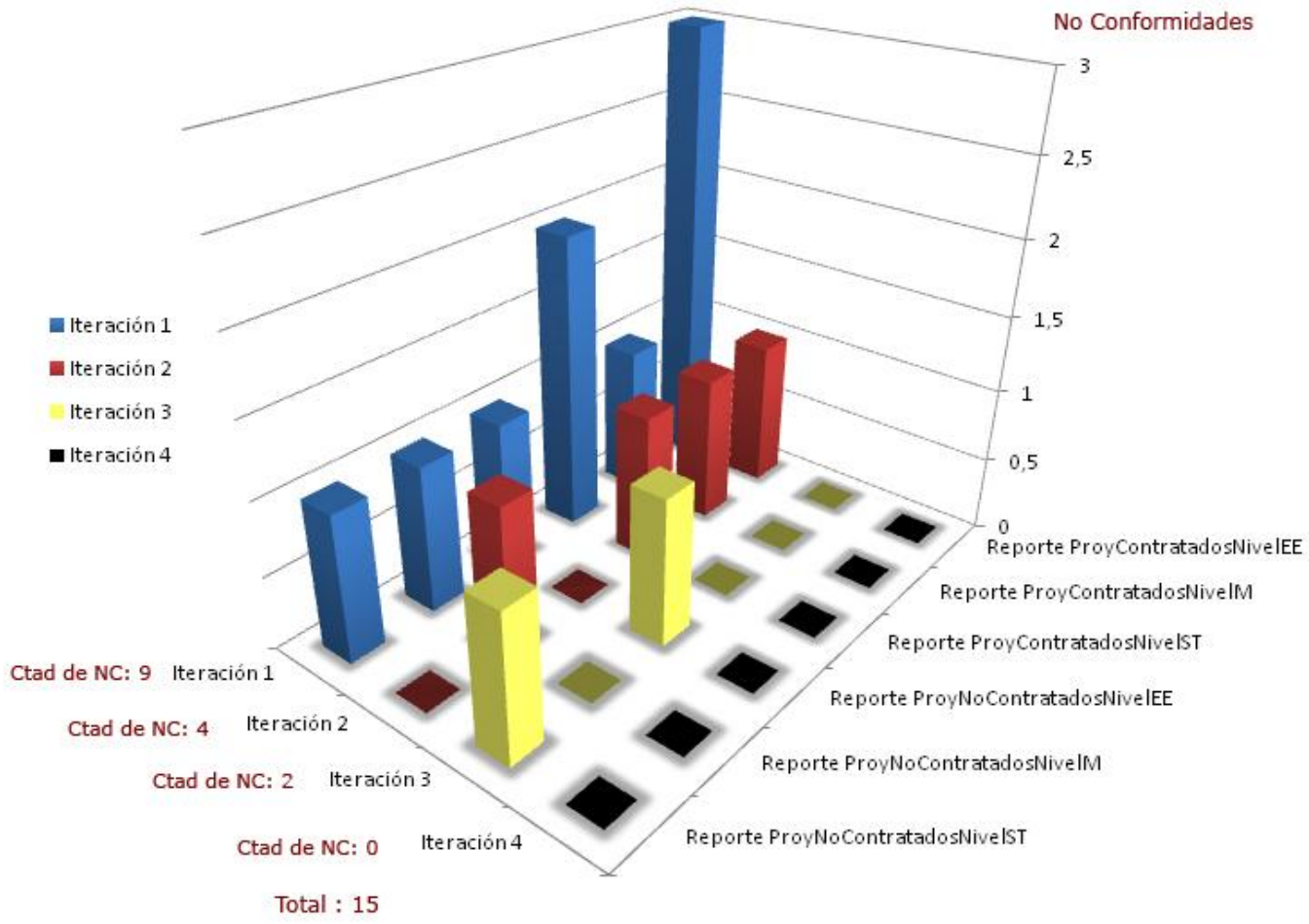


Figura 3.8: No conformidades detectadas en las iteraciones de prueba de caja negra.

Como resultado de la última iteración se obtuvo la liberación de la documentación por parte del equipo de calidad de la universidad, como constancia de la factibilidad de los artefactos analizados.

Luego de liberados los artefactos anteriormente mencionados, conjunto con los demás generados por el módulo Contratación, fueron presentados a los clientes del sistema, los cuales procedieron a su revisión, con el objetivo de verificar que la especificación mostrada cumplía con todas las expectativas, necesidades, condiciones y propiedades que esperaban con el desarrollo del módulo Contratación, incluyendo los reportes que debería saber generar el mismo. Los artefactos presentados fueron aceptados, quedando de esta forma legalmente firmadas en el acta de aceptación (**ver Anexo 9**) y el acta de finalización oficial del proyecto (**ver Anexo 10**), lo cual plasma la satisfacción del cliente con el sistema CCV.

3.2. Conclusiones del capitulo

En este capítulo se presentaron diferentes métodos y procedimientos que fueron necesarios para validar la calidad de las funcionalidades del subsistema Reportes, tanto interna como externa. Para analizar los resultados obtenidos primeramente se comprobó la factibilidad del diseño del subsistema mediante métricas de diseño. Se comprobó que el subsistema Reportes tiene una mínima dependencia con el único subsistema que se relaciona y por ende presenta bajo acoplamiento, también presenta alta cohesión y las clases identificadas son relativamente pequeñas lo que significa que no están sobrecargadas de responsabilidades. Un factor importante que determinó un sencillo diseño fue la integración de JasperReport con el framework Spring el cual aporta su modelo MVC para el trabajo con reportes.

Por otra parte se mostró como fueron validadas las funcionalidades que fueron implementadas a través de diferentes pruebas de caja blanca, en este caso pruebas de unidades. Los resultados demostraron como la integración de JasperReport con el Spring al reducir la cantidad y complejidad de líneas de código propician una menor cantidad de errores. Por último también se mostró como las funcionalidades externas del subsistema respondieron adecuadamente a los requisitos funcionales identificados para los reportes. Las diferentes pruebas de caja negra que se hicieron iterativamente garantizaron la satisfacción plena de las necesidades reales de los usuarios y de las demandas del cliente.

CONCLUSIONES

Al finalizar el presente trabajo se arribó a las siguientes conclusiones:

- Después de hacer un estudio del estado del arte acerca de las distintas tecnologías y herramientas generadoras de reportes que existen para aplicaciones empresariales en Java y de código abierto, se llegó a la conclusión de que JasperReports e Ireport, en su conjunto, fueron las más adecuadas para satisfacer los requisitos funcionales y no funcionales identificados por los analistas principales del proyecto.

- Es importante señalar las ventajas que trae como resultado la arquitectura que propone el framework Spring y su integración con JasperReports para generar de una manera más sencilla los reportes: Esta integración por un lado permitió la simplificación del diseño de las clases que encapsula el subsistema Reportes, garantizando por ende la alta cohesión y el bajo acoplamiento de este subsistema, lo que determina la mínima dependencia con respecto a otros subsistemas. Por otro lado permitió que durante la implementación se simplificara el código con respecto a la forma tradicional de generar reportes usando JasperReport por sí solo, garantizando que las clases tuvieran menos responsabilidades y por ende se facilitara la reutilización de las mismas. Durante el proceso de desarrollo se aplicaron métricas de diseño cuyos resultados validaron lo anteriormente expuesto.

- Finalmente las pruebas realizadas para validar las funcionalidades que fueron implementadas, demuestran que el subsistema Reportes cumple satisfactoriamente con los requisitos que garantizan su correcto funcionamiento. De esta manera se le da solución al problema científico planteado en la investigación del presente trabajo.

RECOMENDACIONES

- Los reportes exigidos en los requerimientos, fueron previamente definidos por los clientes, por esta razón el diseño de los mismos se realizó de una manera estática puesto que el resultado visual siempre es el mismo. En caso de que se le quiera agregar cierto dinamismo a los reportes, se recomienda estudiar y utilizar DynamicJasper. El mismo es una librería open-source que permite al desarrollador crear rápidamente una gran variedad de reportes a través de una intuitiva API escrita en Java. Esta permite definir programáticamente las columnas, grupos, totales, gráficos (charts), sub-reportes, el formato de salida (pdf, Excel, html, etc.) en tiempo de ejecución.

- Se recomienda estudiar y utilizar JfreeChart. La misma es una biblioteca de clase Java para la generación de gráficas que incluye soporte para: gráficas de pie, gráficas de barra, gráficas lineales, gráficas de intervalos de tiempo, planillas métricas, planillas de candelabro, y otros elementos que ayudan en el momento de tomar decisiones. Se integra de una manera sencilla con JasperReports.

BIBLIOGRAFÍA

[Online]. - http://www.definicion.org/lenguaje-de-programacion.

Apache FOP home page Apache [Online] // The Apache Software Fundation. - mayo 16, 2009. - http://xmlgraphics.apache.org/fop/.

Avilas Diagramas de Secuencia [Online]. - abril 29, 2009. - http://tvdi.det.uvigo.es/~avilas/UML/node42.html.

BEATON W. Eclipse Platform Technical Overview Eclipse.org [Online]. - mayo 17, 2006. - http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.

Burbeck Steve Applications Programming in Smalltalk-80(TM): How to use Model-View ontroller (MVC). [Book]. - 2006.

C. Walls Breidenbach Spring in Action. [PDF] s.l.: Manning Publications [Book]. - 2005.

Ciberaula 2009 Ciberaula [Online] // Ciberaula. - 5 15, 2009. - http://java.ciberaula.com/articulo/tecnologia_java/.

Danciu Teodor The JasperReports Ultimate Guide Version 1.0 [Book]. - 2004.

Definición.org Definición [Online]. - 5 10, 2009. - http://www.definicion.org/lenguaje-de-programacion.

E.T.S.I Departamento de Lenguajes y sistema Informáticos [Online]. - Universidad de Granada.

Hommel S. Convenciones de Código para el lenguaje de programación Java TM [Book]. - 1999.

Hommel S. Convenciones de Código para el lenguaje de programación Java TM [Book]. - 1999.

Ingienería de Software 1 Introducción a la Ingeniería de Software. [Report]. - Habana, Cuba : UCI, 2007.

iText home page Lowaige [Online]. - mayo 23, 2009. - http://www.lowagie.com/itext.

Jacaboson I, Booch G and Rumbaugh J El Proceso Unificado de Desarrollo de Software [Book]. - [s.l.] : Addison Wesley, 2000.

jasperforge.org Sitio Web Oficial de JasperReport [Online]. - mayo 11, 2009. - http://jasperforge.org/website/.

Juristo N, Moreno A and Vegas S TÉCNICAS DE EVALUACIÓN DE SOFTWARE. [Book]. - 2006.

Kruchten P. The Rational Unified Process: An Introduction [Book]. - [s.l.] : Addison Wesley , 200.

Massol Vincent and Husted Ted JUnit in action [Book]. - 209 Bruce Park Avenue, Geenwich 06830 : Manning Publications Co, 2004. - ISBN 1-930 110-99-5.

Microsystems Sun El lenguaje de Programación Java™ [Book].

Perrone Paul J and Krishna J2EE Developer's Handbook [Book]. - Indianapolis, Indiana : Sam's Publishing, 2003.

Pressman Roger S Ingenieria de Software, un enfoque práctico. [Book]. - 2005.

Saavedra Jorge Word Press [Online]. - abril 25, 2009. - http://jorgesaavedra.wordpress.com/category/patrones-grasp/.

Schildt Herbert Java 2 Manual de referencia [Book]. - Madrid : [s.n.], 2001.

SlideShare 2009 SlideShare [Online]. - mayo 24, 2009. - http://www.slideshare.net/remitos/sw-magazine-mx1.

Soluciones Informáticas 2009 Free Land Site [Online] // Soluciones Informáticas. - mayo 5, 2009. - http://www.freelandsite.es.

UDELAB UDELAB [Online] // Universidad de las americas Puebla. - 2009. - 2009. - http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/gonzalez_d_h/capitulo4.pdf.

Universidad P. Valencia Departamento de Sistemas Informáticos y Computación. Rational Unified Process [Online]. - abril 10, 2009. - http://www.rational.com.ar/herramientas/rup.html.

Usaola M Escuela Superior de Informática [Online]. - abril 25, 2009. - http://www.inf-cr.uclm.es..

Vicente R. Programania [Online]. - mayo 9, 2009. - http://www.programania.net/diseno-de-software/spring-y-junit-4/.

Visual Paradigm 2005 [Online]. - mayo 3, 2009. - . http://www.visual-paradigm.com/product/vpuml/..

Visual Paradigm International Visual Paradigm [Online] // Visual Paradigm. - 2005. - http://www.visual-paradigm.com/product/vpuml/.

Zárate Rea Hector Paradigmas de Programacion [Report]. - Ciudad Mexico : [s.n.], 2008.

ANEXOS

Anexo 1 - **Representación del Diagrama de Casos de Uso.**

Un diagrama de casos de uso es una representación gráfica de parte o el total de los actores y casos de uso del sistema, incluyendo sus interacciones. Todo sistema tiene como mínimo un diagrama principal de casos de uso, que es una representación gráfica del entorno del sistema (actores) y su funcionalidad principal (casos de uso). Un diagrama de casos de uso muestra, por tanto, los distintos requisitos funcionales que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios u otras aplicaciones). [18]

En este caso el siguiente diagrama de casos de uso responde a los seis requisitos funcionales que han sido identificados a partir de las necesidades reales de los usuarios y de las demandas del cliente, resultado del trabajo de captura de requisitos para los reportes identificados en el Módulo de Contratación. Los requisitos funcionales se pueden observar en el Anexo del documento.

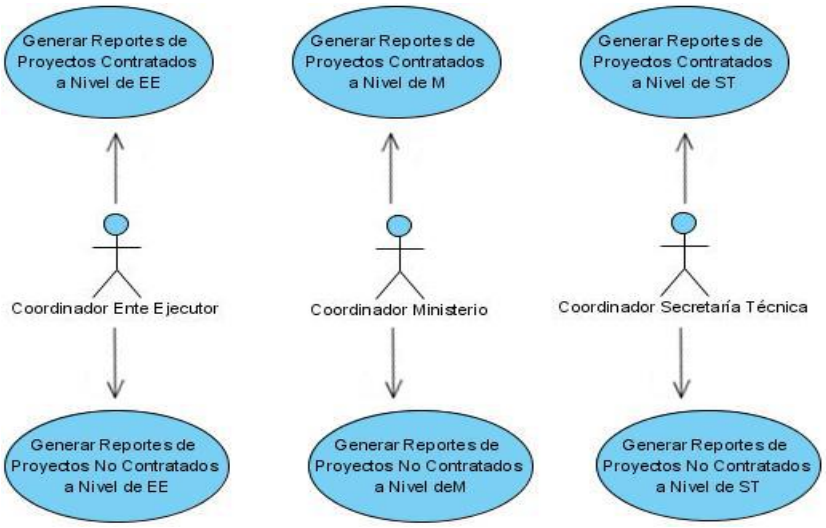


Figura A.1 : Diagrama de Casos de Uso del Subsistema Reportes

Anexo 2 - Especificación de Casos de Uso.

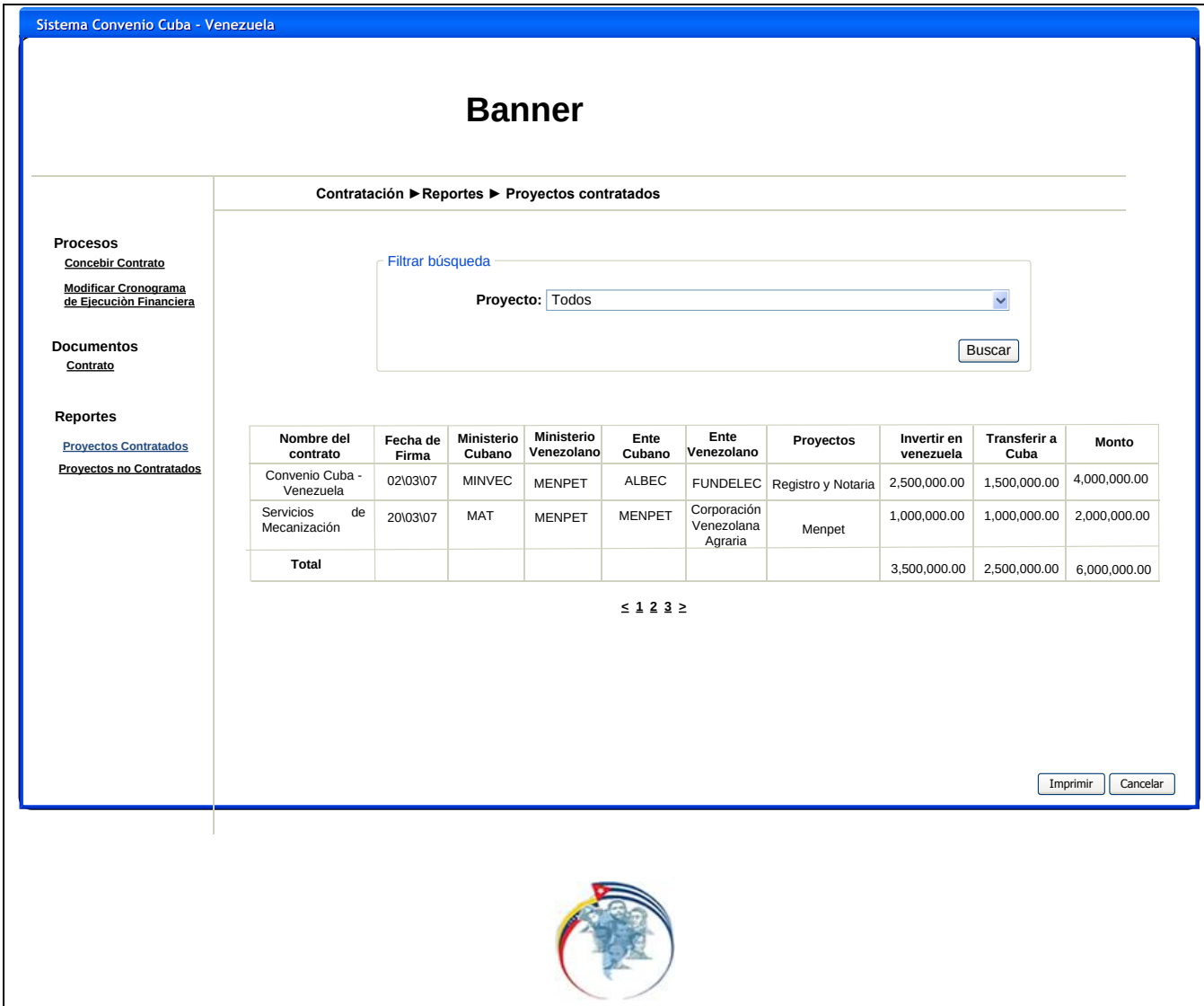
Para establecer qué funcionalidades serán implementadas, la especificación de los casos de uso referentes a los reportes identificados, sirve de guía a lo largo de todo el proceso de trabajo para convertir los requisitos en un producto final, adaptado a las demandas del cliente y a las necesidades de los usuarios. Las especificaciones de los 6 casos de uso identificados en el módulo contratación se detallan a continuación y ayudarán a tener una visión más detallada de los requisitos funcionales, incluyendo los prototipos visuales de las plantillas de los reportes para su posterior diseño en la herramienta IReport (ver **Anexo 8**). Cada caso de uso responde a un requisito funcional.

Tabla A1- Generar Reporte de Proyectos Contratados a nivel de EE

Caso de Uso:	Generar Reporte de Proyectos Contratados a nivel de EE
Actores:	Coordinador EE
Resumen:	El caso de uso inicia cuando se genera un reporte de todos los proyectos contratados pertenecientes al ente.
Precondiciones:	1. El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.
Referencias	RF02.033, RF02.035
Prioridad	Secundario
Nivel	Usuario

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor solicita un reporte de los proyectos contratados.	2. El sistema muestra una interfaz que permite realizar una búsqueda de proyecto contratados por: • Proyectos

3. El actor solicita realizar la búsqueda especificando un criterio.	4. El sistema realiza la búsqueda por el criterio especificado mostrando los siguientes datos: • Nombre del Contrato • Fecha de Firma • Ministerio Cubano • Ministerio Venezolano • Ente Cubano • Ente Venezolano • Proyectos • Invertir en Venezuela • Transferir a Cuba • Monto
5. El actor solicita imprimir.	6. El sistema genera un documento en formato PDF con el reporte el cual está listo para ser impreso, terminando así el caso de uso.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1. El actor cancela la acción de imprimir.	5.2. El sistema vuelve a la interfaz de inicio.
Poscondiciones	1. Se obtiene el reporte de los proyectos contratados. 2. Se imprime el reporte de los proyectos contratados.

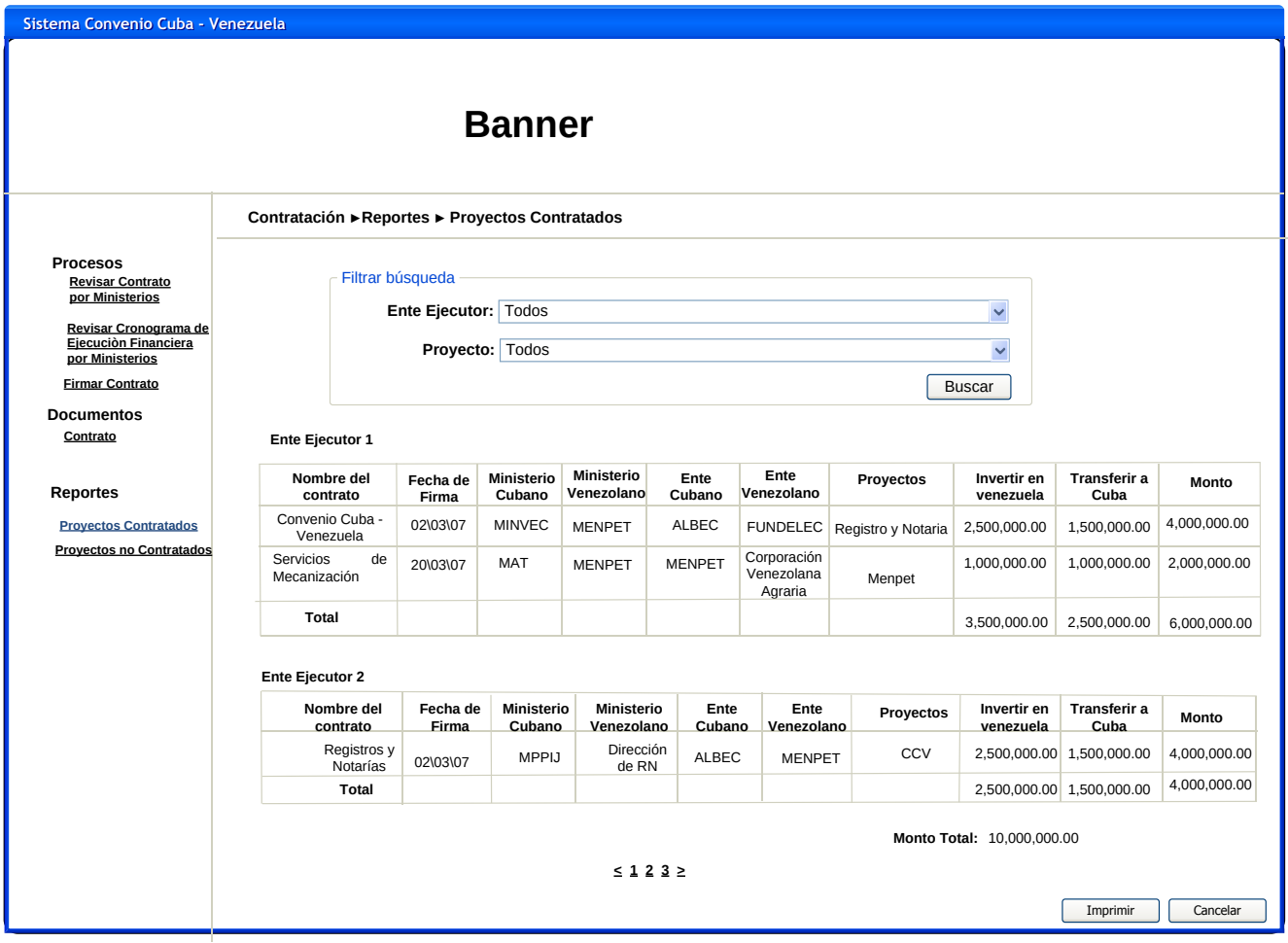
Tabla A2- Generar Reporte de Proyectos Contratados a nivel de Ministerio

Caso de Uso:	Generar Reporte de proyectos contratados a nivel de Ministerio.
Actores:	Coordinador de M
Resumen:	El caso de uso inicia cuando se genera un reporte de todos los proyectos contratados pertenecientes al Ministerio.
Precondiciones:	1. El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.
Referencias	RF02.033, RF02.035
Prioridad	Secundario
Nivel	Usuario

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor solicita un reporte de los proyectos contratados.	2. El sistema muestra una interfaz que permite realizar una búsqueda de proyecto contratados por: • Ente Ejecutor • Proyectos
3. El actor solicita realizar la búsqueda especificando un criterio.	4. El sistema realiza la búsqueda por el criterio especificado, en caso de que se seleccione un ente específico se muestran los proyectos del mismo sino de todos los entes del ministerio, mostrando los siguientes datos: • Nombre del Contrato • Fecha de Firma • Ministerio Cubano • Ministerio Venezolano • Ente Cubano • Ente Venezolano • Proyectos • Invertir en Venezuela • Transferir a Cuba • Monto

5. El actor solicita imprimir.	6. El sistema genera un documento en formato PDF con el reporte el cual está listo para ser impreso, terminando así el caso de uso.

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

5.1. El actor cancela la acción de imprimir.	5.2. El sistema vuelve a la interfaz de inicio.
Poscondiciones	1. Se obtiene el reporte de los proyectos contratados. 2. Se imprime el reporte de los proyectos contratados.
	3.

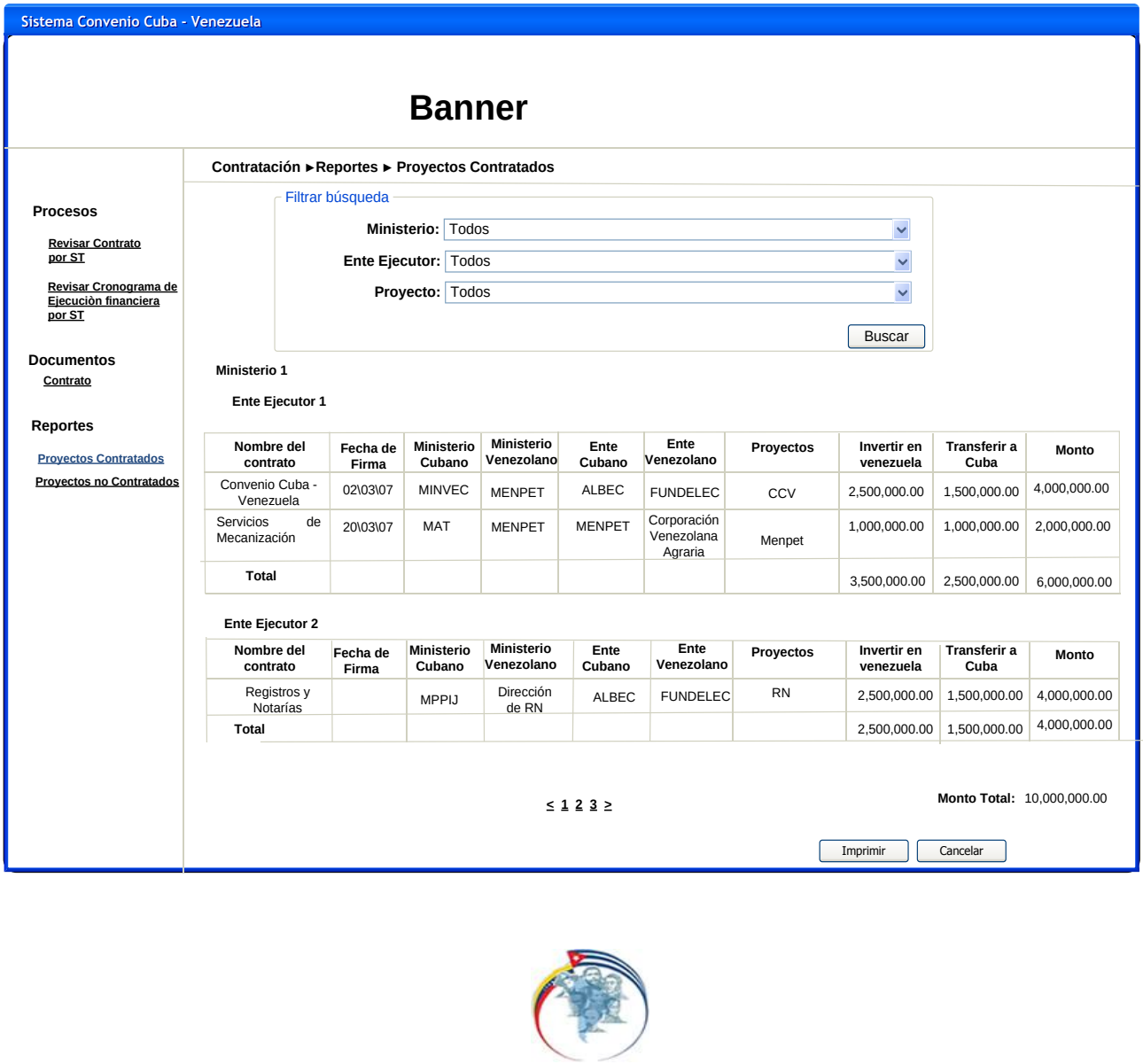
Tabla A3- Generar Reporte de Proyectos Contratados a nivel de ST

Caso de Uso:	Generar Reporte de proyectos contratados a nivel de Secretaría Técnica
Actores:	Coordinador ST
Resumen:	El caso de uso inicia cuando se genera un reporte de todos los proyectos contratados a nivel de ST.
Precondiciones:	1. El sistema debe estar instalado y ejecutado correctamente. El actor debe estar autenticado con los permisos necesarios.
Referencias	RF02.033, RF02.035
Prioridad	Secundario
Nivel	Usuario

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor solicita un reporte de los proyectos contratados.	2. El sistema muestra una interfaz que permite realizar una búsqueda de proyecto contratados por: • Ministerio • Ente Ejecutor • Proyectos
3. El actor solicita realizar la búsqueda especificando un criterio.	4. El sistema realiza la búsqueda por el criterio especificado, mostrando los siguientes datos agrupados por ministerios y por entes: • Nombre del Contrato • Fecha de Firma • Ministerio Cubano • Ministerio Venezolano • Ente Cubano • Ente Venezolano • Proyectos • Invertir en Venezuela • Transferir a Cuba

	• Monto
5. El actor solicita imprimir.	6. El sistema genera un documento en formato PDF con el reporte el cual está listo para ser impreso, terminando así el caso de uso.

Prototipo de Interfaz



Flujos Alternos

Acción del Actor	Respuesta del Sistema
5.1. El actor cancela la acción de imprimir.	5.2. El sistema vuelve a la interfaz de inicio.

Poscondiciones	1. Se obtiene el reporte de los proyectos contratados. 2. Se imprime el reporte de los proyectos contratados.

Tabla A4- Generar Reporte de proyectos no Contratados a nivel de EE

Caso de Uso:	Generar Reporte de proyectos no Contratados a nivel de EE
Actores:	Coordinador EE.
Resumen:	El caso inicia cuando se genera un reporte de los proyectos que no se encuentran contratados a nivel de EE.
Precondiciones:	1. El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.
Referencias	RF02.034, RF02.036
Prioridad	Secundario
Nivel	Usuario

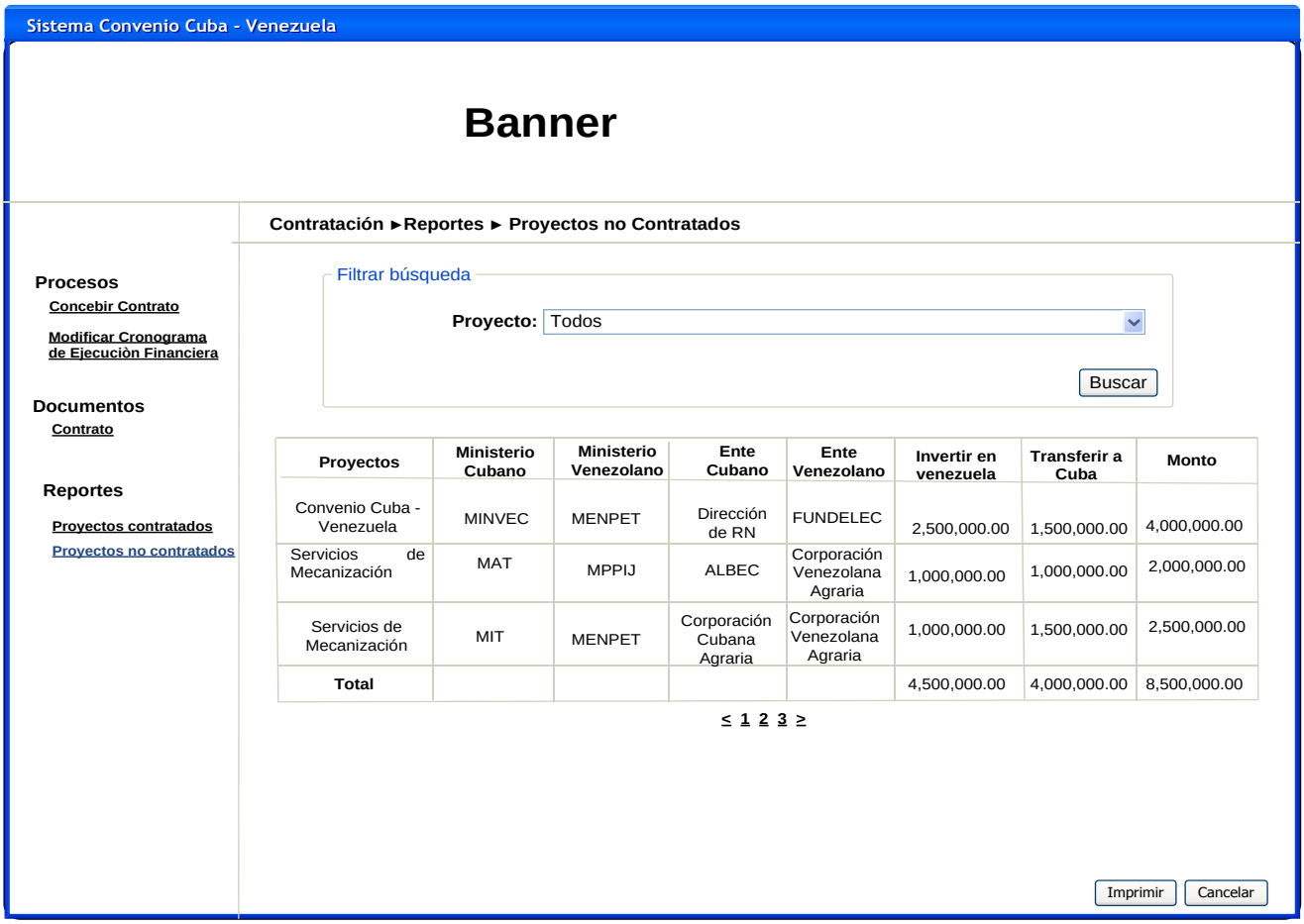
Flujo Normal de Eventos

Sección "Reporte de proyectos no Contratados"

Acción del Actor	Respuesta del Sistema
1. El actor solicita un reporte de los proyectos no contratados.	2. El sistema muestra una interfaz que permite realizar una búsqueda de proyecto contratados por: • Proyectos
3. El actor solicita realizar la búsqueda especificando un criterio.	4. El sistema realiza la búsqueda por el criterio especificado mostrando los siguientes datos: • Proyectos

	• Ministerio Cubano • Ministerio Venezolano • Ente Cubano • Ente Venezolano • Invertir en Venezuela • Transferir a Cuba • Monto
5. El actor solicita imprimir.	6. El sistema genera un documento en formato PDF con el reporte el cual está listo para ser impreso, terminando así el caso de uso.

Prototipo de Interfaz



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1. El actor cancela la acción de imprimir.	5.2. El sistema vuelve a la interfaz de inicio.

Poscondiciones	1. Se obtiene el reporte de los proyectos no contratados. 2. Se imprime el reporte de los proyectos no contratados.

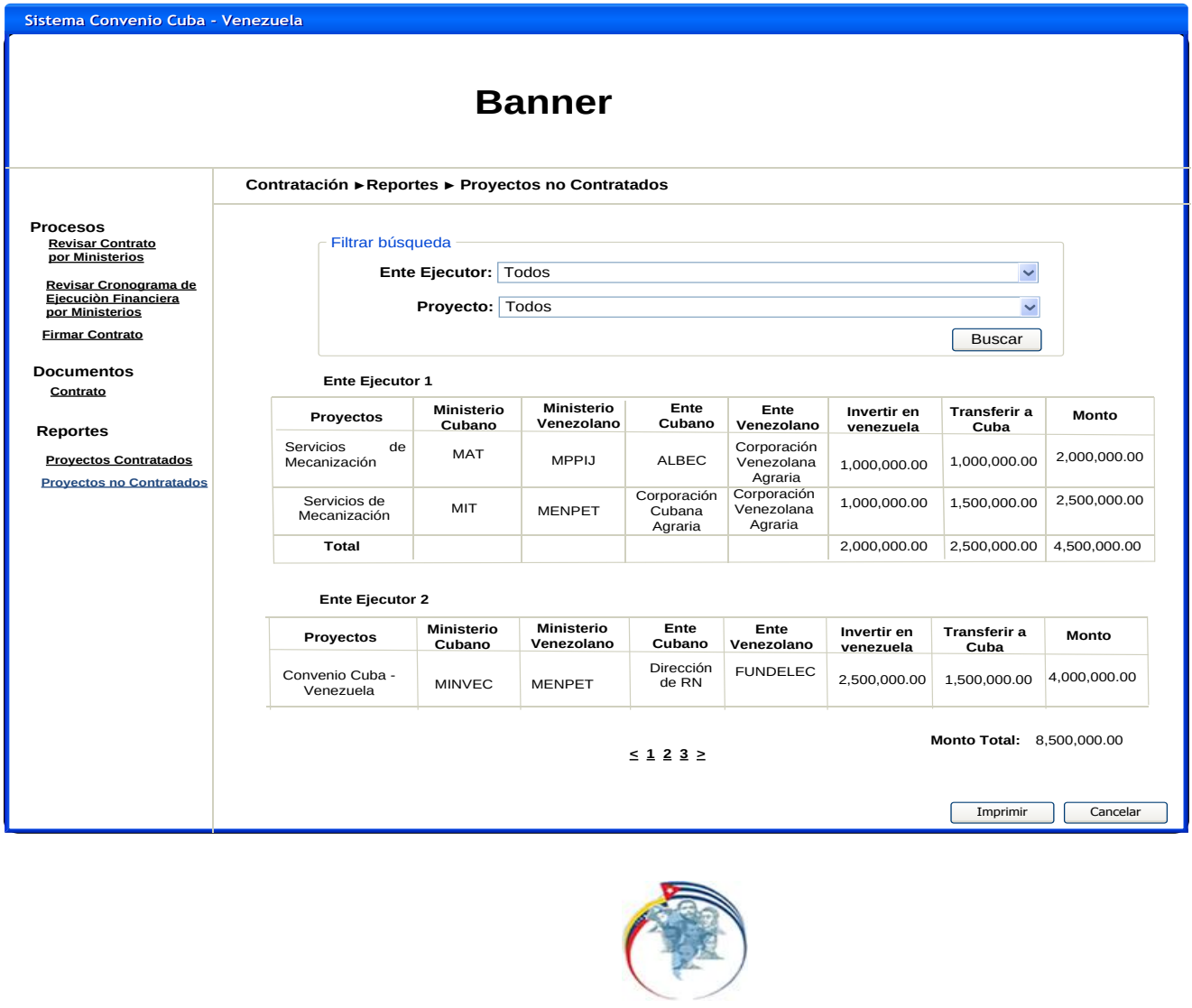
Tabla A5- Generar Reporte de proyectos no Contratados a nivel de Ministerio

Caso de Uso:	Generar Reporte de Proyectos no Contratados a nivel de Ministerios.
Actores:	Coordinador M
Resumen:	El caso inicia cuando se genera un reporte de los proyectos que no se encuentran contratados a nivel de Ministerios.
Precondiciones:	1. El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.
Referencias	RF02.034, RF02.036
Prioridad	Secundario
Nivel	Usuario

Flujo Normal de Eventos	
Sección "Reporte de proyectos no Contratados"	
Acción del Actor	**Respuesta del Sistema**
1. A El actor solicita un reporte de los proyectos no contratados.	2. El sistema muestra una interfaz que permite realizar una búsqueda de proyecto no contratados por: • Ente Ejecutor • Proyectos
3. El actor solicita realizar la búsqueda especificando un criterio.	4. El sistema realiza la búsqueda por el criterio especificado en caso de que se seleccione un ente específico se muestran los proyectos del mismo sino de todos los entes del ministerio, mostrando los siguientes datos: • Proyectos • Ministerio Cubano • Ministerio Venezolano • Ente Cubano

	• Ente Venezolano • Invertir en Venezuela • Transferir a Cuba • Monto
5. El actor solicita imprimir.	6. El sistema genera un documento en formato PDF con el reporte el cual está listo para ser impreso, terminando así el caso de uso.

Prototipo de Interfaz

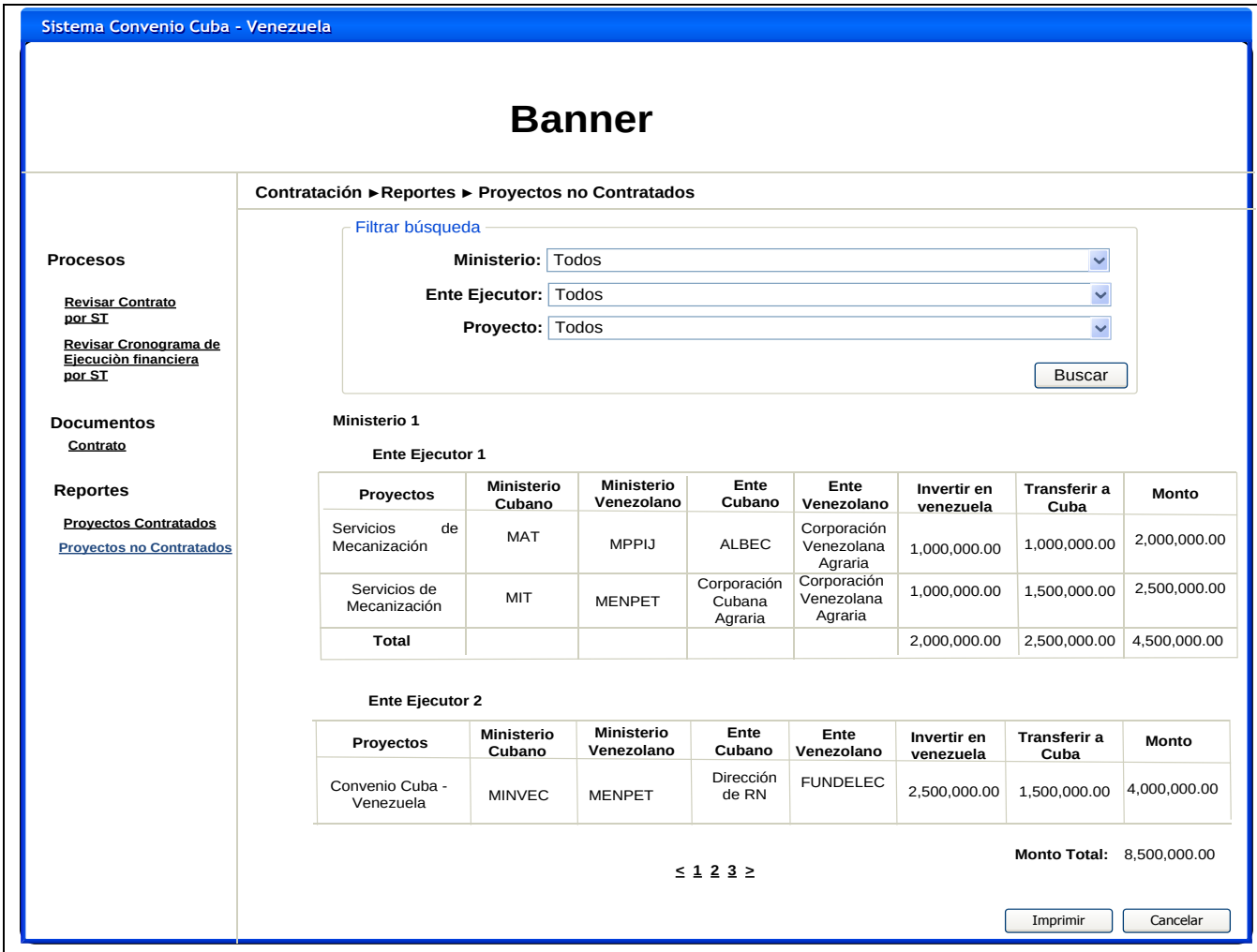


Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1. El usuario el usuario cancela la acción de imprimir.	5.2. El sistema vuelve a la interfaz de inicio.
Poscondiciones	1. Se obtiene el reporte de los proyectos no contratados. 2. Se imprime el reporte de los proyectos no contratados.

Tabla A6 - Generar Reporte de Proyectos no Contratados a nivel de ST

Caso de Uso:	Generar Reporte de Proyectos no Contratados a nivel de ST
Actores:	Coordinador ST.
Resumen:	El caso inicia cuando se genera un reporte de los proyectos que no se encuentran contratados a nivel de ST.
Precondiciones:	1. El sistema debe estar instalado y ejecutándose correctamente. El actor debe estar autenticado con los permisos necesarios.
Referencias	RF02.034, RF02.036
Prioridad	Secundario
Nivel	Usuario
Flujo Normal de Eventos	
Sección "Reporte de proyectos no Contratados"	
Acción del Actor	**Respuesta del Sistema**
1. El actor solicita un reporte de los proyectos no contratados.	2. El sistema muestra una interfaz que permite realizar una búsqueda de proyecto contratados por: • Ministerio • Ente Ejecutor • Proyectos

3. El actor solicita realizar la búsqueda especificando un criterio.	4. El sistema realiza la búsqueda por el criterio especificado, mostrando los siguientes datos: • Proyectos • Ministerio Cubano • Ministerio Venezolano • Ente Cubano • Ente Venezolano • Invertir en Venezuela • Transferir a Cuba • Monto
5. El actor solicita imprimir.	6. El sistema genera un documento en formato PDF con el reporte el cual está listo para ser impreso, terminando así el caso de uso.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
5.1. El actor cancela la acción de imprimir.	5.2. El sistema vuelve a la interfaz de inicio.

Poscondiciones	1. Se obtiene el reporte de los proyectos no contratados. 2. Se imprime el reporte de los proyectos no contratados.

Anexo 3 - Descripción de las clases que conforman el subsistema Reportes

Tabla A7 - Descripción de la Clase ProyContratadosNivelMController.

Nombre:	ProyContratadosNivelMController	
Tipo de clase	Controladora	
Atributo		**Tipo**
datasource		JRDataSourcefichaService
reporteContratacionService		ReporteContratacionService
Responsabilidades		
Nombre:	**Descripción:**	
getReporteContratacionService()	Método que devuelve el objeto reporteContratacionService	
setReporteContratacionService(ReporteContratacionService reporteContratacionService)	Método que setea el objeto reporteContratacionService	
handleRequest(HttpServletRequest arg0,HttpServletResponse arg1)	Método que devuelve el ModelAndView que requiere la petición. El ModelAndView contiene el modelo con los datos y el nombre de la vista.	
getModel(dir,idProyecto,idEnte,Parte)	Método que construye y devuelve el modelo, adicionándole al mismo la dirección física donde se encuentra la imagen del logotipo del reporte, la fecha actual en el momento de la generación del reporte, el DataSource con los datos de los proyectos contratados a nivel de ministerio y los parámetros: invertirtotal, transferirtotal y montototal, es decir, toda la información necesaria que será distribuida posteriormente en el diseño de la plantilla del reporte	

Tabla A8 - Descripción de la Clase ProyContratadosNivelSController.

Nombre:	ProyContratadosNivelSController	
Tipo de clase	Controladora	
Atributo		**Tipo**
datasource reporteContratacionService		JRDataSourcefichaService ReporteContratacionService
Responsabilidades		
Nombre:	**Descripción:**	
getReporteContratacionService()	Método que devuelve el objeto reporteContratacionService	
setReporteContratacionService(ReporteContratacionService reporteContratacionService)	Método que setea el objeto reporteContratacionService	
handleRequest(HttpServletRequest arg0,HttpServletResponse arg1)	Método que devuelve el ModelAndView que requiere la petición. El ModelAndView contiene el modelo con los datos y el nombre de la vista.	
getModel(dir,idProyecto,idEnte,Parte)	Método que construye y devuelve el modelo, adicionándole al mismo la dirección física donde se encuentra la imagen del logotipo del reporte, la fecha actual en el momento de la generación del reporte, el DataSource con los datos de los proyectos contratados a nivel de Secretaria y los parámetros: invertirtotal, transferirtotal y montototal, es decir, toda la información necesaria que será distribuida posteriormente en el diseño de la plantilla del reporte	

Tabla A9 - Descripción de la Clase ProyNoContratadosNivelEEController.

Nombre:	**ProyNoContratadosNivelEEController**
Tipo de clase	Controladora
Atributo	**Tipo**
datasource	JRDataSourcefichaService
reporteContratacionService	ReporteContratacionService
Responsabilidades	
Nombre:	**Descripción:**
getReporteContratacionService()	Método que devuelve el objeto reporteContratacionService
setReporteContratacionService(ReporteContratacionService reporteContratacionService)	Método que setea el objeto reporteContratacionService
handleRequest(HttpServletRequest arg0,HttpServletResponse arg1)	Método que devuelve el ModelAndView que requiere la petición. El ModelAndView contiene el modelo con los datos y el nombre de la vista.
getModel(dir,idProyecto,idEnte,Parte)	Método que construye y devuelve el modelo, adicionándole al mismo la dirección física donde se encuentra la imagen del logotipo del reporte, la fecha actual en el momento de la generación del reporte, el DataSource con los datos de los proyectos no contratados a nivel de Ente Ejecutor y los parámetros: invertirtotal, transferirtotal y montototal, es decir, toda la información necesaria que será distribuida posteriormente en el diseño de la plantilla del reporte

Tabla A10 - Descripción de la Clase ProyNoContratadosNivelMController.

Nombre:	**ProyNoContratadosNivelMController**
Tipo de clase	Controladora
Atributo	**Tipo**
datasource reporteContratacionService	JRDataSourcefichaService ReporteContratacionService
Responsabilidades	
Nombre:	**Descripción:**
getReporteContratacionService()	Método que devuelve el objeto reporteContratacionService
setReporteContratacionService(ReporteContratacionService reporteContratacionService)	Método que setea el objeto reporteContratacionService
handleRequest(HttpServletRequest arg0,HttpServletResponse arg1)	Método que devuelve el ModelAndView que requiere la petición. El ModelAndView contiene el modelo con los datos y el nombre de la vista.
getModel(dir,idProyecto,idEnte,Parte)	Método que construye y devuelve el modelo, adicionándole al mismo la dirección física donde se encuentra la imagen del logotipo del reporte, la fecha actual en el momento de la generación del reporte, el DataSource con los datos de los proyectos no contratados a nivel de Ministerio y los parámetros: invertirtotal, transferirtotal y montototal, es decir, toda la información necesaria que será distribuida posteriormente en el diseño de la plantilla del reporte

Tabla A11 - Descripción de la Clase ProyNoContratadosNivelSController.

Nombre:	**ProyNoContratadosNivelSController**
Tipo de clase	Controladora
Atributo	**Tipo**
datasource reporteContratacionService	JRDataSourcefichaService ReporteContratacionService

Responsabilidades	
Nombre:	**Descripción:**
getReporteContratacionService()	Método que devuelve el objeto reporteContratacionService
setReporteContratacionService(ReporteContratacionService reporteContratacionService)	Método que setea el objeto reporteContratacionService
handleRequest(HttpServletRequest arg0,HttpServletResponse arg1)	Método que devuelve el ModelAndView que requiere la petición. El ModelAndView contiene el modelo con los datos y el nombre de la vista.
getModel(dir,idProyecto,idEnte,Parte)	Método que construye y devuelve el modelo, adicionándole al mismo la dirección física donde se encuentra la imagen del logotipo del reporte, la fecha actual en el momento de la generación del reporte, el DataSource con los datos de los proyectos no contratados a nivel de Secretaria y los parámetros: invertirtotal, transferirtotal y montototal, es decir, toda la información necesaria que será distribuida posteriormente en el diseño de la plantilla del reporte

Tabla A12 - Descripción de la Clase ReportePCNivelM

Nombre:	**ReportePCNivelM**
Tipo de clase	Entidad
Atributo	**Tipo**
nombreente	String
parte	String
nombreministerio	String
nombrecontrato	String
fechafirma	String
nombreentecontraparte	String
nombreministeriocontraparte	String
invertirvenezuela	String
transferircuba	String
monto	String

proyectos totalmonto totalinvertir totaltransferir	String String String String
Responsabilidades	
Nombre:	**Descripción:**
getNombreente() setNombreente(String nombreente) String getNombrecontrato() setNombrecontrato(String nombrecontrato) setNombreentecontraparte(String nombreentecontraparte) getNombreministeriocontraparte() setNombreministeriocontraparte(String nombreministeriocontraparte) getInvertirvenezuela() setInvertirvenezuela(String invertirvenezuela) getTransferircuba() setTransferircuba(String transferircuba) getMonto() setMonto(String monto) getProyectos() setProyectos(String proyectos) getNombreministerio() setNombreministerio(String nombreministerio) String getFechafirma() setFechafirma(String fechafirma) getParte() setParte(String parte) getTotalMonto() setTotalMonto(String totalmonto) getTotaltransferir() setTotaltransferir(String totaltransferir) getTotalinvertir() setTotalinvertir(String totalinvertir)	Métodos para retornar y setear todos los atributos de la clase.

Tabla A13 - Descripción de la Clase ReportePCNivelS

Nombre:	**ReportePCNivelS**
Tipo de clase	Entidad
Atributo	**Tipo**

nombreente parte nombreministerio nombrecontrato fechafirma nombreentecontraparte nombreministeriocontraparte invertirvenezuela transferircuba monto proyectos totalmonto totalinvertir totaltransferir	String String String String String String String String String String String String String String

Responsabilidades	
Nombre:	**Descripción:**
getNombreente() setNombreente(String nombreente) String getNombrecontrato() setNombrecontrato(String nombrecontrato) setNombreentecontraparte(String nombreentecontraparte) getNombreministeriocontraparte() setNombreministeriocontraparte(String nombreministeriocontraparte) getInvertirvenezuela() setInvertirvenezuela(String invertirvenezuela) getTransferircuba() setTransferircuba(String transferircuba) getMonto() setMonto(String monto) getProyectos() setProyectos(String proyectos) getNombreministerio() setNombreministerio(String nombreministerio) String getFechafirma() setFechafirma(String fechafirma) getParte() setParte(String parte) getTotalMonto() setTotalMonto(String totalmonto) getTotaltransferir() setTotaltransferir(String totaltransferir) getTotalinvertir() setTotalinvertir(String totalinvertir)	Métodos para retornar y setear todos los atributos de la clase.

Tabla A14 - Descripción de la Clase ReportePnoCNivelEE

Nombre:	**ReportePnoCNivelEE**
Tipo de clase	Entidad
Atributo	**Tipo**
parte nombreente nombreministerio nombreproyecto nombreentecontraparte nombreministeriocontraparte invertirvenezuela transferircuba monto	String String String String String String String String String
Responsabilidades	
Nombre:	**Descripción:**
getNombreente() setNombreente(String nombreente) setNombreentecontraparte(String nombreentecontraparte) getNombreministeriocontraparte() setNombreministeriocontraparte(String nombreministeriocontraparte) getInvertirvenezuela() setInvertirvenezuela(String invertirvenezuela) getTransferircuba() setTransferircuba(String transferircuba) getMonto() setMonto(String monto) getNombreministerio() setNombreministerio(String nombreministerio) getParte() setParte(String parte)	Métodos para retornar y setear todos los atributos de la clase.

Tabla A15 - Descripción de la Clase ReportePnoCNivelM

Nombre:	**ReportePnoCNivelM**
Tipo de clase	Entidad
Atributo	**Tipo**

parte nombreente nombreministerio nombreproyecto nombreentecontraparte nombreministeriocontraparte invertirvenezuela transferircuba monto totalmonto totalinvertir totaltransferir	String String String String String String String String String String String String
Responsabilidades	
Nombre:	**Descripción:**
getNombreente() setNombreente(String nombreente) setNombreentecontraparte(String nombreentecontraparte) getNombreministeriocontraparte() setNombreministeriocontraparte(String nombreministeriocontraparte) getInvertirvenezuela() setInvertirvenezuela(String invertirvenezuela) getTransferircuba() setTransferircuba(String transferircuba) getMonto() setMonto(String monto) getNombreministerio() setNombreministerio(String nombreministerio) getParte() setParte(String parte)	Métodos para retornar y setear todos los atributos de la clase.

Tabla A16 - Descripción de la Clase ReportePnoCNivelS

Nombre:	**ReportePnoCNivelS**
Tipo de clase	Entidad
Atributo	**Tipo**
parte nombreente	String String

nombreministerio nombreproyecto nombreentecontraparte nombreministeriocontraparte invertirvenezuela transferircuba monto totalmonto totalinvertir totaltransferir	String String String String String String String String String String
Responsabilidades	
Nombre:	**Descripción:**
getNombreente() setNombreente(String nombreente) setNombreentecontraparte(String nombreentecontraparte) getNombreministeriocontraparte() setNombreministeriocontraparte(String nombreministeriocontraparte) getInvertirvenezuela() setInvertirvenezuela(String invertirvenezuela) getTransferircuba() setTransferircuba(String transferircuba) getMonto() setMonto(String monto) getNombreministerio() setNombreministerio(String nombreministerio) getParte() setParte(String parte)	Métodos para retornar y setear todos los atributos de la clase.

Tabla A 17 - Descripción de la Clase ContratoDataSource_Secretaria

Nombre:	**ContratoDataSource_Secretaria**
Tipo de clase	Clase Auxiliar
Atributo	**Tipo**
secretaria	ReportePCNivelS
Responsabilidades	
Nombre:	**Descripción:**

next()	Este método retorna true si quedan más elementos por iterar, y false en caso contrario.
getFieldValue(JRField field)	En este método se define el modo en que serán obtenidos los campos necesarios, según la estrategia de iteración y de obtención de los mismos, sobre los elementos en cuestión.

Tabla A18 - Descripción de la Clase ProyectoDataSource_Secretaria

Nombre:	**ProyectoDataSource_Secretaria**
Tipo de clase	Clase Auxiliar
Atributo	**Tipo**
secretaria	ReportePnoCNivelS
Responsabilidades	
Nombre:	**Descripción:**
next()	Este método retorna true si quedan más elementos por iterar, y false en caso contrario.
getFieldValue(JRField field)	En este método se define el modo en que serán obtenidos los campos necesarios, según la estrategia de iteración y de obtención de los mismos, sobre los elementos en cuestión.

Tabla A19 - Descripción de la Clase ProyectoDataSource

Nombre:	**ProyectoDataSource**
Tipo de clase	Clase Auxiliar
Atributo	**Tipo**
secretaria	ReportePnoCNivelM

Responsabilidades	
Nombre:	**Descripción:**
next()	Este método retorna true si quedan más elementos por iterar, y false en caso contrario.
getFieldValue(JRField field)	En este método se define el modo en que serán obtenidos los campos necesarios, según la estrategia de iteración y de obtención de los mismos, sobre los elementos en cuestión.

Anexo 4- Diagramas de clases de diseño por CU.

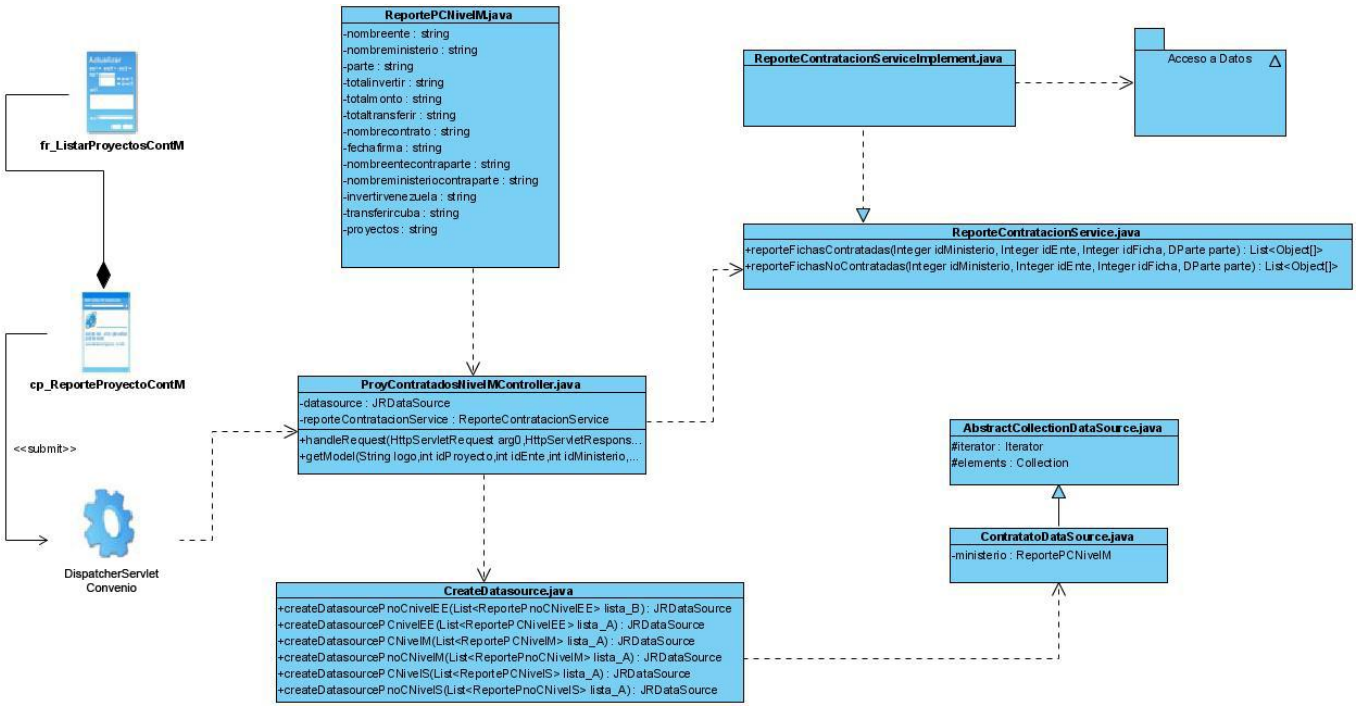


Figura A1 : Diagrama de clases de diseño, CU Generar Reportes de Proyectos Contratados a nivel de Ministerio.

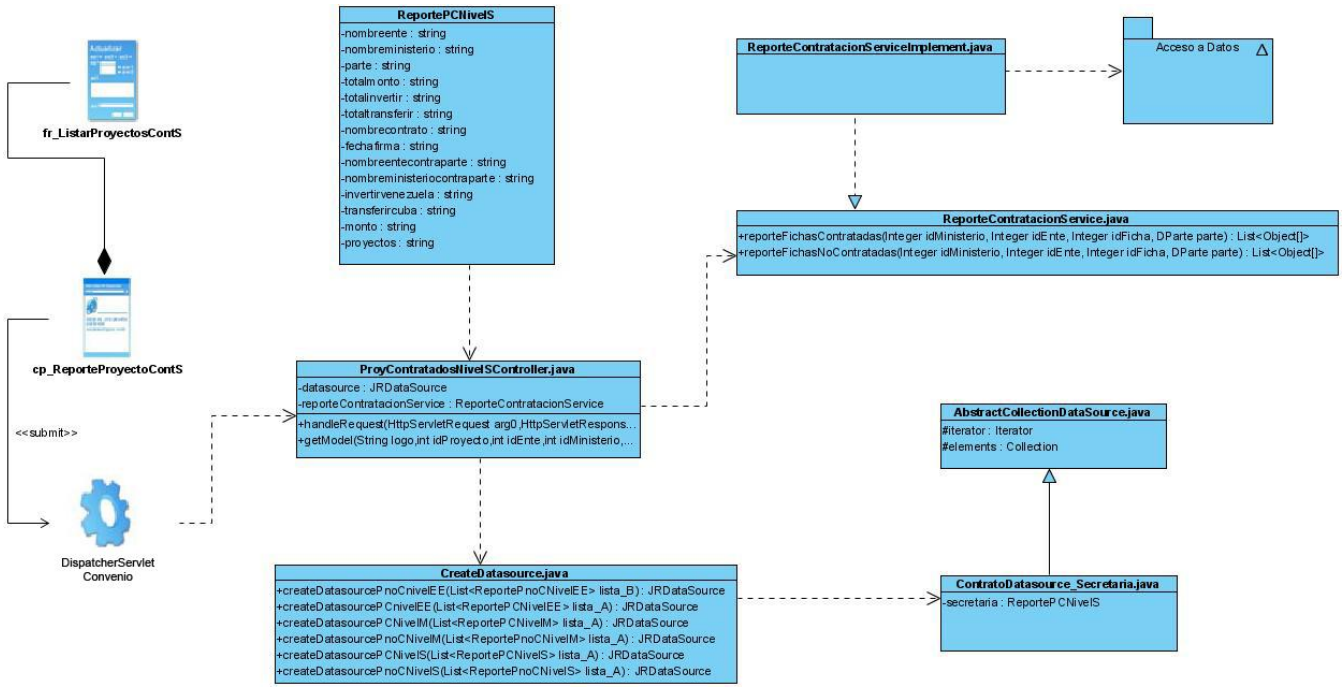


Figura A2 : Diagrama de clases de diseño, CU Generar Reportes de Proyectos Contratados a nivel de Secretaría Técnica.

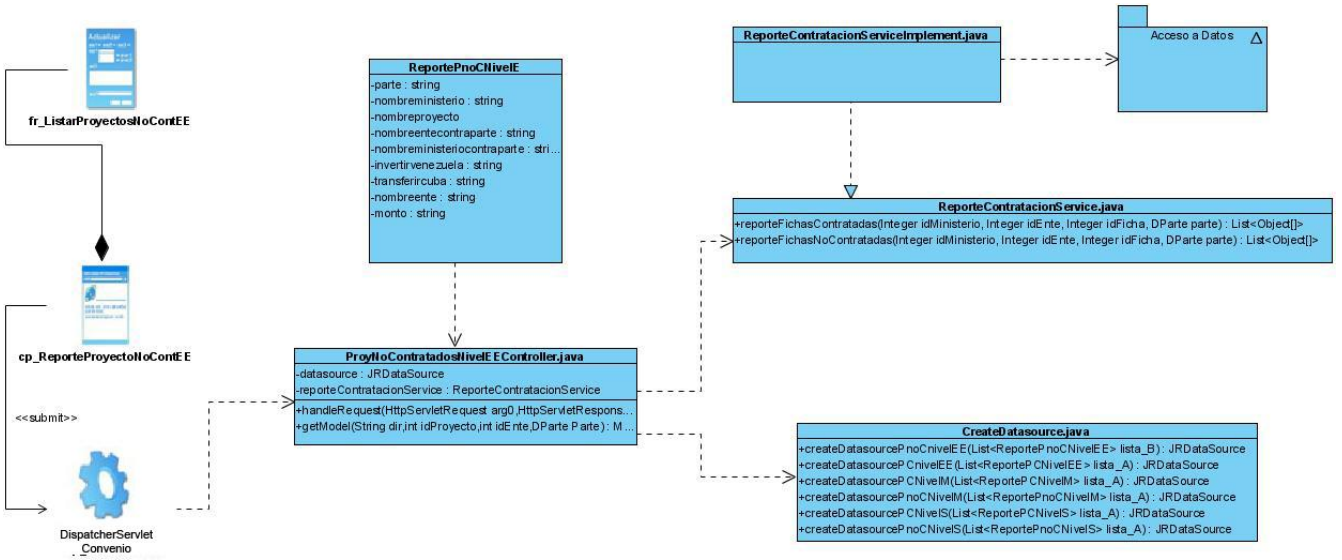


Figura A3 : Diagrama de clases de diseño, CU Generar Reportes de Proyectos No Contratados a nivel de Ente Ejecutor.

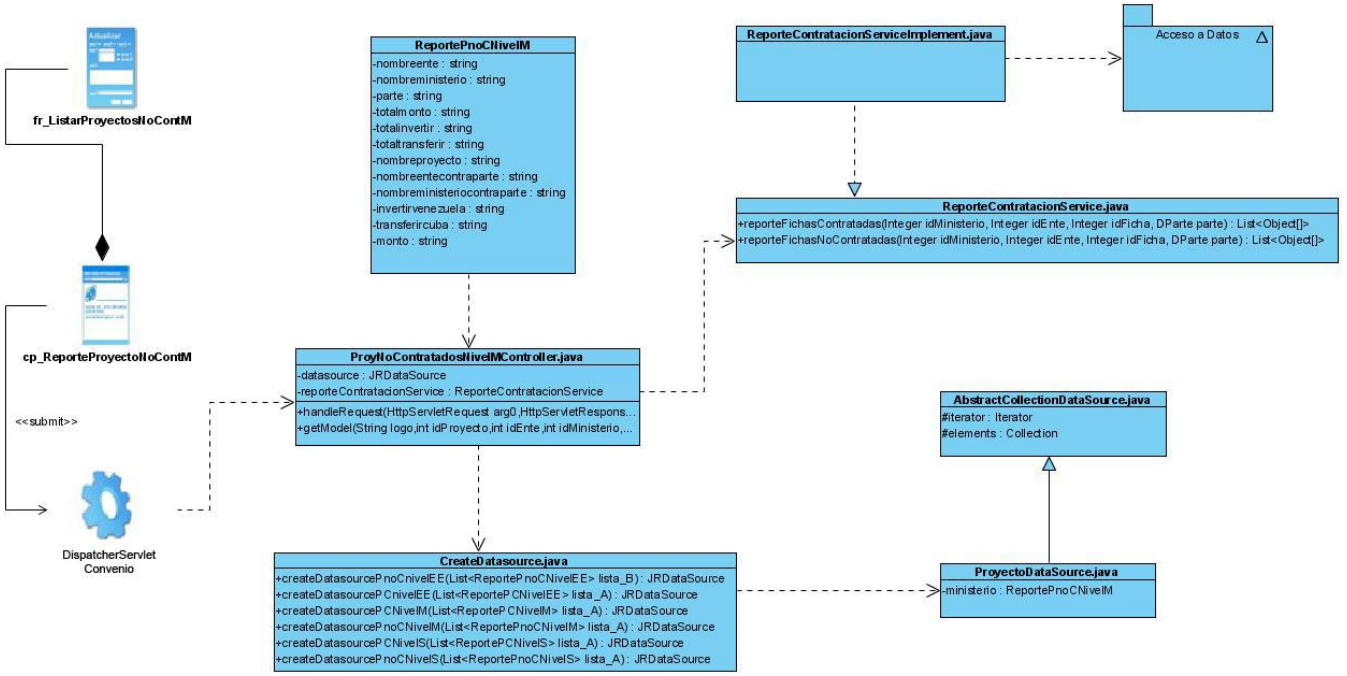


Figura A4 : Diagrama de clases de diseño, CU Generar Reportes de Proyectos No Contratados a nivel de Ministerio.

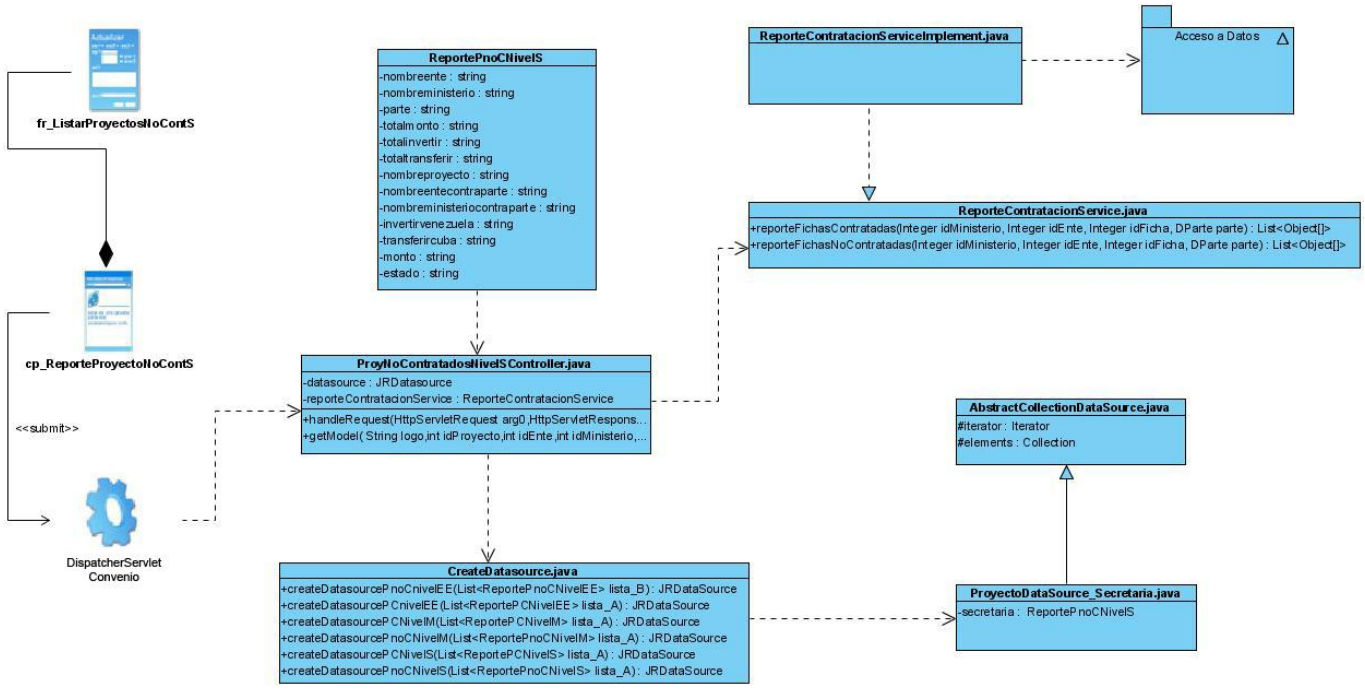


Figura A5 : Diagrama de clases de diseño, CU Generar Reportes de Proyectos No Contratados a nivel de Secretaría Técnica.

Anexo 5- Diagramas de Secuencia por escenario de CU.

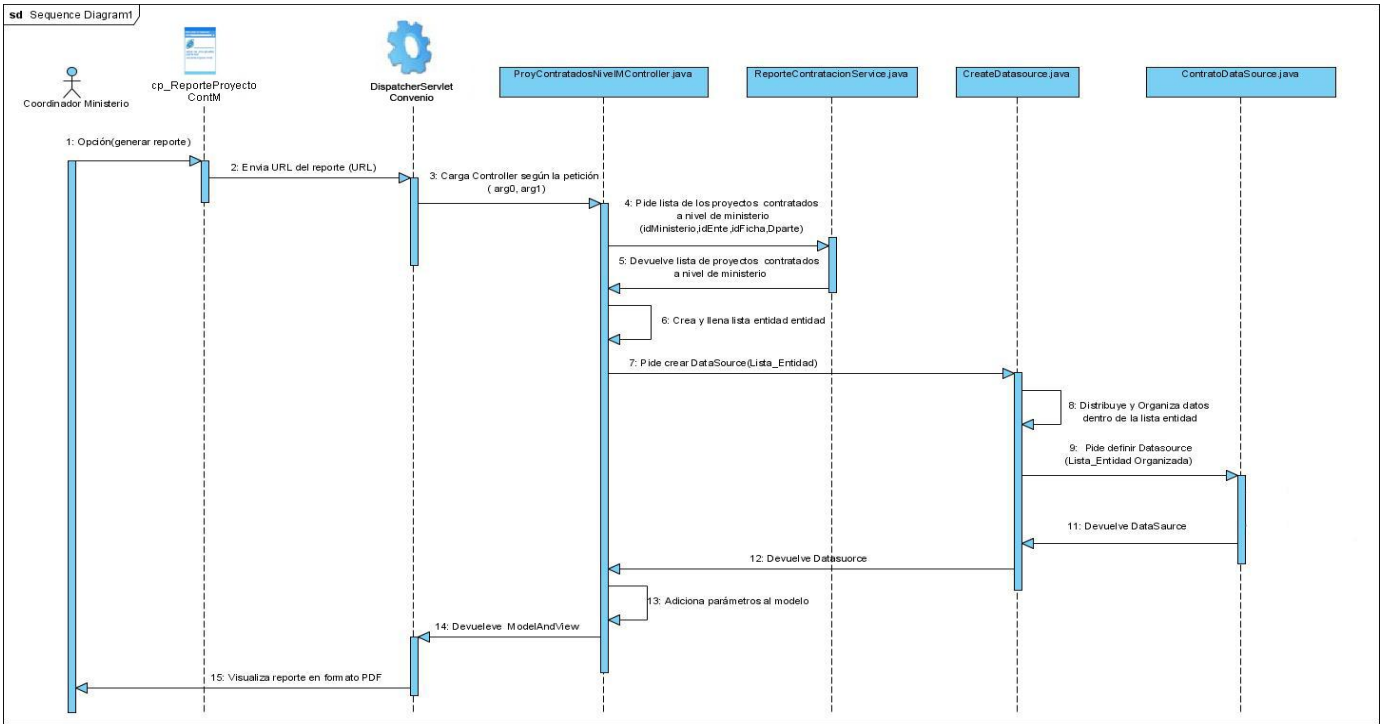


Figura A6: Diagrama de secuencia del CU Generar Reportes de Proyectos Contratados a nivel de Ministerio.

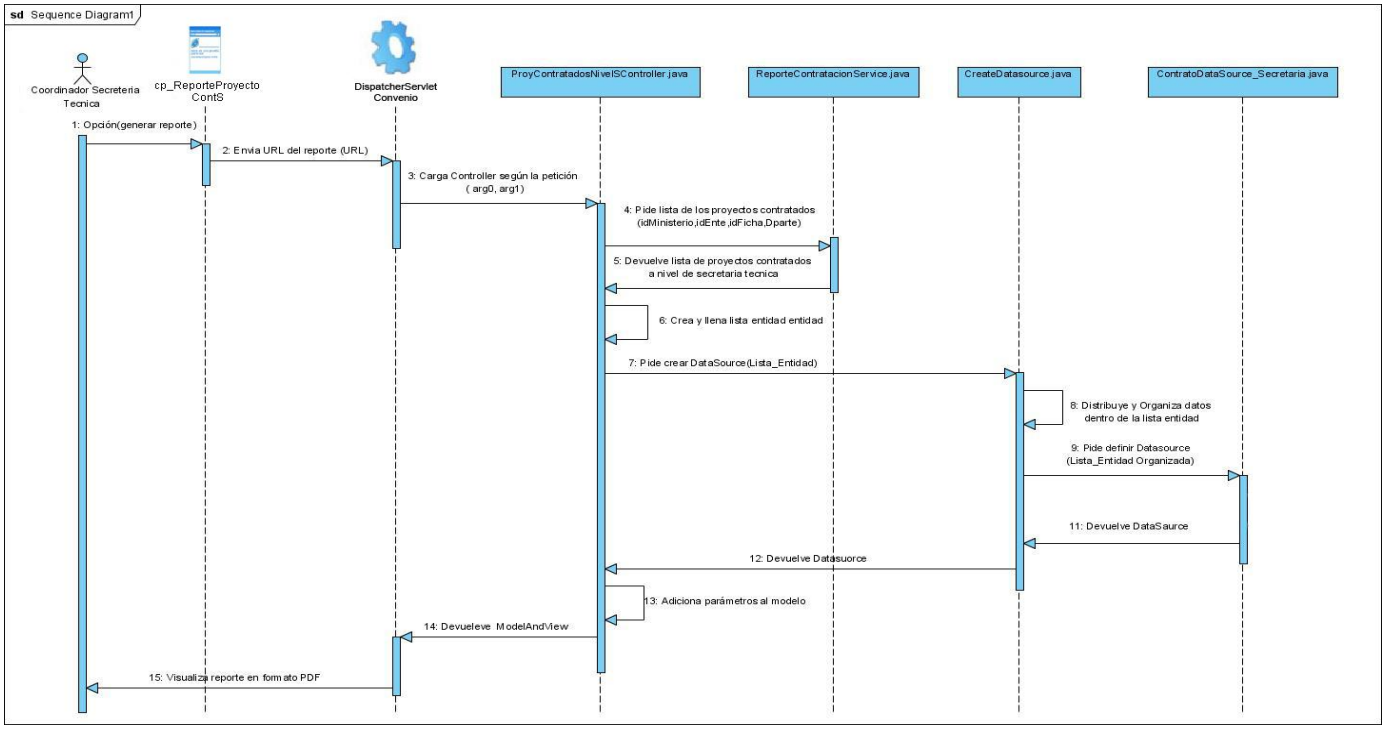


Figura A7: Diagrama de secuencia del CU Generar Reportes de Proyectos Contratados a nivel de Secretaria Técnica.

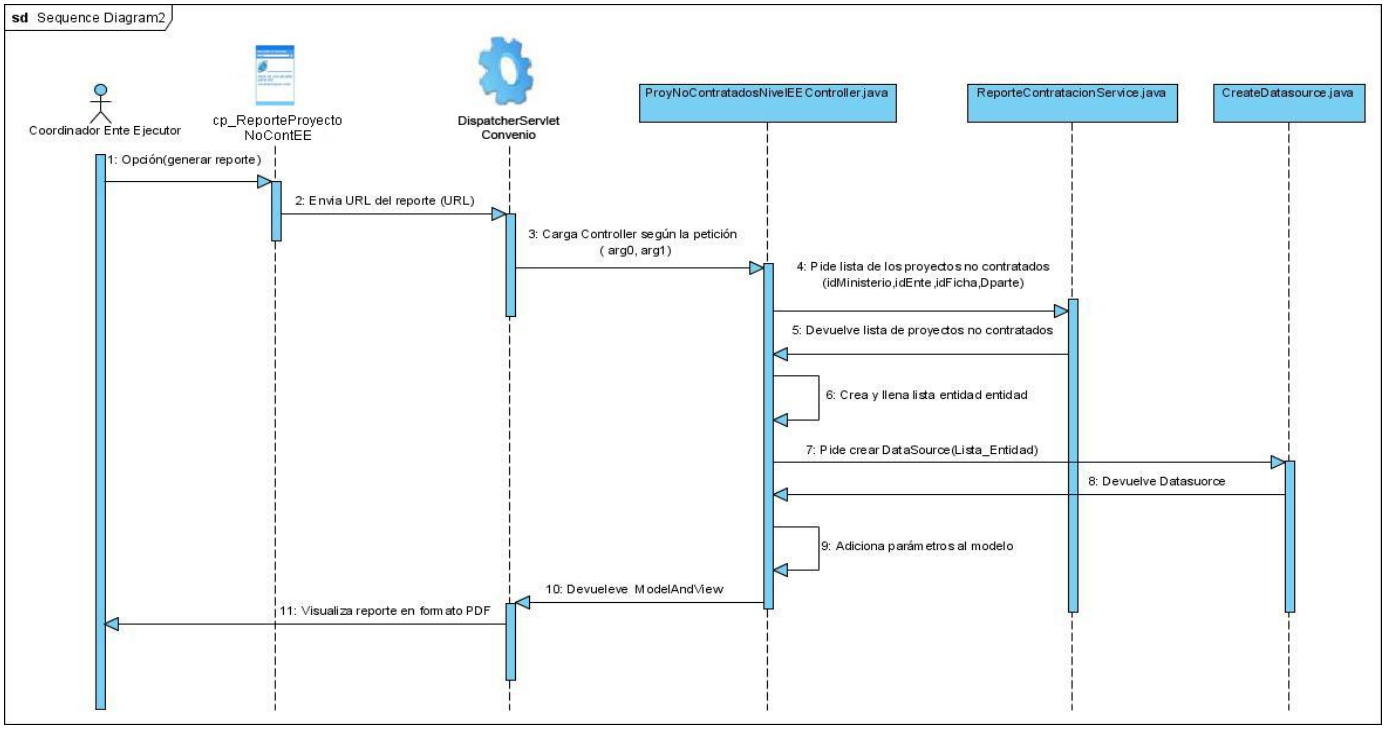


Figura A8: Diagrama de secuencia del CU Generar Reportes de Proyectos no Contratados a nivel de Ente Ejecutor.

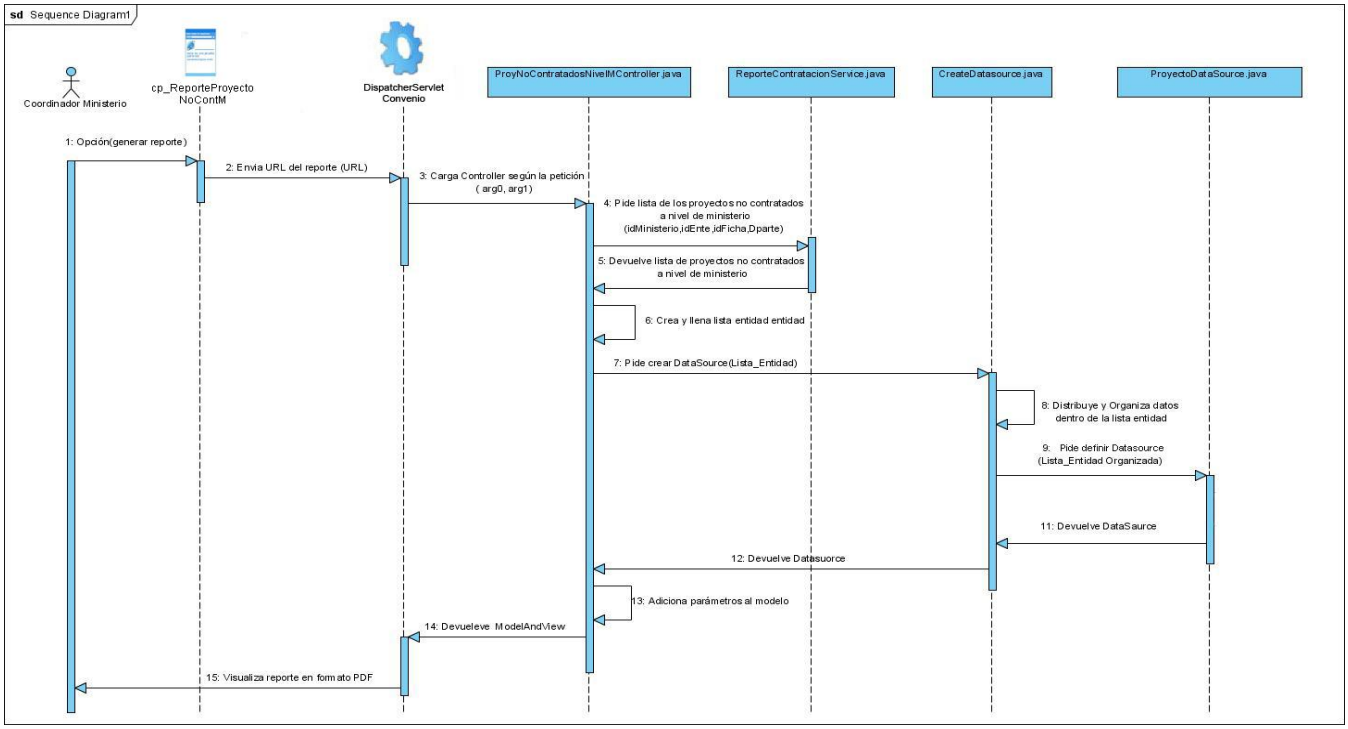


Figura A9 : Diagrama de secuencia del CU Generar Reportes de Proyectos no Contratados a nivel de Ministerio

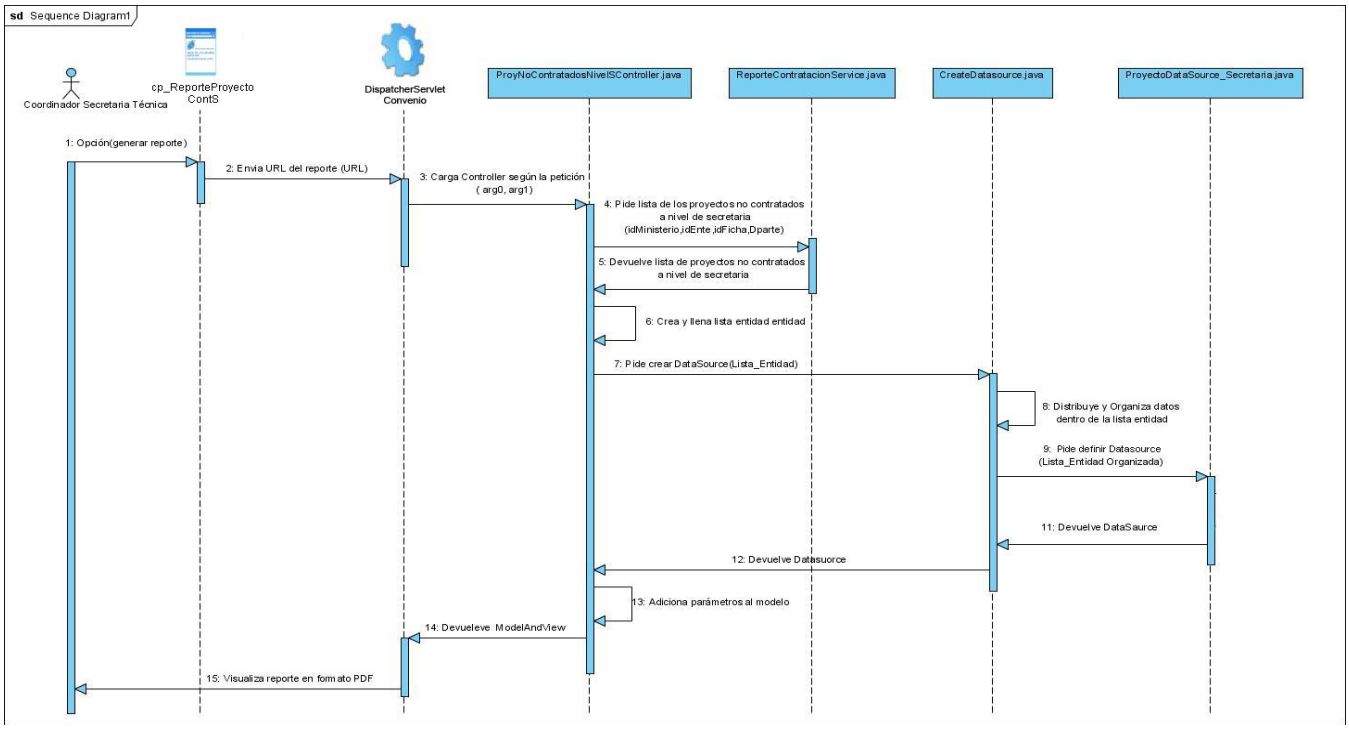


Figura A10: Diagrama de secuencia del CU Generar Reportes de Proyectos no Contratados a nivel de Secretaría Técnica.

Anexo 6 - Pruebas de caja blanca aplicadas al subsistema Reportes.

Las imágenes que se muestran son resultado de las pruebas después de haber pasado por varias iteraciones, como se muestra en la figura anterior, el eje Y representa la cantidad de errores, mientras que el eje X las clases.

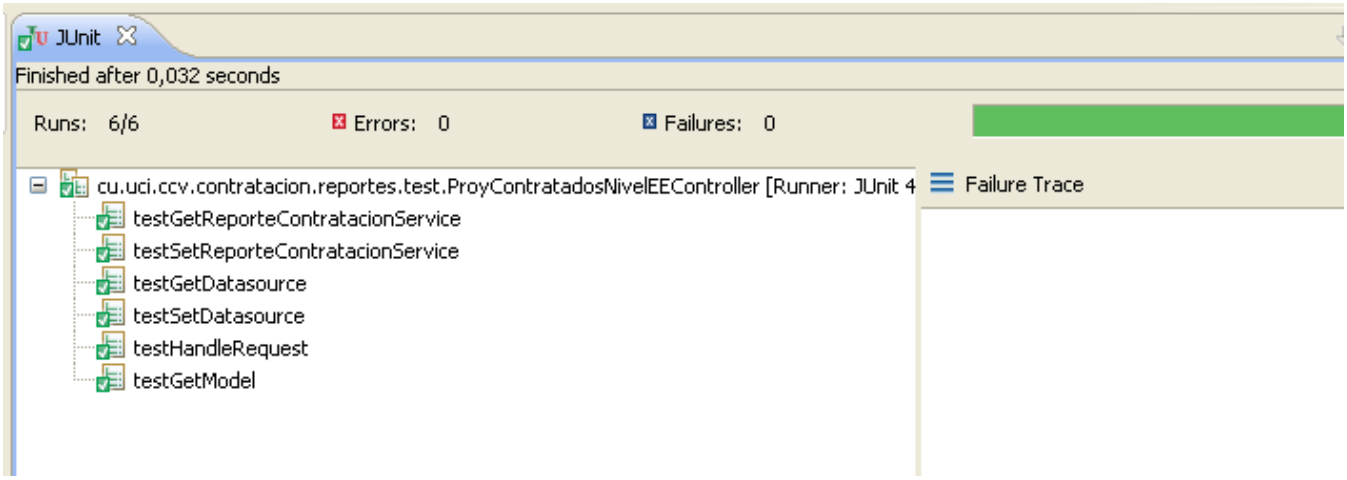


Figura A11: Prueba realizada a la clase ProyContratadosNivelEEController.

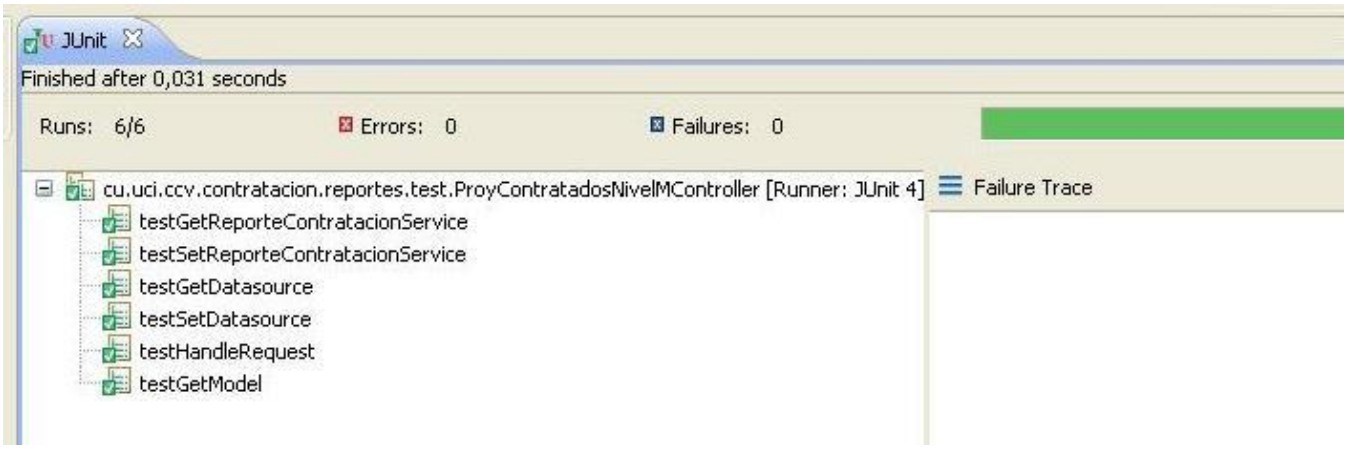


Figura A12: Prueba realizada a la clase ProyContratadosNivelMController.

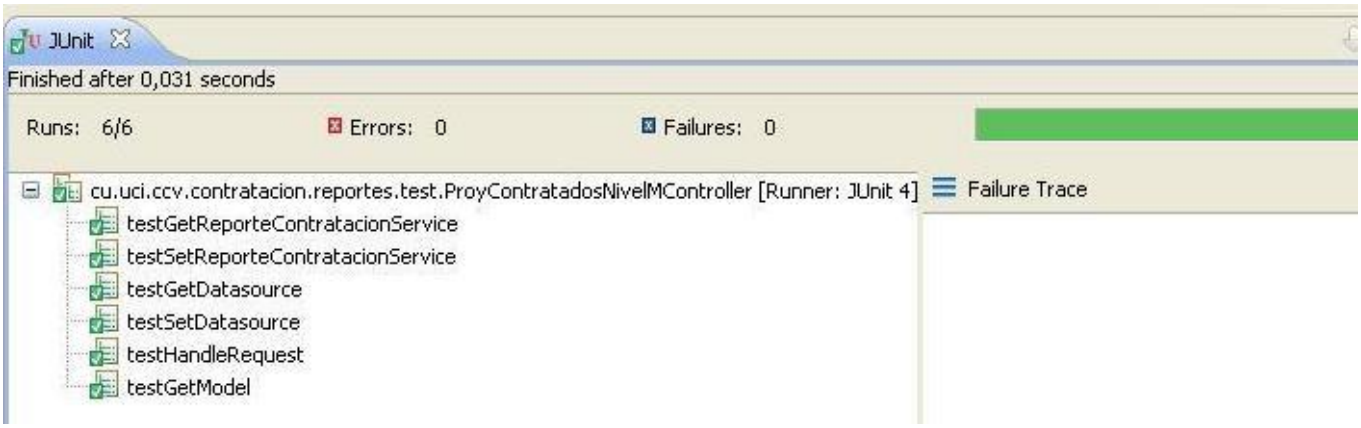


Figura A13: Prueba realizada a la clase ProyContratadosNivelSController.

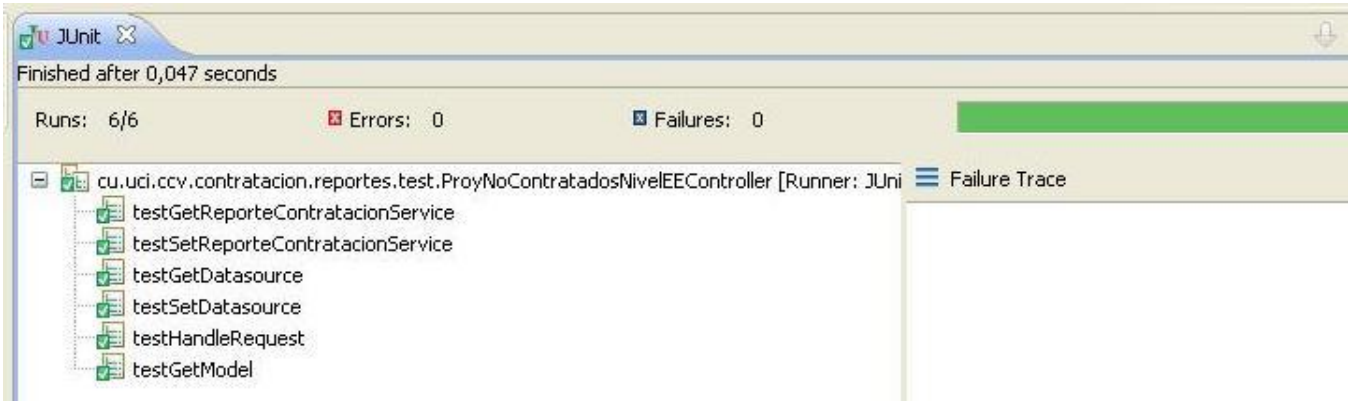


Figura A14: Prueba realizada a la clase ProyNoContratadosNivelEEController.

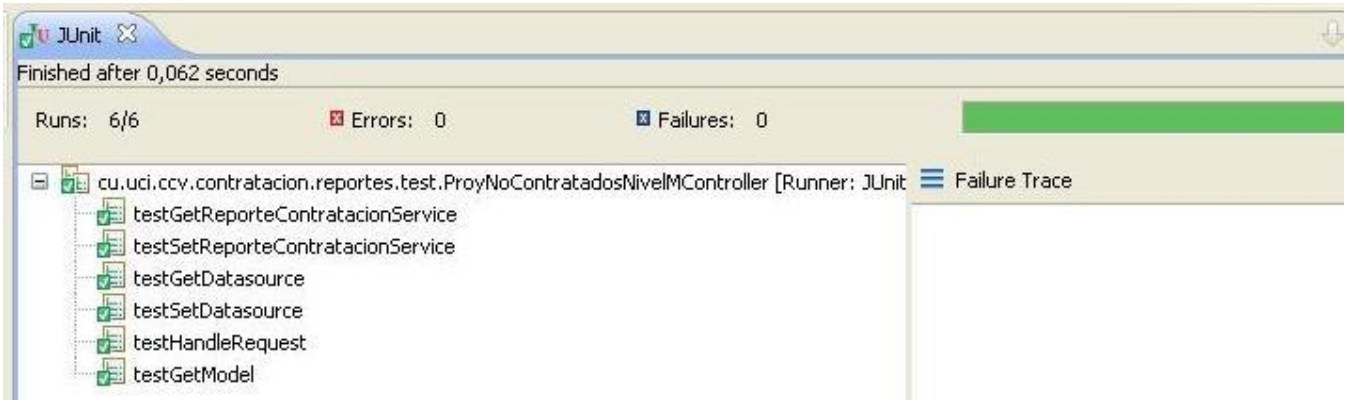


Figura A15: Prueba realizada a la clase ProyNoContratadosNivelMController.

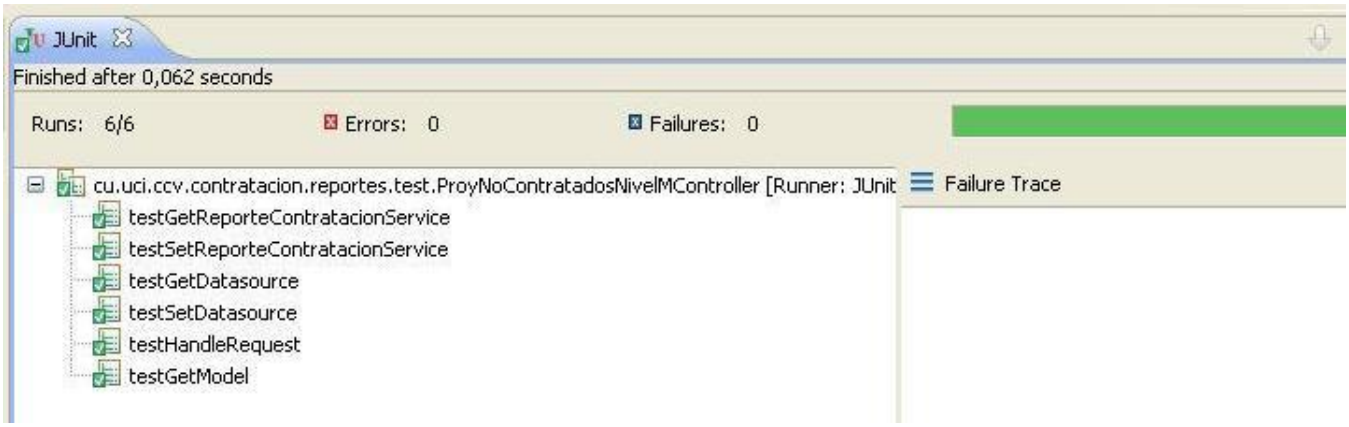


Figura A16: Prueba realizada a la clase ProyContratadosNivelSController.

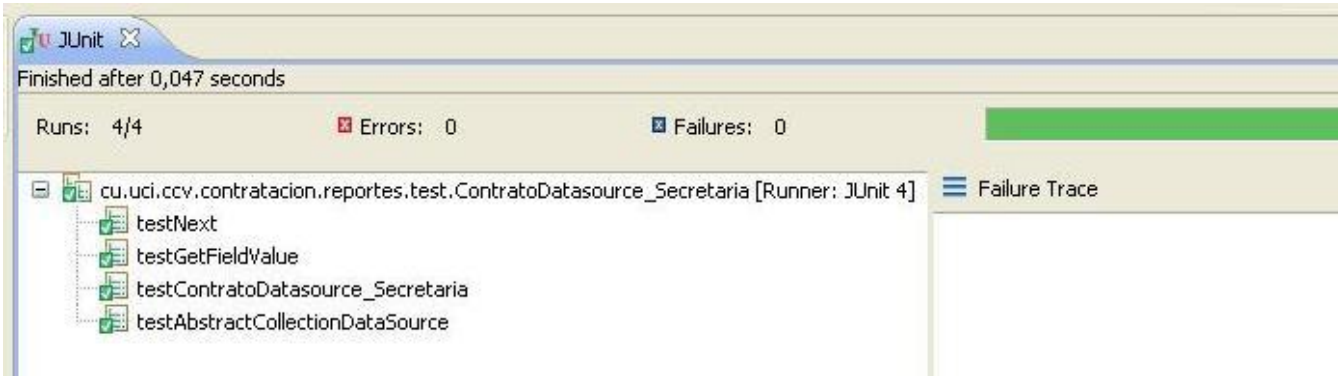


Figura A17: Prueba realizada a la clase ContratoDatasource_Secretaria.

Anexo 7 – Imágenes del diseño de las plantillas de los reportes en Ireport.

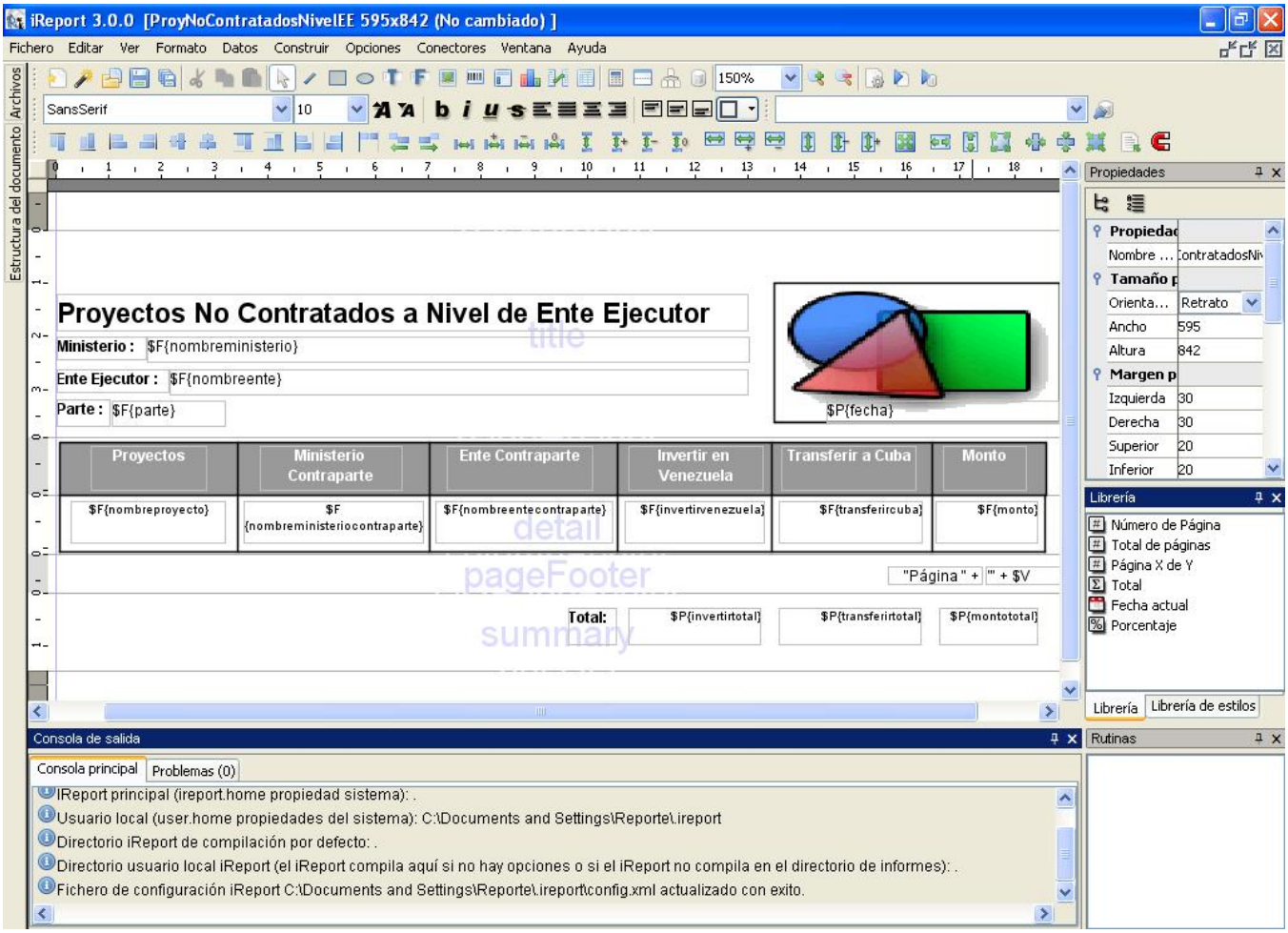


Figura A17: Diseño de la plantilla correspondiente al reporte Proyectos No Contratados a Nivel de Ente Ejecutor.

Anexo 8 – Representación del código fuente de los principales componentes.

Figura A19: Codigo fuente de la clase ProyNoContratadosNivelEEController.

```
package cu.uci.ccv.reportes.controladores;

import java.math.BigDecimal;


public class ProyNoContratadosNivelEEController implements Controller
{
     private JRDataSource datasource;
     private ReporteContratacionService reporteContratacionService;



        public ReporteContratacionService getReporteContratacionService() {

        public void setReporteContratacionService(


        public JRDataSource getDatasource() {

        public void setDatasource(JRDataSource datasource) {


        @Override
        public ModelAndView handleRequest(HttpServletRequest arg0,HttpServletResponse arg1) throws Exception
        {

         String dir = arg0.getRealPath("/WEB-INF/reportes/Imagenes/logo.png");
         int idProyecto =Integer.parseInt(arg0.getParameter("idProy"));
         SecurityContext sc = (SecurityContext) arg0.getSession().getAttribute("ACEGI_SECURITY_CONTEXT");

         DUsuario usuario = (DUsuario) sc.getAuthentication().getPrincipal();
```

```
 int idEnte=usuario.getNivel().getIdNivel();
 DParte Parte=usuario.getNivel().getParte();
 return new ModelAndView("pnocnivelee",getModel(dir,idProyecto,idEnte,Parte));

}

public Map getModel(String dir,int idProyecto,int idEnte,DParte Parte) throws BOException
{
    Map model = new HashMap();

    if(idProyecto==-1)
    {

        List<Object[]> ListaObjetos=reporteContratacionService.reporteFichasNoContratadas(-1,
        idEnte, -1, Parte);

        List<ReportePnoCNivelEE> lista_Entidad =new ArrayList<ReportePnoCNivelEE>();

        ReportePnoCNivelEE objeto=new ReportePnoCNivelEE();

        for(int i=0;i<ListaObjetos.size();i++)
        {

            objeto.setNombreente((String) ListaObjetos.get(i)[0]);
            objeto.setParte((String) ListaObjetos.get(i)[1]);
            objeto.setNombreministerio((String) ListaObjetos.get(i)[2]);

            objeto.setNombreproyecto((String) ListaObjetos.get(i)[3]);
            objeto.setNombreentecontraparte((String) ListaObjetos.get(i)[4]);
            objeto.setNombreministeriocontraparte((String) ListaObjetos.get(i)[5]);

         objeto.setInvertirvenezuela((String) ListaObjetos.get(i)[6]);
         objeto.setTransferircuba((String) ListaObjetos.get(i)[7]);
         objeto.setMonto((String) ListaObjetos.get(i)[8]);

         lista_Entidad.add(objeto);
         objeto=new ReportePnoCNivelEE();

    }


    Double invertir=0.0;
    Double transferir=0.0;
    Double monto=0.0;

    for(int i=0; i<lista_Entidad.size();i++)
    {
        invertir+= Double.parseDouble(lista_Entidad.get(i).getInvertirvenezuela());
        transferir+=Double.parseDouble(lista_Entidad.get(i).getTransferircuba());
        monto+=Double.parseDouble(lista_Entidad.get(i).getMonto());
    }

    BigDecimal invert=new BigDecimal(invertir);
    BigDecimal transf=new BigDecimal(transferir);
    BigDecimal mont=new BigDecimal(monto);

    String invertirtotal= BigDecimalFormat.convertBigDecimalToString(invert, 2);
    String transferirtotal=BigDecimalFormat.convertBigDecimalToString(transf, 2);
    String montototal=BigDecimalFormat.convertBigDecimalToString(mont, 2);

    for(int i=0; i<lista_Entidad.size();i++)
    {
```

```
            BigDecimal invertx=new BigDecimal(Double.parseDouble(lista_Entidad.get(i).
                    getInvertirvenezuela()));
            lista_Entidad.get(i).setInvertirvenezuela(BigDecimalFormat.convertBigDecimalToString
                    (invertx, 2));

            BigDecimal transfx=new BigDecimal(Double.parseDouble(lista_Entidad.get(i).
                    getTransferircuba()));
            lista_Entidad.get(i).setTransferircuba(BigDecimalFormat.convertBigDecimalToString
                    (transfx, 2));

            BigDecimal montox=new BigDecimal(Double.parseDouble(lista_Entidad.get(i).getMonto()));
            lista_Entidad.get(i).setMonto(BigDecimalFormat.convertBigDecimalToString(montox, 2));

        }

        datasource = CreateDatasource.createDatasourcePnoCnivelEE(lista_Entidad);
        model.put("LOGO_DIR", dir);
        model.put("fecha", DateFormater.getDateInSpanish());
        model.put("jrdatasource", datasource);
        model.put("invertirtotal", invertirtotal );
        model.put("transferirtotal", transferirtotal );
        model.put("montototal", montototal );

    }

    else
    {
        List<Object[]> ListaObjetos=reporteContratacionService.reporteFichasNoContratadas(-1,
                idEnte, idProyecto, Parte);
        List<ReportePnoCNivelEE> lista_Entidad =new ArrayList<ReportePnoCNivelEE>();
        ReportePnoCNivelEE objeto=new ReportePnoCNivelEE();


            objeto.setNombreente((String) ListaObjetos.get(0)[0]);
            objeto.setParte((String) ListaObjetos.get(0)[1]);
            objeto.setNombreministerio((String) ListaObjetos.get(0)[2]);
            objeto.setNombreproyecto((String) ListaObjetos.get(0)[3]);
            objeto.setNombreentecontraparte((String) ListaObjetos.get(0)[4]);
            objeto.setNombreministeriocontraparte((String) ListaObjetos.get(0)[5]);
            objeto.setInvertirvenezuela((String) ListaObjetos.get(0)[6]);
            objeto.setTransferircuba((String) ListaObjetos.get(0)[7]);
            objeto.setMonto((String) ListaObjetos.get(0)[8]);

            lista_Entidad.add(objeto);

            Double invertir=0.0;
            Double transferir=0.0;
            Double monto=0.0;

            invertir+= Double.parseDouble(lista_Entidad.get(0).getInvertirvenezuela());
            transferir+=Double.parseDouble(lista_Entidad.get(0).getTransferircuba());
            monto+=Double.parseDouble(lista_Entidad.get(0).getMonto());

            BigDecimal invert=new BigDecimal(invertir);
            BigDecimal transf=new BigDecimal(transferir);
            BigDecimal mont=new BigDecimal(monto);

            String invertirtotal= BigDecimalFormat.convertBigDecimalToString(invert, 2);
            String transferirtotal=BigDecimalFormat.convertBigDecimalToString(transf, 2);
```

```
            String montototal=BigDecimalFormat.convertBigDecimalToString(mont, 2);



            BigDecimal invertx=new BigDecimal(Double.parseDouble(lista_Entidad.get(0).
                    getInvertirvenezuela()));
            lista_Entidad.get(0).setInvertirvenezuela(BigDecimalFormat.convertBigDecimalToString(
                    invertx, 2));

            BigDecimal transfx=new BigDecimal(Double.parseDouble(lista_Entidad.get(0).
                    getTransferircuba()));
            lista_Entidad.get(0).setTransferircuba(BigDecimalFormat.convertBigDecimalToString(
                    transfx, 2));

            BigDecimal montox=new BigDecimal(Double.parseDouble(lista_Entidad.get(0).getMonto()));
            lista_Entidad.get(0).setMonto(BigDecimalFormat.convertBigDecimalToString(montox, 2));


        datasource = CreateDatasource.createDatasourcePnoCnivelEE(lista_Entidad);
        model.put("LOGO_DIR", dir);
        model.put("fecha", DateFormater.getDateInSpanish());
        model.put("jrdatasource", datasource);
        model.put("invertirtotal", invertirtotal );
        model.put("transferirtotal", transferirtotal );
        model.put("montototal", montototal );

    }


    return model;
}
```

Figura A 20: Codigo fuente de la clase CreateDatasource

```
package cu.uci.ccv.reportes.utiles;

import java.math.BigDecimal;

public class CreateDatasource
{
    static public JRDataSource createDatasourcePnoCnivelEE(List<ReportePnoCNivelEE> lista_B)

//*************************************************************************************************************

    static public JRDataSource createDatasourcePCnivelEE(List<ReportePCNivelEE> lista_A)


//*************************************************************************************************************

    static public JRDataSource createDatasourcePCNivelM(List<ReportePCNivelM> lista_A)
    {

        List<ReportePCNivelM> lista_B =new ArrayList<ReportePCNivelM>();
        String nombreente=lista_A.get(0).getNombreente();
        String nombrecontrato=lista_A.get(0).getNombrecontrato();
        boolean bandera1=false;
        boolean bandera2=false;
        int caso=0;


        lista_B.add(new ReportePCNivelM(
                lista_A.get(0).getNombreente(),
                lista_A.get(0).getNombreente1(),
                lista_A.get(0).getParte(),
                lista_A.get(0).getNombreministerio(),
                "0.0",
```

```
            "0.0",
            "1",
            lista_A.get(0).getNombrecontrato(),
            lista_A.get(0).getNombrecontrato1(),
            lista_A.get(0).getFechafirma(),
            lista_A.get(0).getNombreentecontraparte(),
            lista_A.get(0).getNombreministeriocontraparte(),
            lista_A.get(0).getInvertirvenezuela(),
            lista_A.get(0).getTransferircuba(),
            lista_A.get(0).getMonto(),
            lista_A.get(0).getProyectos()));

    for(int i=1;i<lista_A.size();i++)
    {


        String nombreente2=lista_A.get(i).getNombreente();
        if(nombreente.equals(nombreente2)) bandera1=true;

        String nombrecontrato2=lista_A.get(i).getNombrecontrato();
        if(nombrecontrato.equals(nombrecontrato2)) bandera2=true;

        if(bandera1==true && bandera2==true)
            caso=1;
        if(bandera1==true && bandera2==false)
            caso=2;
        if(bandera1==false)
            caso=3;

         switch (caso)
            {
            case 1:     lista_B.add(new ReportePCNivelM(


                        "",
                        lista_A.get(i).getNombreente1(),
                        lista_A.get(i).getParte(),
                        lista_A.get(i).getNombreministerio(),
                        "0.0",
                        "0.0",
                        "0.0",
                        "1",
                        "",
                        lista_A.get(i).getNombrecontrato1(),
                        lista_A.get(i).getFechafirma(),
                        lista_A.get(i).getNombreentecontraparte(),
                        lista_A.get(i).getNombreministeriocontraparte(),
                        lista_A.get(i).getInvertirvenezuela(),
                        lista_A.get(i).getTransferircuba(),
                        lista_A.get(i).getMonto(),
                        lista_A.get(i).getProyectos()));

                        nombreente=lista_A.get(i).getNombreente();
                        nombrecontrato=lista_A.get(i).getNombrecontrato();
                        bandera1=false;
                        bandera2=false;

    break;

        case 2:
                        nombreente=lista_A.get(i).getNombreente();

                        lista_B.add(new ReportePCNivelM(
                        "",
                        lista_A.get(i).getNombreente1(),
```

```
                lista_A.get(i).getParte(),
                lista_A.get(i).getNombreministerio(),
                "0.0",
                "0.0",
                "0.0",
                "1",
                lista_A.get(i).getNombrecontrato(),
                lista_A.get(i).getNombrecontrato1(),
                lista_A.get(i).getFechafirma(),
                lista_A.get(i).getNombreentecontraparte(),
                lista_A.get(i).getNombreministeriocontraparte(),
                lista_A.get(i).getInvertirvenezuela(),
                lista_A.get(i).getTransferircuba(),
                lista_A.get(i).getMonto(),
                lista_A.get(i).getProyectos()));

                nombreente=lista_A.get(i).getNombreente();
                nombrecontrato=lista_A.get(i).getNombrecontrato();
                bandera1=false;
                bandera2=false;
    break;

      case 3:
          lista_B.add(new ReportePCNivelM(

                  "Ente Ejecutor",
                  "Ente Ejecutor",
                  lista_A.get(i).getParte(),
                  "Ministerio",
                  "0.0",
                  "0.0",
                  "0.0",



                  "",
                  "Nombre del Contrato",
                  "Nombre del Contrato",
                  "Fecha de Firma",
                  "Ente Contraparte",
                  "Ministerio Contraparte",
                  "0.0",
                  "0.0",
                  "0.0",
                  "Proyectos"));

          lista_B.add(new ReportePCNivelM(
          lista_A.get(i).getNombreente(),
          lista_A.get(i).getNombreente1(),
          lista_A.get(i).getParte(),
          lista_A.get(i).getNombreministerio(),
          "0.0",
          "0.0",
          "0.0",
          "1",
          lista_A.get(i).getNombrecontrato(),
          lista_A.get(i).getNombrecontrato1(),
          lista_A.get(i).getFechafirma(),
          lista_A.get(i).getNombreentecontraparte(),
          lista_A.get(i).getNombreministeriocontraparte(),
          lista_A.get(i).getInvertirvenezuela(),
          lista_A.get(i).getTransferircuba(),
          lista_A.get(i).getMonto(),
          lista_A.get(i).getProyectos()));

          nombreente=lista_A.get(i).getNombreente();
          nombrecontrato=lista_A.get(i).getNombrecontrato();
```

```
                bandera1=false;
                bandera2=false;
  break;
        }
    }

    String nombretemp="";
    Double sumamonto=0.0;
    Double sumainvertir=0.0;
    Double sumatransferir=0.0;
    int pos1=0;
    boolean bandera3=true;

for(int i=0;i<lista_B.size();i=pos1)
{
   if(bandera3==true)
   {
     nombretemp="";

     while(nombretemp=="")
       {
         sumamonto+=Double.parseDouble(lista_B.get(pos1).getMonto());
         sumainvertir+=Double.parseDouble(lista_B.get(pos1).getInvertirvenezuela());
         sumatransferir+=Double.parseDouble(lista_B.get(pos1).getTransferircuba());
         pos1++;

        if(pos1<lista_B.size())
        {
        nombretemp=lista_B.get(pos1).getNombreente();
        }

        if(pos1==(lista_B.size()))
```

```
            {
                nombretemp="stop";
                bandera3=false;
            }
           }

           lista_B.get((pos1)-1).setTotalmonto(sumamonto.toString());
           lista_B.get((pos1)-1).setTotalinvertir(sumainvertir.toString());
           lista_B.get((pos1)-1).setTotaltransferir(sumatransferir.toString());

           sumamonto=0.0;
           sumainvertir=0.0;
           sumatransferir=0.0;
       }

       if(bandera3==false)
       {
        pos1++;
       }
   }

   Double invertir=0.0;
   Double transferir=0.0;
   Double monto=0.0;
   Double invertirtotalente=0.0;
   Double transferirtotalente=0.0;
   Double montototalente=0.0;

   for(int i=0; i<lista_B.size();i++)
   {
       invertir= Double.parseDouble(lista_B.get(i).getInvertirvenezuela());
       transferir=Double.parseDouble(lista_B.get(i).getTransferircuba());
```

```
    monto=Double.parseDouble(lista_B.get(i).getMonto());

    invertirtotalente=Double.parseDouble(lista_B.get(i).getTotalinvertir());
    transferirtotalente=Double.parseDouble(lista_B.get(i).getTotaltransferir());
    montototalente=Double.parseDouble(lista_B.get(i).getTotalmonto());

    BigDecimal invert=new BigDecimal(invertir);
    BigDecimal transf=new BigDecimal(transferir);
    BigDecimal mont=new BigDecimal(monto);

    BigDecimal inverttotal=new BigDecimal(invertirtotalente);
    BigDecimal transftotal=new BigDecimal(transferirtotalente);
    BigDecimal monttotal=new BigDecimal(montototalente);


    String invertirtemp= BigDecimalFormat.convertBigDecimalToString(invert, 2);
    String transferirtemp=BigDecimalFormat.convertBigDecimalToString(transf, 2);
    String montotemp=BigDecimalFormat.convertBigDecimalToString(mont, 2);

    String invertirtotalentetemp= BigDecimalFormat.convertBigDecimalToString(inverttotal, 2);
    String transferirtotalentetemp=BigDecimalFormat.convertBigDecimalToString(transftotal, 2);
    String montototalentetemp=BigDecimalFormat.convertBigDecimalToString(monttotal, 2);

    lista_B.get(i).setInvertirvenezuela(invertirtemp);
    lista_B.get(i).setTransferircuba(transferirtemp);
    lista_B.get(i).setMonto(montotemp);

    lista_B.get(i).setTotalinvertir(invertirtotalentetemp);
    lista_B.get(i).setTotaltransferir(transferirtotalentetemp);
    lista_B.get(i).setTotalmonto(montototalentetemp);

}
```

```
for(int i=0; i<lista_B.size();i++)
{
    if(lista_B.get(i).getTotalmonto().equals("0,00"))
    {
        lista_B.get(i).setTotalmonto("");
    }
    if(lista_B.get(i).getInvertirvenezuela().equals("0,00"))
    {
        lista_B.get(i).setInvertirvenezuela("");
    }
    if(lista_B.get(i).getTransferircuba().equals("0,00"))
    {
        lista_B.get(i).setTransferircuba("");
    }
    if(lista_B.get(i).getMonto().equals("0,00"))
    {
        lista_B.get(i).setMonto("");
    }
}

 String nombreente2="";
 String nombreente3="Ente Ejecutor";

 for(int i=0; i<lista_B.size()-1;i++)
    {
     if(lista_B.get(i+1).getNombreente().equals(nombreente3))
     {
        if(lista_B.get(i).getTotalmonto().equals(nombreente2))
            lista_B.get(i).setTotalmonto("0,00");
     }
    }



       JRDataSource ds = new ContratoDataSource(lista_B);
       return ds;
  }


//*************************************************************************************************************

   public static JRDataSource createDatasourcePnoCNivelM(List<ReportePnoCNivelM> lista_A)

//*************************************************************************************************************

   static public JRDataSource createDatasourcePCNivelS(List<ReportePCNivelS> lista_A)

//*************************************************************************************************************

   public static JRDataSource createDatasourcePnoCNivelS(List<ReportePnoCNivelS> lista_A)

//*************************************************************************************************************
```

Anexo 9: Acta de Aceptación del Módulo Contratación.

albet INGENIERÍA Y SISTEMAS | PROYECTOS

ACTA DE ACEPTACIÓN

Producto: Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela.

Categoría: Aceptación CU Módulo de Contratación.

Fecha de la conciliación: 3/07/2008.

Involucrados en el proceso:

Por la parte del Cliente (MENPET): Sandra Cortés
Por la parte del Suministrador (ALBET): Ing. Nahuel Massón

Observaciones del proceso:

Por acuerdo entre las partes involucradas en el proceso se Acepta el Documento CU del Módulo de Contratación de la versión v.2.0 del Sistema de Gestión para Seguimiento de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela, con fecha 3 de julio de 2008.

Para que conste la aceptación de la descripción de los CU y requerimientos del Módulo de Contratación, dando fé al acuerdo, firman la presente los principales representantes de las Partes.

Sandra Cortés
Representante MENPET

Ing. Nahuel Massón
Representante ALBET

Referencia: CV-SW-CC-011

ALBET, S.A.
Centro de Negocios Miramar. Edificio Barcelona, Oficina 322. Avenida 5ta e/ 76 y 78. Miramar. Playa, Ciudad Habana, Cuba
Tel/Fax: +53 (7) 837 2407
E-mail: albet@albet.cu

Anexo 10: Acta de finiquito.

Entre la República Bolivariana de Venezuela, actuando por órgano del Ministerio del Poder Popular para la Energía y Petróleo de la República Bolivariana de Venezuela, representado en este acto por el ciudadano **Ammar Jabour**, venezolano, mayor de edad, de este domicilio, titular de la Cédula de Identidad No. 9962729, quien actúa en su condición de Coordinador General del Convenio Integral de Cooperación Cuba-Venezuela, suficientemente facultado para este acto y debidamente designado para ejercer tal condición conforme a lo establecido en **CONTRATO E08-001-000 SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA,** que en lo sucesivo se denominará la "**Parte Venezolana**", por una parte; y por la otra, la sociedad mercantil **ALBET, Ingeniería y Sistemas, S.A.**, sociedad mercantil cubana constituida mediante Escritura 271 de fecha 7 de Noviembre de 2005, autorizada por la Notario Lic. Isabel Cristina Martínez Alfonso con sede en Notaría Especial del Ministerio de Justicia de Ciudad de la Habana, inscrita en el Registro Mercantil de esta ciudad con fecha 14 de Noviembre del año 2005, al Tomo XVIII, Folio 120, Hoja 11, Sección SM, con N° de inscripción 1 con domicilio social en Centro de Negocios de Miramar, Edificio Barcelona, Oficina 322, Avenida 5ta entre 76 y 78, Miramar, Municipio Playa, Ciudad de La Habana, República de Cuba, representada en este acto por el ciudadano cubano **Ibrahim Nápoles Albanés**, mayor de edad, portador de carné de identidad N° 62032504808 en su condición de Coordinador General, suficientemente facultado para este acto según lo dispuesto en **CONTRATO E08-001-000 SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA**, que en lo sucesivo se denominará **"Parte Cubana"**, al tenor de las siguientes declaraciones y cláusulas:

CONSIDERANDO

Primero: Consta de documento privado suscrito en fecha 18 de marzo de 2008, que ambas partes celebraron un **CONTRATO DE SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA,** el cual fue celebrado al Amparo del Convenio Integral de Cooperación, suscrito el 30 de octubre del 2000 y su Addendum; de la Declaración Conjunta y el Acuerdo suscrito entre ambas Repúblicas, para la aplicación de la Alternativa Bolivariana para las Américas, firmados en diciembre del 2004 y de los Acuerdos y las Condiciones Generales firmados en el Acta de la Reunión de la Comisión Mixta Cuba-Venezuela, celebrada en La Habana, Cuba, cuyos contenidos se dan aquí enteramente por reproducidos.

Segundo: Consta que la **Parte Cubana** ha cumplido a entera satisfacción de la **Parte Venezolana** con el objeto, alcance y actividades previstas en el **Contrato** ya mencionado, así como con sus Anexos y Suplementos, cumpliendo por tanto con todos sus deberes y obligaciones por lo que se considera que la Solución ha sido implementada en las condiciones previstas y bajos los requisitos y especificaciones técnicas pactadas entres **Las Partes**.

Tercero: Consta que la **Parte Cubana** ha hecho entrega del **Informe Técnico Final**, de conjunto a toda la documentación que avala y soporta lo descrito en el Segundo de los **CONSIDERANDOS**, debidamente firmada y aceptada por los especialistas de la **Parte Venezolana**.

Cuarto: Consta que la **Parte Venezolana** ha pagado la totalidad de **UN MILLÓN CUATROCIENTOS DIECISIETE MIL OCHOCIENTOS VEINTIOCHO DOLARES DE LOS ESTADOS UNIDOS DE AMERICA CON VEINTIUN CENTAVOS** mediante la forma de pago prevista en el Contrato ya mencionado, a entera satisfacción de la Parte Cubana, quedando a la firma de la presente **Acta**.

LAS PARTES CONVIENEN:

Primero: Dar por terminada la relación contractual derivada del **CONTRATO DE SOLUCIÓN TECNOLÓGICA INTEGRAL PARA EL DESARROLLO DEL SISTEMA DE GESTIÓN PARA SEGUIMIENTO DE LOS PROYECTOS DEL CONVENIO INTEGRAL DE COOPERACIÓN CUBA VENEZUELA** de fecha 16 de marzo de 2008.

Segundo: Las Partes se otorgan el finiquito más amplio que en derecho proceda, no reservándose acción o derecho que ejercitar con posterioridad.

Leída que les fue la presenta Acta, las partes se ratificaron en su contenido, para constancia firman en cuatro (4) ejemplares de igual tenor en la Ciudad de Caracas el día 17 del mes de diciembre del 2008.

Por la PARTE VENEZOLANA	Por la PARTE CUBANA
Ammar Jabour Coordinador General	**Ibrahim Nápoles Albanés** Coordinador General

GLOSARIO

Sistema de Información: es el sistema de personas, registros de datos y actividades que procesa los datos y la información en cierta organización, incluyendo manuales de procesos o procesos automatizados.

Software colaborativo: Un sistema colaborativo es un conjunto de hardware y de herramientas de software que soportan el trabajo en colaboración de equipos de personas, que se desarrolla sobre una red de telecomunicaciones. Se trata de herramientas informáticas que son especialmente diseñadas para ayudar a los usuarios a trabajar en colaboración de forma más eficaz. Existen diferentes tipos de software colaborativos como son: software para trabajo en grupo, software para gestión de documentos y software para gestión de proyectos.

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

EJB: Resulta una de las tecnologías más complejas y avanzadas de Java hoy en día, si no la más (hasta tal punto que resulta muy difícil resumirla). Su objetivo es facilitar a las aplicaciones Java un framework que permita gestionar la persistencia de los objetos de su modelo de una manera transparente y efectiva, así como otorgar grandes facilidades para la distribución de aplicaciones y la gestión de concurrencia.

JDBC: API de la plataforma Java para el acceso a sistemas gestores de base de datos.

JNDI (Java Naming and Directory Interface): Java nos ofrece JNDI como ayuda en entornos de red, principalmente entornos distribuidos, para localizar objetos e información en estas redes mediante lo que se denomina un *servicio de directorio*.

JNI: Es un framework de programación que permite que un programa escrito en Java ejecutado en la máquina virtual de java (JVM) pueda interactuar con programas escritos en otros lenguajes.

JSP: Las JSP surgieron como manera de que se pudiesen crear páginas web dinámicas mediante Java pero evitando que al programarlas el proceso Java predominase sobre la definición del aspecto

gráfico (como ocurre en los servlets), y haciendo que se definiesen estas páginas dinámicas como páginas HTML normales en las que se embebía código Java.

JSF: Es un framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones J2EE.

Apache 2.0 License: Licencia de software libre creada por la Apache Software Foundation.

Hibernate: Herramienta de Mapeo objeto-relacional para la plataforma Java, que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

LGPL: Es una licencia de software creada por la Free Software Foundation.

Mozilla Public: License(Licencia Pública de Mozilla) licencia de código abierto y software libre.

PNG: Imágenes de formato de gráficas portables de red.

SVG: Imágenes en formato de gráficas vectoriales escalables

Apache Ant: es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción

NetBeans: Es una plataforma para el desarrollo de aplicaciones de escritorio usando Java.

J2RE: Java Runtime Environment (entorno en tiempo de ejecución Java) y se corresponde con un conjunto de utilidades que permite la ejecución de programas java sobre todas las plataformas soportadas.

Servlets: Es una Api que define una valiosa tecnología que permite la creación de páginas web dinámicas como resultado de la ejecución de un proceso programado en Java (el servlet). Fue en su tiempo una tecnología pionera, y formó la base de lo que hoy en día, en unión con otras tecnologías, es la plataforma para aplicaciones web en Java.