Universidad de las Ciencias Informáticas

Facultad 15

Diseño, implementación y validación del módulo Control de armamento, equipos y medios de seguridad del Sistema de Gestión Penitenciaria.

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

Autores: **Nelson Rivera Carbonell**

Cristian Fernández López

Tutor: **Ing. Madelín Haro Pérez**

Ciudad de La Habana, mayo de 2010

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de este trabajo y autorizamos a la Facultad 15 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2010.

_____ _____

Nelson Rivera Carbonell **Cristian Fernández López**

_____

Ing. Madelín Haro Pérez

DATOS DE CONTACTO

Síntesis del Tutor Ing. Madelín Haro Pérez:

- Graduada en 2002 de la carrera Ingeniería Informática en el ISPJAE.
- Ha trabajado en la Empresa de Servicios Informáticos (ESI) de Cienfuegos y a partir de enero de 2003 en la UCI.
- Ha sido tutora de tesis durante los 6 primeros cursos de la UCI y cotutor o consultora. Los temas trabajados han estado relacionados con el ciclo de vida del software (análisis, base de datos, arquitectura, prototipos, requisitos, mantenimiento y soporte), Gestión de Proyecto, Seguridad de sistemas informáticos y Linux.
- Ha participado en eventos como el Fórum de Ciencia y Técnica, UCIENCIA y Universidad 2008 a nivel de base.
- Se ha vinculado a los proyectos productivos en el rol de Jefe de Proyecto, Analista y Jefe de Capacitación.
- Ha impartido cursos de postgrado a funcionarios de empresas, profesores y trabajadores de la UCI y a los Directores de los IPI.
- Ha impartido cursos de capacitación en dos proyectos de producción en el extranjero.
- Ha recibido un curso de postgrado en el instituto TATA, en la India, recibiendo el certificado internacional de habilidades.
- Es Jefe de Departamento Docente Central de Práctica Profesional y su categoría docente actual es Asistente.

AGRADECIMIENTOS

Queremos agradecer a todas las personas que han contribuido para lograr que este trabajo llegue a un feliz término, en especial a nuestra tutora por las horas dedicadas y el esfuerzo invertido en guiarnos y a Isabel, por supuesto, por permitírselo.

Llegue nuestro más sincero agradecimiento también, a los que aportaron en nuestra formación integral: a los nuevos del SIGEP y a nuestros compañeros de aula y a los profesores de todos estos los años.

A la Revolución y a nuestra Universidad, por brindarnos las opciones, el espacio y las condiciones necesarias, para realizar nuestra meta.

DEDICATORIA

A mi madre por enseñarme a vivir.

A mi abuela por permitirme ser un niño feliz.

A mi papá por darme los conceptos claros de sacrificio y disciplina.

A mi hermano por ser mi ángel guardián.

A todos mis verdaderos amigos.

Cristian Fernández López

Este trabajo de diploma y mis éxitos en la universidad, se lo dedico en primer lugar a mi familia, en especial mi padre, mi madre y mi hermano, que ha sido mi apoyo y mi patrón de conducta durante estos 5 años y en mi vida en general. Se lo dedico, además, a todos mis amigos, los amigos viejos de escuelas anteriores con los cuales todavía tengo sólidas relaciones de amistades y los buenos amigos que he hecho en la universidad, todos ellos han aportado su grano de arroz, al hacer posible y llevadero mi transcurso en esta universidad. A todos se lo dedico.

Nelson Rivera Carbonell

RESUMEN

Para contribuir a la solución de los problemas por los que atraviesa el sistema penitenciario venezolano, la Universidad de las Ciencias Informáticas fue contratada para desarrollar el Sistema de Gestión Penitenciaria (SIGEP). Este sistema tiene como misión automatizar los procesos del sistema penitenciario venezolano y garantizar la gestión rápida y segura de toda la información correspondiente a los internos y funcionarios. En su versión 2.0 se le incorporan nuevas funcionalidades al software SIGEP, entre ellas, las funcionalidades que permiten controlar el inventario del armamento, municiones y los medios técnicos asignados a los custodios para el cumplimiento de sus funciones dentro de los establecimientos penitenciarios. Estas funcionalidades serán agrupadas en un módulo dentro del área Seguridad y Custodia. Para esto se diseñó e implementó la solución del módulo Control de armamentos, equipos y medios de seguridad, cumpliendo con los requisitos, teniendo en cuenta el uso de los patrones de diseño y aplicando los lineamientos definidos en la arquitectura del sistema.

Como resultado de todas las actividades realizadas se integró el módulo como componente ejecutable al SIGEP 2.0 y se validó en un ambiente de trabajo real en Venezuela.

PALABRAS CLAVE

SIGEP, Arquitectura, Diseño, Implementación, Control, Armamentos, Equipos, Medios de Seguridad.

ÍNDICE

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 4

1.1 Introducción 4
1.2 Sistemas Penitenciarios 4
1.3 Sistemas penitenciarios y sistemas informáticos 5
1.3.1 Análisis general 5
1.4 Sistema Penitenciario de Venezuela 6
1.4.1 Subsistema Seguridad y Custodia 6
1.4.2 Módulo Control de armamentos, medios y equipo de seguridad 7
1.5 Tecnologías y herramientas utilizadas en el desarrollo 8
1.5.1 Programación orientada a objetos 9
1.5.2 Metodología de Desarrollo 9
1.5.4 Patrones de Diseño 12
1.5.5 Herramienta de modelado 13
1.5.6 Plataforma de desarrollo: Java 2 Enterprise Edition (J2EE) 13
1.5.7 Frameworks 14
1.5.8 Gestor de Base de Datos 17
1.5.9 Entorno integrado de desarrollo (IDE) 17
1.6 Arquitectura del módulo Control de armamento, equipos y medios de seguridad 18

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA 25

2.1 Introducción 25
2.2 Análisis de las funcionalidades 25
2.3 Diseño del módulo Control de armamento, equipos y medios de seguridad del SIGEP 26
2.3.1 Diseño de las entidades del dominio 26
.2.3.2 Diseño del modelo de datos 27
2.3.3 Diseño de la capa de acceso a datos 29
2.3.4 Diseño de la capa de negocio. 30
2.3.5 Diseño de la capa de presentación. 32
2.4 Implementación módulo Control de armamento, equipos y medios de seguridad del SIGEP 33
2.4.1 Implementación de las entidades del dominio 33

2.4.2 Implementación de la capa de acceso a datos 33
2.4.3 Implementación de la capa de negocio. 35
2.4.3.1 Transacciones 37
2.4.4 Implementación de la capa de presentación 38
2.4.5 Implementación de los controladores 38
2.4.6 Implementación de las páginas JSP 39
2.4.7 Implementación de la lógica en el cliente 39
2.4.8 Implementación de la funcionalidad Registrar Entrega 39
2.4.9 Implementación de la funcionalidad Registrar Devolución 42
CAPÍTULO 3: VALIDACIÓN DEL SOFTWARE 45
3.1 Introducción 45
3.2 Pruebas de aceptación parcial del cliente (validaciones) 45
3.2.1 Objetivos generales de las pruebas 45
3.2.2 Fases para la realización de las pruebas 46
3.2.3 Pruebas funcionales 46
3.2.4 Participantes en la prueba de aceptación parcial 47
3.2.5 Flujo de trabajo 47
3.2.6 Descripción del flujo de trabajo 47
3.2.7 Acción de recolección y valoración de no conformidades y solicitudes de cambio 48
3.3 Pruebas de aceptación final del cliente 49
3.3.1 Objetivos generales de las pruebas 49
3.3.2 Fases para la realización de las pruebas 50
3.3.3 Participantes en la prueba piloto por la parte cubana 50
3.3.4 Flujo de trabajo 51
3.3.5 Descripción del flujo de trabajo 51
3.3.6 Acción de recolección y valoración de no conformidades y solicitudes de cambio 52
CONCLUSIONES 54
RECOMENDACIONES 55
BIBLIOGRAFÍA 56
GLOSARIO 57
ANEXOS 58

2.1 Modelo de Dominio 58
2.2 Diagrama Entidad Relación de la base de datos. 59
2.3 Diseño de la Capa de Presentación del módulo Control de armamento, equipos y medios de seguridad. 60
2.3.1 Gestión de armamentos, equipos y medios de seguridad. 60
2.3.2 Gestión de bajas 61
2.3.3 Gestión de entregas 62
2.4 Ejemplo de consultas usando el API Criteria y el lenguaje HQL de Hibernate (fragmentos del EntregaDAOImpl). 64
2.5 Implementación del método persistirEntrega 66
2.6 Implementación de la clase RegistrarDevolucionSimpleFormController 68

ÍNDICE DE FIGURAS

Figura 1.1 Sistemas y Subsistemas del SIGEP 6
Figura 1.2 Módulos del subsistema Seguridad y Custodia 7
Figura 1.3 Funcionalidades del modulo Control de armamentos, equipos y medios de seguridad 8
Figura 1.4 Flujo de trabajo de la metodología RUP 10
Figura 1.5 Modelos de capas de un módulo del SIGEP 19
Figura 1.6 Estructura de paquetes de un módulo del SIGEP 24
Figura 2.1 Diagrama de clases persistentes del módulo Control de armamentos, equipos y medios de seguridad 27
Figura 2.2 Diagrama entidad relación del módulo Control de armamentos, equipos y medios de seguridad 28
Figura 2.3 Estructura de la capa de acceso a datos 30
Figura 2.4 Estructura de la capa de negocio 31
Figura 2.5 Configuración del fichero Baja.hbm.xml 34
Figura 2.6 Configuración de la clase BajaDAO en el fichero sigep-seguridadcustodia-armamento.dataaccess-context.xml 35
Figura 2.7 Configuración del manager ElementoSeguridadProteccion en el fichero sigep-seguridadcustodia-armamento-bussiness-context.xml 36
Figura 2.8 Configuración de la fachada en el fichero sigep-seguridadcustodia-armamento-bussiness-context.xml 36
Figura 2.9 Aplicación de transacciones al objeto ElementoSeguridadProteccionImpl 38
Figura 2.10 Configuración del controlador RegistrarEntregaMultiActionController 41

Figura 2.11 Método ModelView del controlador RegistrarEntregaMultiActionController 41
Figura 2.12 Diagrama de Secuencia de una Nueva Entrega 42
Figura 2.13 Pantalla de Registrar Devolución..... 44
Figura 2.14 Pantalla Devolver Elemento Entregado 44

INTRODUCCIÓN

El sistema penitenciario de la República Democrática de Venezuela, presenta serias deficiencias que abarcan aspectos de infraestructura, recursos humanos, gestión, comunicación y organización. Entre sus atributos esenciales se encuentran desorden, ingobernabilidad, tolerancia, concesiones, condiciones infrahumanas de vida, violencia, armas de fuego, ocio, droga, retardo judicial, carencia de mecanismos de control y comunicación insuficiente entre los diferentes niveles.

En el marco de la cooperación bilateral entre Cuba y Venezuela, surge como acuerdo desarrollar un sistema informático integral que permita gestionar las actividades penitenciarias, como solución a las deficiencias que las mismas presentan. Este sistema informático abarca las áreas de Control Penal, Clasificación y Tratamiento, Salud Integral, Seguridad y Custodia, Control Logístico, Administración de la Dirección General y la Gestión de Unidades de Apoyo, este software es el Sistema de Gestión Penitenciaria (SIGEP).

Un estudio realizado por los analistas del proyecto SIGEP a los procesos claves de la institución mostraron las siguientes deficiencias en los procesos relacionados con el control de los medios destinados a la custodia de los reclusos: La información generada en estos procesos se realizan totalmente en papel y de forma manual, la documentación que se genera ocupa un gran volumen por la cantidad de datos que se registra en cada hecho vinculado con las actividades de custodia a los reclusos, provocando que los procesos de generar y consultar este volumen de información sea un proceso tedioso, largo y lento, lo cual dificulta la tarea de llevar un control exacto de los recursos asignados para la custodia de los reclusos. Esta situación en los casos más graves conlleva a la pérdida de armamentos dentro del establecimiento. Otra deficiencia presente en estos procesos en el sistema penitenciario venezolano es que no hay formatos estándares para la información que genera cada establecimiento penitenciario.

En el tiempo transcurrido a partir de la primera versión del SIGEP, el equipo de desarrollo y el cliente profundizaron sus conocimientos del funcionamiento de la institución y de sus necesidades de informatización. Esto permitió un perfeccionamiento de la estructura del sistema y su alcance, los cuales inicialmente estuvieron basados en un estudio preliminar, realizado en un momento en que las áreas de la institución tenían un alto nivel de indefinición. Partiendo de estos elementos y de las deficiencias encontradas en los procesos relacionados con el control de los medios destinados a la custodia de los reclusos; se toma el acuerdo entre las partes venezolanas interesadas y la dirección del proyecto, integrar a la versión 2.0 del SIGEP, dentro del subsistema Seguridad y Custodia, un proceso que permitirá controlar el inventario del

armamento, equipos y los medios técnicos asignados a los custodios dentro de los establecimientos penitenciarios.

Dentro del marco de desarrollo del SIGEP tomamos como problema que *el Sistema de Gestión Penitenciaria (SIGEP), tiene definido el modelo de análisis del módulo de Control de armamento, equipos y medios de seguridad y necesita continuidad de su ciclo de desarrollo.*

Como objetivo principal de nuestro trabajo de diploma:

Diseñar, implementar y validar el módulo de Control de armamento, equipos y medios de seguridad.

Los resultados esperados son:

- Modelo de diseño validado del módulo Control de armamento, equipos y medios de seguridad del proyecto SIGEP.
- Modelo de Implementación validado del módulo Control de armamento, equipos y medios de seguridad del proyecto SIGEP.
- Módulo Control de armamento, equipos y medios de seguridad, integrado al SIGEP.
- Resultados obtenidos por el módulo Control de armamento, equipos y medios de seguridad durante las pruebas piloto.

Nuestro objeto de estudio se enmarca en *el Proceso de desarrollo del subsistema Seguridad y Custodia perteneciente al Sistema Penitenciario venezolano* y el campo de acción, *el Modelos de análisis e implementación del Módulo de Control de armamento, equipos y medios de seguridad.*

Este trabajo de diploma se basa en la labor realizada durante las tareas de diseño e implementación de los autores en el proyecto SIGEP. Teniendo en cuenta los objetivos y actividades a realizar partiendo de los roles desempeñados, nos planteamos las siguientes tareas:

- Estudio de las tecnologías y arquitectura a usar en el desarrollo.
- Estudio de los requisitos de software y del Modelo de Negocio correspondientes al módulo Control de armamento, equipos y medios de seguridad.
- Diseño de la solución de software para los requisitos relacionados con el módulo Control de armamento, equipos y medios de seguridad.*

- Implementación del diseño realizado al módulo Control de armamento, equipos y medios de seguridad. Pruebas unitarias y de integración.
- Integración de Control de armamento, equipos y medios de seguridad al SIGEP.
- Análisis de los resultados obtenidos por el módulo Control de armamento, equipos y medios de seguridad durante las pruebas de aceptación.

El documento consta de 3 Capítulos y Anexos

Capitulo 1 Fundamentación Teórica

Capitulo 2 Diseño e Implementación del módulo Control de armamento, equipos y medios de seguridad

Capitulo 3 Validación y pruebas pilotos del módulo Control de armamento, equipos y medios de seguridad

Los anexos muestran documentación generada como resultado de los proceso de diseño, implementación y validación del módulo Control de armamento, equipos y medios de seguridad.

El trabajo de diploma desarrollado, forma parte del informe técnico del proyecto SIGEP.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el capítulo inicial se abordan los términos del Sistema Penitenciario venezolano que están vinculados con el desarrollo del trabajo de diploma. Se realiza un análisis de software para sistemas penitenciarios y las tecnologías, herramientas y metodología de desarrollo utilizadas en las etapas de diseño e implementación del módulo de control de armamentos, medio y equipos de seguridad del Sistema Penitenciario venezolano, dentro del Sistema de Gestión Penitenciario (SIGEP).

1.2 Sistemas Penitenciarios

Sistema Penitenciario se establece como el conjunto de normas, procedimientos y dependencias dispuestas por el Estado para la ejecución del régimen penitenciario entre los que se encuentran además los principios, programas, recursos humanos, dependencias e infraestructura que se encuentran relacionadas y destinadas a este régimen.

En el diccionario de Ciencias Jurídicas, Políticas y Sociales creado por Manuel Ossorio, se enlaza el concepto de sistema penitenciario con régimen de penitenciario, definiendo éste régimen como: “conjunto de normas legislativas administrativas encaminadas a determinar los diferentes sistemas adoptados para que *penados*[1] cumplan sus *penas*[2]. Se encamina obtener la mayor eficacia en la custodia o en él la readaptación social de los delincuentes. Esos regímenes son múltiples, varían a través de los tiempos y van desde el aislamiento absoluto y tratamiento rígido hasta el sistema de puerta abierta con libertad vigilada. Entre ambos extremos existe una gran gradación.” (OSSORIO).

[1] *Penado*: “En general, el condenado por sentencia firme al cumplimiento de una pena”. (OSSORIO)

[2] *Pena*: “Castigo impuesto por autoridad legítima, especialmente de índole judicial, a quien ha cometido un delito o falta”. (OSSORIO)

1.3 Sistemas penitenciarios y sistemas informáticos

A nivel mundial existen sistemas informáticos que sirven de apoyo a la gestión de los procesos en los centros penitenciarios. La mayoría se han desarrollado en conjunto con los diferentes organismos nacionales e internacionales con mayor experiencia en el tema. Esto ha contribuido a erradicar las deficiencias en la estandarización de procesos penitenciarios que se presentaban de forma común.

Para el desarrollo del SIGEP se realizó un análisis de diferentes sistemas, sus entornos de desarrollo y procesos a automatizar. Citamos algunos sistemas penitenciarios analizados.

- SACORE, SIGEP Cuba
- Sistema Penitenciario del Gobierno de Panamá
- Sistema Penitenciario Bonaerense (Buenos Aires, Argentina)
- Sistema de Gestión Penitenciaria SIGPEN (Ecuador)
- Establecimiento Penitenciario Virtual en Red (España)

1.3.1 Análisis general

Como característica de mayor importancia para este trabajo de diploma se encontró que en los sistemas estudiados no se tiene en cuenta un proceso para el control de armamentos, equipos y medios de seguridad, se centran en los procesos primarios de los centros penitenciarios.

Otras características importantes de estos sistemas son la falta de definición de los procesos penitenciarios y la mala estructura de las infraestructuras en las cuales son implantados. No presentan una conectividad con otros centros penitenciarios imposibilitando la centralización e integridad de la información generada en el trámite de la información. Están sujetos a la situación legislativa y judicial imperante en un país, lo cual lo convierte en sistemas no portables para otros sistemas penitenciarios.

En los sistemas penitenciarios, donde están implantados los sistemas antes mencionados, se logra automatizar los procesos tratados y funcionan como herramientas de trabajo.

1.4 Sistema Penitenciario de Venezuela

En la República Bolivariana de Venezuela, donde el sistema penitenciario actual posee fundamentos similares a los sistemas penitenciarios de Latinoamérica, existen múltiples estrategias para recoger la información de los procesos penitenciarios en formato digital, ya sea en hojas de cálculo o documentos de texto, con el fin de facilitar el trabajo de los funcionarios se seguridad y custodia, sin que existan formatos estándares para ello. Por ende, no se tiene un registro de un sistema informático precedente que pueda apoyar a los procesos penitenciarios.

El desarrollo de la primera versión del SIGEP en colaboración con instituciones especializadas cubanas se plantea la resolución a muchas de estas problemáticas. El resultado alcanzado de este convenio fue un sistema informático centralizado que estandarizó e implementó los procesos fundamentales en el sistema penitenciario venezolano y se convirtió en una herramienta de trabajo para funcionarios en los centros penitenciarios.

Partiendo de la estructura inicial del sistema de gestión penitenciario, en el **Figura 1.1** se muestran los sistemas y subsistemas del SIGEP.

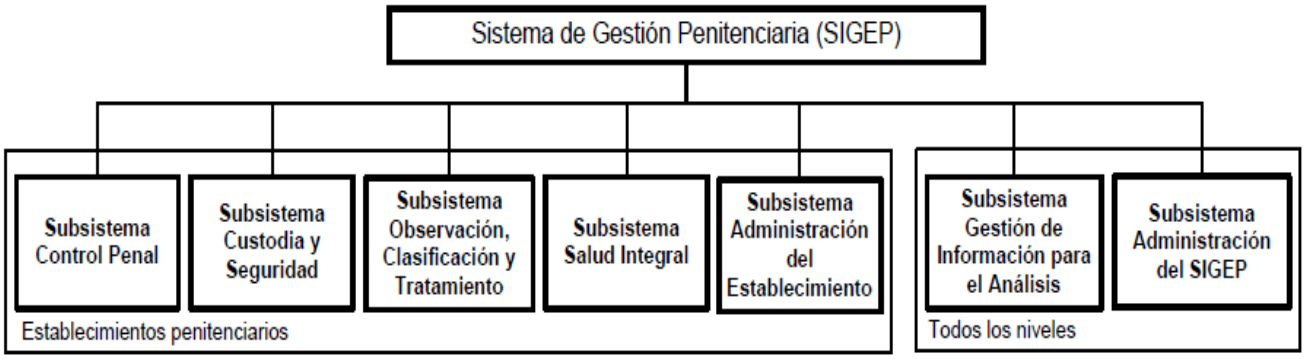


Figura 1.1 Sistemas y Subsistemas del SIGEP

1.4.1 Subsistema Seguridad y Custodia

“Esta área permitirá controlar las actividades de custodia a los reclusos, tanto dentro del penal como fuera de este. También es la encargada del control del inventario del armamento, municiones y los medios técnicos asignados a los custodios para el cumplimiento de sus funciones dentro de los establecimientos penitenciarios. Permite mantener un control sobre las capacidades de los establecimientos penitenciarios.

Además aquí se lleva un control sobre las visitas que se reciben en los establecimientos penitenciarios, tanto las relacionadas con los reclusos como de carácter administrativo.” (ARIAS)

Seguridad y Custodia contiene los procesos gestión de cupo, salidas transitorias, traslados inter-penales, visitas familiares, visitas institucionales, requisas y decomisos, registro de pertenencias, novedades y contingencias, medidas preventivas y disciplinarias, ubicación y control de armamentos, equipos y medios de seguridad.

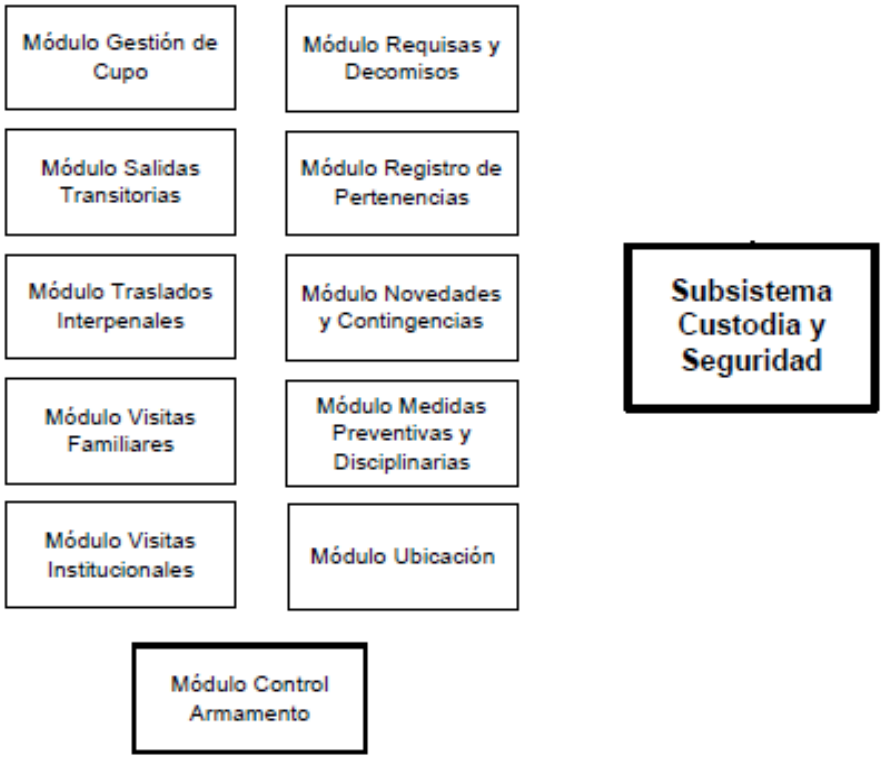


Figura 1.2 Módulos del subsistema Seguridad y Custodia

1.4.2 Módulo Control de armamentos, medios y equipo de seguridad

'”Este módulo permitirá controlar el inventario del armamento, equipos y los medios de seguridad que utilizan los custodios para el cumplimiento de sus funciones dentro de los establecimientos penitenciarios. Igualmente, permite llevar el control de entregas y Devoluciones del armamento al servicio de guardia, permitiendo conocer en todo momento quien posee cada medio. El módulo incluye funcionalidades además para gestionar pérdidas y bajas”. (ARIAS)

El módulo se inserta dentro de 4 áreas de procesos principales: Gestión de los armamentos, equipos y medios de seguridad donde el funcionario de seguridad y custodia podrá registrar y modificar los armamentos, equipos o medios de seguridad, eliminarlos o consultar los detalles de cada elemento; Gestión de entregas donde se controlará el registro de entregas, el historial de las entregas, las Devoluciones y los detalles de las mismas; Gestión de pérdidas donde se manejan las pérdidas y las reincorporaciones; Gestión de bajas donde se maneja los armamentos, equipos y medios de seguridad dados de baja y su reincorporación.

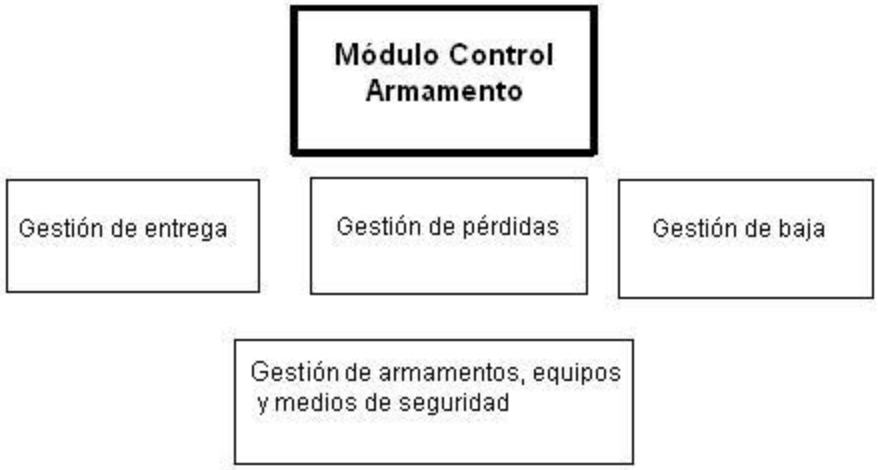


Figura 1.3 Funcionalidades del módulo Control de armamentos, equipos y medios de seguridad

1.5 Tecnologías y herramientas utilizadas en el desarrollo

Una adecuada selección de las herramientas y tecnologías (frameworks) que conforman el ambiente de desarrollo (Development Environment) es imprescindible en la producción de un software e incidirá directamente en el tiempo de desarrollo y la calidad final del producto.

Todas las herramientas y tecnologías utilizadas en la realización de SIGEP, así como sus versiones y su integración fueron definidas por el grupo de arquitectura de SIGEP, es por esto que en el presente capítulo no se realiza un análisis de ellas sino, se enumeran las utilizadas y se mencionan sus principales características.

A continuación se abordan algunas herramientas y tecnologías utilizadas en la realización de SIGEP.

1.5.1 Programación orientada a objetos

La programación orientada a objetos (POO) es una forma especial de programar con un enfoque más cercano a como se expresan las cosas en la vida real que otros tipos de programación, es un paradigma de programación que expresa un programa como un conjunto de objetos que colaboran entre ellos para realizar tareas. Un objeto no es más que un conjunto de atributos o datos que representan las características del objeto, y métodos que son las funcionalidades asociadas al objeto, relacionados entre sí, que se usan para modelar objetos, procesos o conceptos del mundo real.

Un lenguaje orientado a objetos es un lenguaje de programación que permite el diseño de aplicaciones orientadas a objetos. En SIGEP se utiliza como lenguaje orientado a objetos Java, exponente puro de este paradigma, este lenguaje incorpora el uso de la orientación a objetos como uno de los pilares básicos y fundamentales del lenguaje.

1.5.2 Metodología de Desarrollo

Una metodología de desarrollo es una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. La finalidad de una metodología de desarrollo es garantizar la eficacia (cumplir los requisitos iníciales) y la eficiencia (Ejemplo: minimizar las pérdidas de tiempo) en el proceso de generación de software.

Las metodologías proponen la realización de un número de actividades y la elaboración de un conjunto de artefactos que, dependiendo de las características de cada proyecto de software pueden realizarse o no, puede ser por falta de tiempo o porque descubren que no son necesarios. En estos casos se define una adecuación de la metodología a utilizar. Dentro del proyecto SIGEP se decide la utilización de RUP como metodología de desarrollo y se define una adecuación de la misma, partiendo de las características del proyecto.

Rational Unified Process (RUP)

Es un proceso de desarrollo de software que captura las mejores prácticas del conocimiento de líderes en ingeniería de software (Desarrollo iterativo del software, Administración de requerimientos, Uso de arquitecturas basadas en componentes, Modelamiento visual del software, Verificación de la calidad del software, Control de cambios) y proporciona a los equipos de desarrollo guías, estándares y recomendaciones para la construcción de software de alta calidad. Las mejores prácticas de desarrollo de software están documentadas como principios clave. RUP unifica los mejores elementos de metodologías

anteriores, es una metodología de desarrollo preparada para desarrollar grandes y complejos proyectos, los cuales presentan extensos cronogramas y equipos de desarrollo números. Utiliza el Lenguaje de Modelado Unificado (Unified Modeling Language – UML) como lenguaje de representación visual.

Tiene como objetivo asegurar la producción de software de calidad dentro de plazos y presupuestos predecibles.

En su libro El Proceso Unificado de Desarrollo de Software los tres metodólogos creadores de RUP(Ivar Jacobson, Grady Booch and James Rumbaugh) lo caracterizan como:

- Dirigido por Casos de Uso
- Centrado en Arquitectura
- Iterativo e Incremental

En la Figura 1.4 se muestran las distintas fases y flujos de trabajos e iteraciones que considera RUP en el desarrollo de un software.

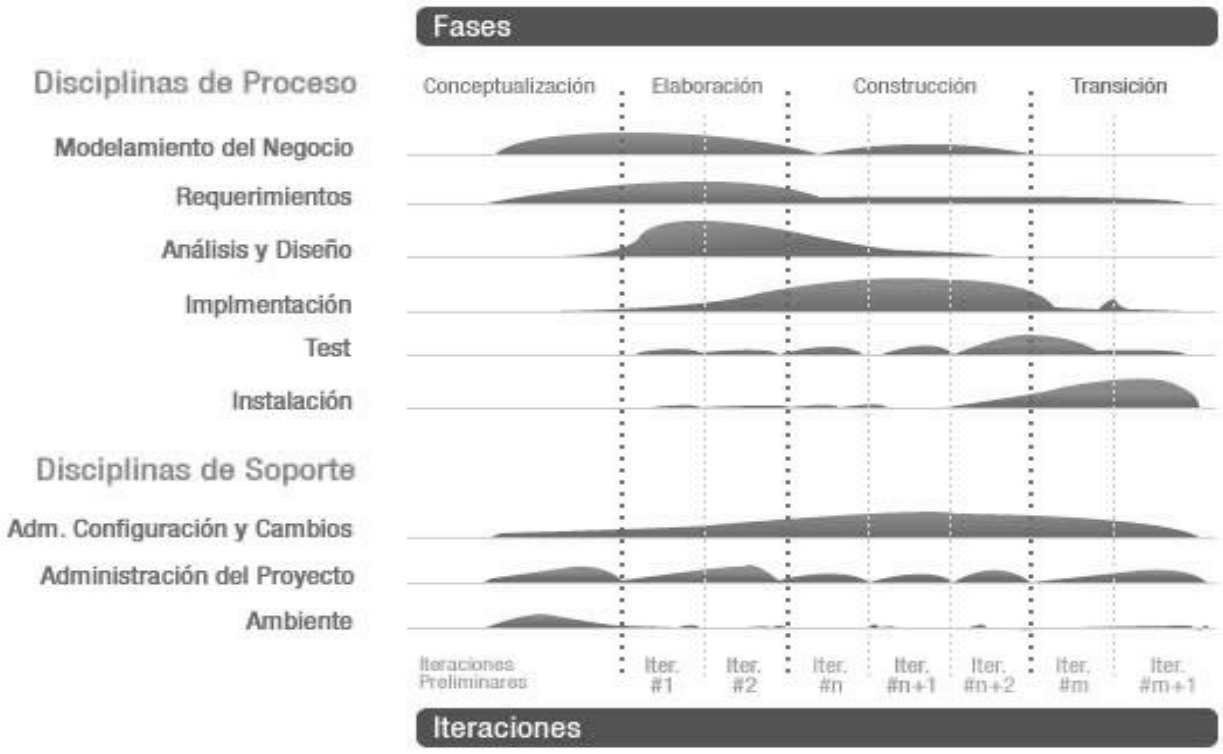


Figura 1.4 Flujo de trabajo de la metodología RUP

Los flujos de trabajo que más peso tendrán durante el desarrollo de nuestro trabajo serán Diseño, Implementación y Prueba.

Fase Diseño

Los objetivos del Flujo de Trabajo Diseño son transformar los requerimientos en un diseño de cómo va a ser implementado el sistema, evolucionar hacia una arquitectura del software robusta, adaptar el diseño para que coincida con el ambiente de implementación, diseñando el sistema con un enfoque hacia el rendimiento. Los principales artefactos que propone RUP para esta disciplina son: Modelo de Análisis, Modelo de Diseño, Modelo de Datos y Modelo de Despliegue. Las entradas fundamentales de esta disciplina son las especificaciones de los requisitos de software obtenidas en la disciplina Requisitos. Los trabajadores más importantes de este flujo son el diseñador y el arquitecto.
Según la adecuación hecha por SIGEP de la metodología de desarrollo RUP para nuestro proceso de desarrollo, la cual se especifica en “Visión del sistema de gestión penitenciaria”(ARIAS), durante este flujo de trabajo los artefactos que se van a generar son:

Modelo de Diseño: Es un modelo de objetos que describe la realización física de los casos de uso. Contiene subsistemas, paquetes, clases de diseño y diagramas de clases e interacción, sirve de abstracción de la implementación y es usada como entrada fundamental a las actividades de implementación.

Modelo de datos: Describe la representación lógica y física de los datos persistentes de nuestro negocio.

Fase de Implementación

Los objetivos de esta fase son definir la organización del sistema en términos de Subsistemas de Implementación organizados en capas, probar los componentes desarrollados independientemente como unidades, a los cuales se le realiza pruebas de unidad e integrar los resultados en un sistema ejecutable.

Como salida de esta fase obtenemos el Modelo de Implementación el cual incluye: Subsistemas de implementación y sus dependencias, interfaces y contenidos, componentes (ficheros, ejecutables) y dependencias entre ellos.

Fase de pruebas

La prueba está enfocada principalmente en la evaluación y determinación de la calidad del producto. Durante

esta fase lo fundamental es encontrar y exponer las debilidades en el software.

1.5.4 Patrones de Diseño

Un *patrón de diseño* es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).Encierran la experiencia que programadores e ingenieros han adquirido en la solución de problemas comunes.

El uso de patrones de diseño incrementa la compresión de un diseño o de su implementación, disminuye en gran medida la cantidad de código al eliminar las implementaciones innecesarias, ayuda a los diseñadores a separar conceptos y simplifican la descripción de las implementaciones. Además facilitan la comunicación entre diseñadores, pues establecen un marco de referencia (terminología, justificación) y dota a los programadores de un lenguaje común.

Algunos de los patrones utilizados en la solución que se brinda en este trabajo de diploma para el diseño del módulo de Armamento son:

Patrón Facade: Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. Proporciona un bajo acoplamiento al diseño porque reduce la excesiva dependencia entre objetos.

Data Access Object (DAO): Define una capa encargada de manejar todo lo relacionado con la persistencia de los datos (Capa de Acceso a Datos). Aísla los objetos de la capa de negocio de la tecnología de persistencia utilizada de forma que lo hace menos complejos. En el caso de cambio o actualización de la tecnología de persistencia, solo sufre cambio esta capa de acceso a datos.

Controller (Controlador): Este patrón define una clase que tendrá la responsabilidad de controlar el flujo de eventos de un sistema, estas clases se denominan controladores. Los controladores delegan las tareas en otras clases generalmente de la capa de negocio.

Model-View-Controller (Modelo-Vista-Controlador, MVC): Divide una aplicación en tres componentes, el modelo, las vistas y los controladores. El modelo es la representación específica del dominio de la información sobre cual funciona la aplicación; las vistas presentan el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario; el controlador responde a eventos, usualmente acciones del usuario.

1.5.5 Herramienta de modelado

Visual Paradigm

Es una suite completa de herramientas CASE profesional y multiplataforma que soporta el ciclo de vida completo del desarrollo de software. Este software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, eficientes y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación, se integra con IDEs de desarrollo como el Eclipse. Proporciona además abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Sitio web oficial: http://www.visual-paradigm.com

1.5.6 Plataforma de desarrollo: Java 2 Enterprise Edition (J2EE)

La J2EE define un estándar para el desarrollo de aplicaciones empresariales multicapa, diseñado por Sun Microsystems. J2EE simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, proveyendo un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja por parte de los desarrolladores.

J2EE incluye soporte completo para los componentes de API[3]s, framework y patrones de implementación como las APIs Java Servlets y JDBC, y las tecnologías JavaServer Pages y XML entre otras, las cuales son usadas en SIGEP.

Java Servlets

La tecnología Java Servlets proporciona a los desarrolladores web de un mecanismo sencillo y consistente para extender la funcionalidad de un servidor web que es accesible a través del modelo petición-respuesta. Generalmente es utilizado en el desarrollo de aplicaciones web**.** Los servlets se cargan de forma dinámica por el entorno de ejecución Java del servidor cuando se necesitan, cuando se recibe una petición a través del protocolo HTTP del cliente, el contenedor/servidor web inicia el servlet requerido. El servlet procesa la petición del cliente y envía la respuesta de vuelta al contenedor/servidor, que es enrutada al cliente. La tecnología servlet proporciona las mismas ventajas del lenguaje Java en cuanto a portabilidad y seguridad, ya que un

[3] API, del inglés *Application Programming Interface*: 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.

servlet es una clase de Java igual que cualquier otra y por tanto, tiene en ese sentido todas las características del lenguaje.

Java Data Base Connectivity (JDBC.)

Es un API que brinda un conjunto de clases e interfaces, que permite el acceso por parte de programas escritos en Java a los datos de un base de datos relacional, utilizando diversos controladores (conocidos también como drivers) y realizar sobre estos las operaciones más comunes.

El resultado es un conjunto de clases e interfaces, que pueden ser utilizadas con cualquier gestor que tenga un controlador JDBC apropiado. JDBC es utilizada por la mayoría de frameworks de persistencia, como es el caso de Hibernate, el cual es el que se usa en SIGEP para el trabajo sobre la base de datos.

Apache Tomcat

Es un contenedor de servlets creado por la fundación Apache dentro del proyecto Jakarta en un entorno abierto y participativo y publicado bajo la licencia del software de Apache. Implementa las especificaciones de los servlets y de JavaServer Page (JSP). Es usado en números as aplicaciones web de gran escala y críticas en una amplia gama de industrias y organizaciones.

Sitio web oficial: http://tomcat.apache.org/

1.5.7 Frameworks

Entre las definiciones de frameworks tenemos**:**

"...diseño abstracto orientado a objetos para un determinado tipo de aplicación, que se compone de una clase abstracta para cada componente principal del diseño; contendrá normalmente una librería de subclases que pueden ser utilizadas como componentes del diseño..." (JOHNSON).

"una mini arquitectura reusable que provee una estructura y comportamiento Genéricos para una familia de abstracciones de software [...]" (KAISLER).

Por esta razón un framework se identifica a veces como "el viejo código que llama al nuevo código".

Existen varias clasificaciones para los frameworks de acuerdo a diferentes criterios. A continuación se muestra una de estas clasificaciones según Taligent (KAISLER):

• **Frameworks de aplicación**: Encapsulan una capa de funcionalidad horizontal que se puede aplicar en la construcción de una gran variedad de aplicaciones. Uno de los ejemplos más completos de este tipo de frameworks los constituyen los que implementan las interfaces gráficas de usuario (GUI)...

• **Frameworks de dominio**: Un framework de dominio implementa una capa de funcionalidad vertical correspondiéndose con un dominio de aplicación o una línea de producto. Su evolución deberá ser también la más rápida, pues deben adaptarse a las áreas de negocio para las que están diseñados.

• **Frameworks de soporte**: Proporcionan servicios básicos a nivel de sistema como manejo de memoria, reportes, etcétera.

Dentro de SIGEP utilizamos, entre otros, el framework de aplicación Spring y el framework de soporte Hibernate los cuales brindan múltiples ventajas, en el desarrollo de aplicaciones web dentro de la plataforma J2EE.

1.5.7.1 Frameworks Spring

Es un framework de aplicación, se distribuye de forma libre y su código es abierto. Como es el objetivo fundamental de los frameworks, Spring ayuda al desarrollo de aplicaciones, permitiendo la reutilización de código, el ahorro en tiempo de diseño e implementación por parte de los desarrolladores. Le brinda una estructura completa a las aplicaciones de una manera consistente y productiva para crear arquitecturas coherentes. Interviene en todas las capas de la aplicación y presenta un diseño que brinda gran flexibilidad arquitectónica. Permite configurar complejas aplicaciones a partir de componentes simples. En Spring los objetos de la aplicación se declaran en ficheros (normalmente en formato XML) y el framework se encarga de instanciarlos y configurarlos correctamente a través de la inyección de dependencias.

El framework Spring está dividido en varios módulos los cuales presentan puntos de contactos con otros frameworks y librerías de la plataforma J2EE como Hibernate JMS, JMX, y RMI. De sus módulos los más usados dentro de la arquitectura de SIGEP se encuentran:

• **Core (Núcleo):** Es la parte fundamental del framework ya que provee toda la funcionalidad de Inyección de Dependencias permitiendo administrar la funcionalidad del contenedor de beans. El concepto básico de este módulo es el BeanFactory que implementa el patrón de diseño Factory (fábrica) eliminando la necesidad de crear objetos de forma programada, permitiendo desligar la configuración y especificación de las dependencias de la lógica de programación.

• **Spring-ORM:** Provee capas de integración para APIs de mapeo objeto – relacional como Java Persistence API, Java Data Objects, Apache OJB, JDO, iBATIS SQL Maps y Oracle's TopLink, e Hibernate como es el caso de SIGEP basados en el patrón DAO. Brinda además algunos servicios, como la integración con el mecanismo de transacciones declarativas de Spring y el manejo transparente de excepciones.

• **Spring-MVC:** Provee de una implementación Modelo - Vista - Controlador para las aplicaciones web. La implementación de Spring MVC permite una separación entre código de modelo de dominio y las formas web y permite el uso de otras características de Spring Framework como lo es la validación. Spring se integra con frameworks de soporte MVC como Struts, JSF, WebWork y Trapestry pero también provee su propia implementación del modelo MVC basada en el patrón Controlador Frontal.

Sitio web oficial: http://www.springframework.org

1.5.7.2 Framework Hibernate

Hibernate es un importante framework de mapeo objeto/relacional y servicio de consultas para Java (y disponible también para. Net con el nombre de Nhibernate).Se distribuye bajo los términos de la licencia GNU LGPL. Facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML), detallando cómo es su modelo de datos, qué relaciones existen y qué forma tienen, con esta información Hibernate le permite a la aplicación manipular los datos de la base de datos, operando sobre ellos como objetos, con todas las características de la Programación orientada a Objetos (POO).

Hibernate genera además las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias. También ofrece un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), y al mismo tiempo APIs para construir las consultas de forma programada.

Sitio web oficial: http://www.hibernate.org/

1.5.7.3 Framework DOJO

Dojo es un framework que da soporte a la capa de aplicación, de código abierto DHTML (Dynamic HTML) escrito en Java Script. Se compone de un conjunto de librerías y widgets preempaquetados de código Java Script, HTML y CSS, para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX y le brinda a las vistas un conjunto de funcionalidades y un estilo atractivo. Su idea es la de abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código Java Script a utilizar sea diferente. Algunas de las características que presenta este framework son:

- Maneja incompatibilidades entre navegadores.
- Oculta el manejo del XMLHttpRequest.
- Manipulación de DOM (Document Object Model).
- Animaciones.

- APIs para el manejo de Ajax y Cometd.
- Manejo de Eventos.
- Drag and Drop.
- Conjunto de Componentes Reutilizables (widgets).
- Gráficos.

Sitio web oficial: http:/www.dojotoolkit.org/

1.5.8 Gestor de Base de Datos

Oracle

Es un sistema de gestión de base de datos relacional utilizando tecnología cliente/servidor, líder en la industria. Permite el manejo de todo tipo de datos del negocio como datos relacionales, documentos, multimedia, XML y datos de localización. Es un sistema de base de datos muy robusto y ofrece una gran disponibilidad, escalabilidad, fiabilidad y seguridad en el manejo de grandes volúmenes de datos. Es un sistema multiplataforma.

Oracle es un sistema de base de datos muy difundido sobre todo en las grandes compañías, transnacionales o instituciones gubernamentales por sus elevados precios de licencias de software y del soporte técnico, además de sus elevados requisitos de hardware.

Sitio web oficial: http://www.oracle.com

1.5.9 Entorno integrado de desarrollo (IDE)

Un ambiente de desarrollo integrado (Integrated Development Environment (IDE)) es un programa compuesto por un conjunto de herramientas para un programador. El mismo puede dedicarse a un solo lenguaje de programación o a varios. Consta de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

Eclipse

Es principalmente una plataforma libre de programación, usada para crear entornos integrados de desarrollo (conocidos por sus siglas en inglés: IDE).Eclipse es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

Aunque mayormente se usa como IDE para el desarrollo de aplicaciones en Java, no es el único lenguaje que permite, el soporte para java es proveído por el plugin JDT (Java Development Tools), existen plugins para

otros lenguajes como C/C++, Cobol, C#. Esta es una de las características más importantes de Eclipse, su facilidad de extensión a través de plugins.

Plugin o Plug-In: es una aplicación que interactúa con otra para agregarle una funcionalidad específica y es ejecutada por la aplicación principal.

Sitio oficial:http://www.eclipse.org/

Algunos de los plugins más importantes en SIGEP son:

• **Spring IDE v2.0 (2007)**: Es un plugin que sirve como interfaz de usuario gráfica para la configuración de los archivos usados por Spring Framework. Agiliza el trabajo con el framework Spring, principalmente en proyectos grandes, puesto que contiene un editor XML que permite autocompletado y un validador y visor de los beans configurados. Permite la búsqueda y visualización de todos los beans y dependencias.

Sitio: http://www.springide.org

• **Web Tool Platform (WTP):** La plataforma de herramientas web de Eclipse o Eclipse Web Tool Platform (WTP) provee varias APIs para el desarrollo de aplicaciones sobre la Web y J2EE. Es diseñada por la fundación Eclipse. Incluye editores para JavaServer Pages, HTML, CSS, JavaScript, XML, y XSD (XML Schema Definition) y API para soportar el despliegue, ejecución y prueba de aplicaciones. (http://www.eclipse.org).

Hibernate Tools: Provee de un conjunto de herramientas para facilitar el uso del framework Hibernate. Entre su principales funcionalidades están: editor de mapeos de los meta datos de la base de datos a las clases, una consola para la ejecución de consultas HQL, permite realizar ingeniería inversa a partir de la base de datos de las clases y de los mapeos de las entidades. (http://www.hibernate.org)

• **Subclipse v1.0.1**: Es un plugin para Eclipse que permite conectar este IDE a repositorios de Subversion (sistema de control de versiones que facilita y permite el trabajo de equipo en el desarrollo de un producto) y la manipulación del proyecto que reside en este repositorio desde el Eclipse. (http://subclipse.tigris.org).

1.6 Arquitectura del módulo Control de armamento, equipos y medios de seguridad

La arquitectura a seguir durante el desarrollo de la solución de diseño e implementación que se presenta en este trabajo de diploma para el módulo Control de armamento, equipos y medios de seguridad, está

condicionada por la arquitectura definida para el proyecto SIGEP, la misma está basada en ArBaWeb (Arquitectura Base sobre la Web), personalizando este framework a las condiciones y necesidades del proyecto y el módulo.

A continuación se abarcan los aspectos arquitectónicos más significativos que define ArBaWeb.

Modelo de Capas

La arquitectura del SIGEP estará regida por una arquitectura de tres capas lógicas fundamentales bien definidas: Capa de Presentación, Capa de Lógica de Negocio, y Capa de Acceso a Datos. Estas capas además de estar separadas lógica y estructuralmente, se encuentran separadas de manera física. En cada módulo existe una estructura de paquetes con este fin, la cual definiremos más adelante.

Este modelo de capa permite que cada capa pueda ser modificada tanto como sea posible sin provocar un impacto en las restantes. Una capa no es consciente de lo que le ocurre a la capa superior, su dependencia es puramente con la capa inmediata inferior. Esta dependencia entre capas es normalmente entre interfaces, asegurando que el acople sea el más bajo posible.

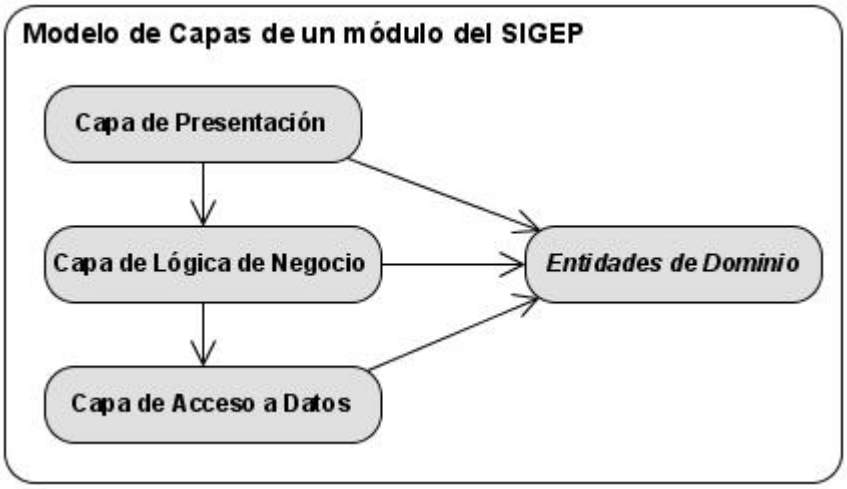


Figura 1.5 Modelos de capas de un módulo del SIGEP

Un concepto tratado en la figura, es “objetos persistentes del dominio (entidades)”. Este conjunto de objetos puede llegar a considerarse otra capa lógica que puede presentar este tipo de arquitectura. Estos objetos de dominios no presentan lógica de negocio, esto permite utilizarlos solo como contenedores de información que tradicionalmente son pasados hasta la capa de presentación, en la cual mostrarán los datos que contienen, pero no podrán ser modificados, ya que esto solo ocurre dentro de los contextos transaccionales definidos en

la capa de servicios de negocio, donde son los objetos de negocio quienes tienen la responsabilidad de la lógica de negocio.

En la arquitectura del SIGEP se define una fachada que representa la interacción entre las Capas de Presentación y Lógica de negocio, la utilización de esta fachada implementa y cumple con el patrón de diseño Facade.

A continuación un análisis de las distintas capas:

Capa de Acceso a Datos

La capa de acceso a datos se encarga de la comunicación con la tecnología de persistencia usada en el proyecto. En esta capa se encuentran los objetos que encapsulan la lógica de acceso a datos o en inglés, Data Access Object (DAO) e interfaces brindadas a través de las cuales la capa de lógica de negocio se comunica con la capa de acceso a datos. Los DAOs encapsulan la persistencia de los objetos de dominios, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos.

Las implementaciones de los DAOs estarán disponibles para los objetos de la capa de lógica de negocio haciendo uso de la inyección de dependencias entre los objetos de negocio y las instancias de los DAOs, configurada en el contenedor de inversión de control de Spring Framework.

Las implementaciones de los DAOs extienden clases de soporte del framework Spring para el uso de este patrón usando el framework ORM Hibernate, para la realización de sus principales métodos sobre el gestor de base de datos:

- **Métodos para descubrir:** Estos localizan los objetos almacenados para ser usados por la capa de servicios de negocio.
- **Métodos para persistir o salvar:** Estos hacen persistentes a los objetos transitorios.
- **Métodos para eliminar:** Estos eliminan a los objetos guardados en el medio de almacenamiento (generalmente una base de datos).
- **Métodos para conteos y otras funciones agregadas:** Estos devuelven los resultados de operaciones que son más eficientes implementarlas usando funcionalidades de la base de datos (procedimientos almacenas, etcétera.) que iterar sobre los mismos objetos.

Capa de lógica de negocio

En esta capa radican los objetos de negocio o Business Objects. Los objetos de negocio separan los datos y la lógica de negocio usando un modelo de objetos. Los objetos de negocio son managers e interfaces. Cada módulo definirá una o más fachadas en caso de que se requiera, que agrupen los métodos de negocio implementados en los manejadores o managers. La fachada de un módulo se basa en el patrón Facade para permitir una clara división entre las capas arquitectónicas. Los Managers son las clases que se especializan en un conjunto de funcionalidades que representan el negocio sobre una o varias entidades. Los Managers son las únicas clases en la aplicación que tendrán lógica de negocio mientras que las fachadas se limitarán solamente a agrupar las funcionalidades para ser expuestas a capas superiores.

Estas son las funcionalidades que presenta esta capa:

- **Lógica de negocio específica de procesos de negocios:** En esta definición de arquitectura los objetos de dominio no presentan ningún tipo de lógica de negocio, sino que esta responsabilidad recae sobre los objetos de negocio, permitiendo usar a los objetos de dominio como objetos de transferencia que se mueven entre las capas arquitectónicas de la aplicación.
- **Puntos de entrada muy bien definidos para las operaciones de negocio implementadas:** Los objetos de negocio brindan las interfaces usadas por la capa de presentación.
- **Control de transacciones:** Las políticas de transaccionales de la aplicación serán planteadas al sobre los objetos de negocio.
- **Ejecución de restricciones de seguridad:** Las restricciones de seguridad en esta capa ser realizará en los puntos de entradas a la capa media, es decir en los objetos de negocio.
- **Ejecución de la auditoría:** Sobre esta capa se llevará a cabo la auditoría sobre los métodos de negocio.

Capa de presentación

La capa de presentación descansa sobre una capa de servicios de negocio. Esto significa que esta capa será fina y no contendrá lógica de negocio, en esta capa residen la definición de las peticiones que el usuario puede realizar sobre la aplicación y los controladores que manejan el flujo web configuradas a través de los ficheros del framework Spring, y la comunicación con las interfaces de la capa de negocio. Además se encuentran las vistas HTML, XML, PDF, XLS que se muestran al usuario y la implementación de los ficheros JavaScript utilizando el framework Dojo responsables del comportamiento dinámico de los documentos HTML. Esta capa puede variar en complejidad en el transcurso del desarrollo de la aplicación, sin embargo ninguna de las capas inferiores debe ser afectada por estos cambios.

Las responsabilidades principales de la capa de presentación son: la navegabilidad del sistema, el formateo de los datos de salida, la validación de los datos de entrada y la construcción de la interfaz gráfica de usuario.

El cliente interactúa con esta capa directamente utilizando el navegador Web a través de peticiones URL. En el cliente se forma dinámicamente una pequeña capa de presentación que está constituida por código html y JavaScript. Esta capa interactúa directamente mediante peticiones basadas en la tecnología Ajax con los componentes necesarios del servidor creados bajo la tecnología de SpringMVC.

Estructura de paquetes

Organización de la aplicación subsistemas y módulos.

Con el objetivo de organizar la aplicación se definieron, como unidades de organización, subsistemas y módulos. Un módulo encapsula un conjunto de funciones que debe realizar el sistema, las cuales son agrupadas por tener características muy similares y se definen en la etapa de diseño. Un subsistema se refiere a un conjunto de módulos que por razones de similitud o de perseguir objetivos comunes, son agrupados.

Los subsistemas se clasifican de acuerdo al proceso que los identificó en el subsistema común, subsistemas de negocio y de soporte:

El subsistema común contiene las funcionalidades y elementos comunes a la aplicación por ejemplo el recuperador de información, la ficha de control de un individuo, el resumen legal, etcétera.

Los subsistemas de negocio son aquellos identificados a partir de la captura de requisitos y contienen la solución a los requisitos funcionales y no funcionales de procesos o áreas de procesos dentro del sistema penitenciario venezolano, por ejemplo: los subsistemas Control Penal, Seguridad y Custodia, Tratamiento y Salud Integral.

Los subsistemas de soporte son aquellos que cubren funcionalidades como respuesta a requisitos no funcionales esperados del SIGEP; por ejemplo, el subsistema Administración, que se encarga de administrar la aplicación.

Estructura de un módulo

En un módulo generalmente existen todas las capas lógicas que se definen en la arquitectura base. Cada componente que se desarrolle en un módulo tiene un lugar definido en esta estructura de paquetes.

Los paquetes que conforman esta estructura son:

configuration: Se localizarán todos los archivos que tienen que ver con la configuración del módulo, los "*Application-Context"* de Spring Framework, los archivos utilizados para la internacionalización, ficheros de propiedades *".properties"* y cualquier otro destinado a estos fines.

dao: Se encontrarán las interfaces DAOs.

dao.impl: Se encontrarán las implementaciones de las interfaces DAOs.

dao.map: Se encontrarán los ficheros de mapeo utilizados por Hibernate.

dao.test: Se encontrarán las clases de prueba utilizadas para realizar las pruebas de unidad a las implementaciones de los DAO.

domain: Se encontrarán todas las entidades persistentes o no del dominio pertenecientes al módulo.

facade: Se encontrarán las interfaces de las fachadas de negocio.

facade.impl: Se encontrarán las implementaciones de las fachadas de negocio.

service: Se encontrarán las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicio.

error: Se encontrarán las excepciones y las clases de utilidad para las validaciones.

web: Se encontrarán los controladores de la capa de presentación.

web.command: Se encontrarán las clases utilizadas para guardar datos procedentes de las peticiones web (Command).

web.editor: Se encontrarán los *PropertyEditors*[4].

web.validator: Se encontrarán las clases utilizadas para validar datos (Validadores).

Figura 1.6 Estructura de paquetes de un módulo del SIGEP

Mecanismos de colaboración

Los mecanismos de colaboración son estrategias propuestas para establecer la forma de comunicación entre los componentes del software.

Como ya se ha explicado, un subsistema agrupa una colección de módulos, por consiguiente, si se detecta que dos módulos deben utilizar una misma funcionalidad, pueden existir varios escenarios para determinar la estrategia por la que se relacionarán. Si varios módulos de un subsistema comparten una misma funcionalidad y ninguno de ellos es responsable de implementarla entonces estamos en presencia de una funcionalidad común entre ellos, en dicho caso si los módulos pertenecen a un mismo subsistema, esta funcionalidad se colocará en el módulo común del subsistema (“common”) con el cual todos los módulos del subsistema presentan una visibilidad total. Si los módulos pertenecen a subsistemas distintos o varios subsistemas necesitan la misma funcionalidad y ninguno puede especializarse en la misma entonces se implementa en el módulo “Common Global”, en el cual se implementan los métodos generales para toda la aplicación. En el caso de que uno de los módulos sea responsable o especialista en implementar la funcionalidad entonces la misma la realiza el módulo especialista, la exporta a los demás módulos a través de una fachada de servicio y estos la consumirán.

CAPÍTULO 2: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

2.1 Introducción

Durante este capítulo se desarrolla la solución brindada en este trabajo de diploma para el diseño y la implementación del módulo Control de armamento, equipos y medios de seguridad. Se muestran por orden de realización los resultados de cada una de las actividades a realizar definidas en la adecuación hecha por SIGEP de la metodología de desarrollo RUP para las fases de diseño e implementación, aunque no se representan todos los diagramas e implementaciones hechas sino solo una muestra representativa de las funcionalidades distintivas del módulo.

2.2 Análisis de las funcionalidades.

El análisis de las funcionalidades a implementar se realiza a partir de la documentación generada en la captura de requisitos, en taller de trabajo. El resultado es el entendimiento común de las funcionalidades que el módulo tiene que proveer y las principales restricciones a implementar.
En esta actividad se identifican puntos de contacto con otros subsistemas o módulos y se establece la forma de proceder en esa comunicación. También se definen las interfaces gráficas a partir de las pautas generales definidas para la aplicación y el flujo básico de navegación.

Los requisitos funcionales han sido agrupados en cuatro áreas fundamentales, como se muestran a continuación:

Gestión de Armamentos, Equipos y Medios de Seguridad

- Listar armamentos, equipos y medios de seguridad
- Registrar armamentos
- Registrar equipos
- Registrar medios de seguridad
- Modificar armamentos
- Modificar equipo
- Modificar medios de seguridad
- Eliminar armamento, equipo o medio de seguridad
- Consultar detalles de armamento, equipo o medio de seguridad

Gestión de Bajas

- Dar baja a armamento, equipo o medio de seguridad
- Reincorporar armamento, equipo o medio de seguridad
- Consultar historial de bajas
- Eliminar baja de armamento, equipo o medio de seguridad
- Consultar detalles de una baja

Gestión de Entregas

- Registrar entrega de armamentos, equipos y/o medios de seguridad
- Modificar entrega de armamentos, equipos y/o medios de seguridad
- Eliminar entrega de armamentos, equipos y/o medios de seguridad
- Consultar detalles de una entrega
- Listar entregas en ejecución
- Consultar historial de entrega
- Registrar devolución

Gestión de Pérdidas

- Listar elementos perdidos activos
- Registrar pérdida
- Consultar detalles de una pérdida
- Consultar historial de pérdidas
- Reincorporar armamento, equipo o medio de seguridad
- Eliminar pérdida

2.3 Diseño del módulo Control de armamento, equipos y medios de seguridad del SIGEP

Las tareas a realizar dentro de esta fase del desarrollo del software son: el diseño de las entidades del dominio, diseño del modelo de datos, diseño de la Capa de Acceso a Datos, diseño de la Capa de Negocio y diseño de la Capa de Presentación.

2.3.1 Diseño de las entidades del dominio

Una de las entradas principales a las restantes actividades de diseño, es el diseño del dominio. En este se identifican las entidades que serán gestionadas por la capa de negocio, persistidas o recuperadas por la capa de acceso a datos y mostradas por la capa de presentación. También se identifican los nomencladores

necesarios. Las clases del dominio se definen en el paquete *domain* de cada módulo. La definición del dominio, en especial de las entidades persistentes sirve como una primera aproximación al diseño definitivo del modelo de datos.

En la figura Figura 2.1 se muestra el diagrama de clases persistentes del módulo Control de armamento, equipos y medios de seguridad.

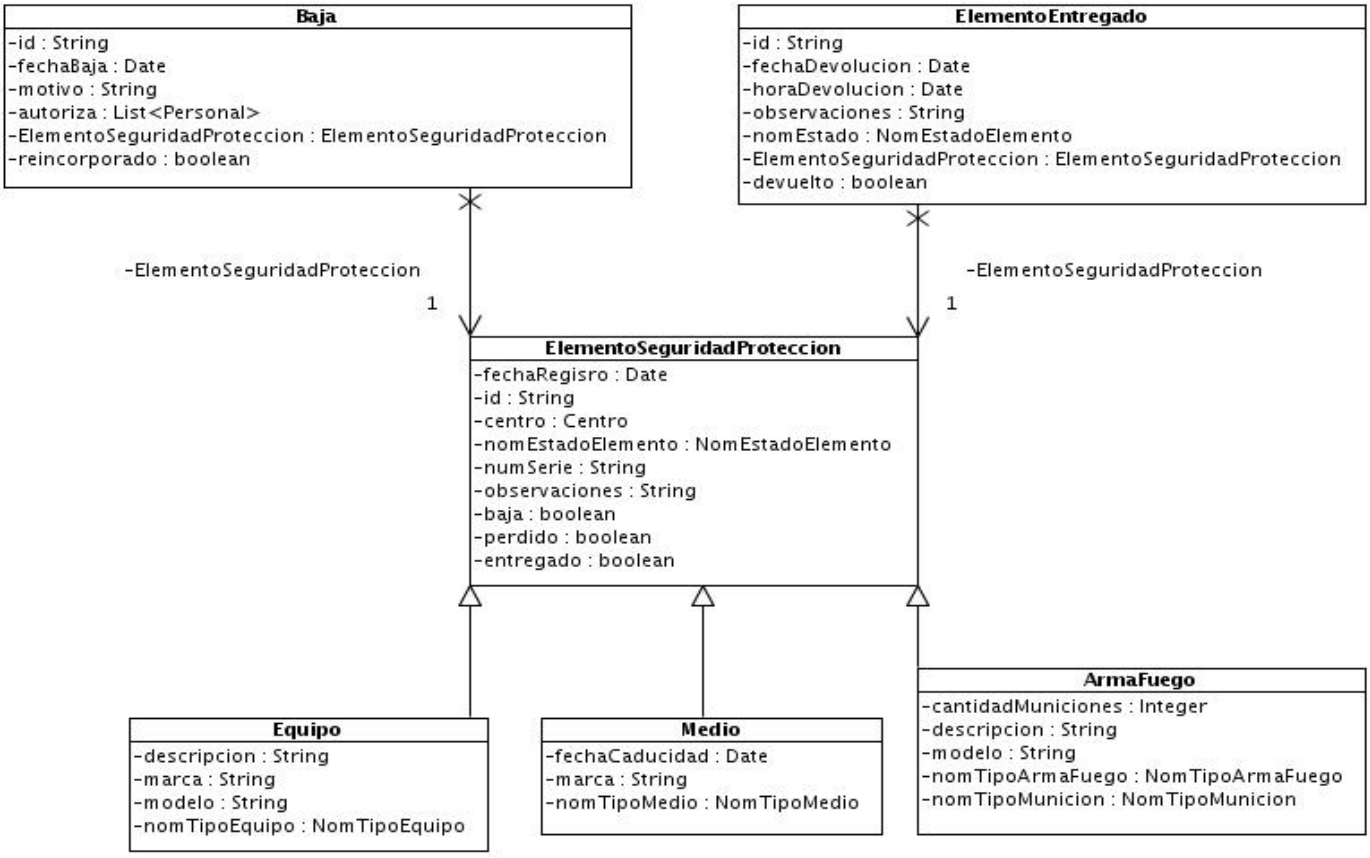


Figura 2.1 Diagrama de clases persistentes del módulo Control de armamentos, equipos y medios de seguridad

El modelo de dominio del módulo Control de armamento, equipos y medio de seguridad general se puede observar en el **Anexo 2.1**, donde se representa el diagrama del dominio

.2.3.2 Diseño del modelo de datos

Como resultado de esta actividad se obtiene el modelo de datos del módulo. La definición de las entidades persistentes aporta al modelo de datos una buena aproximación de la estructura estática de la base de datos. Esta estructura tiene que garantizar que la recuperación y almacenamiento de la información ocurra de manera correcta y que se cumplan las restricciones. En la Figura 2.2 se muestra el diagrama lógico de la base

de datos correspondiente al módulo Control de armamento, equipos y medios de seguridad, generado a partir de la definición de las entidades persistentes.

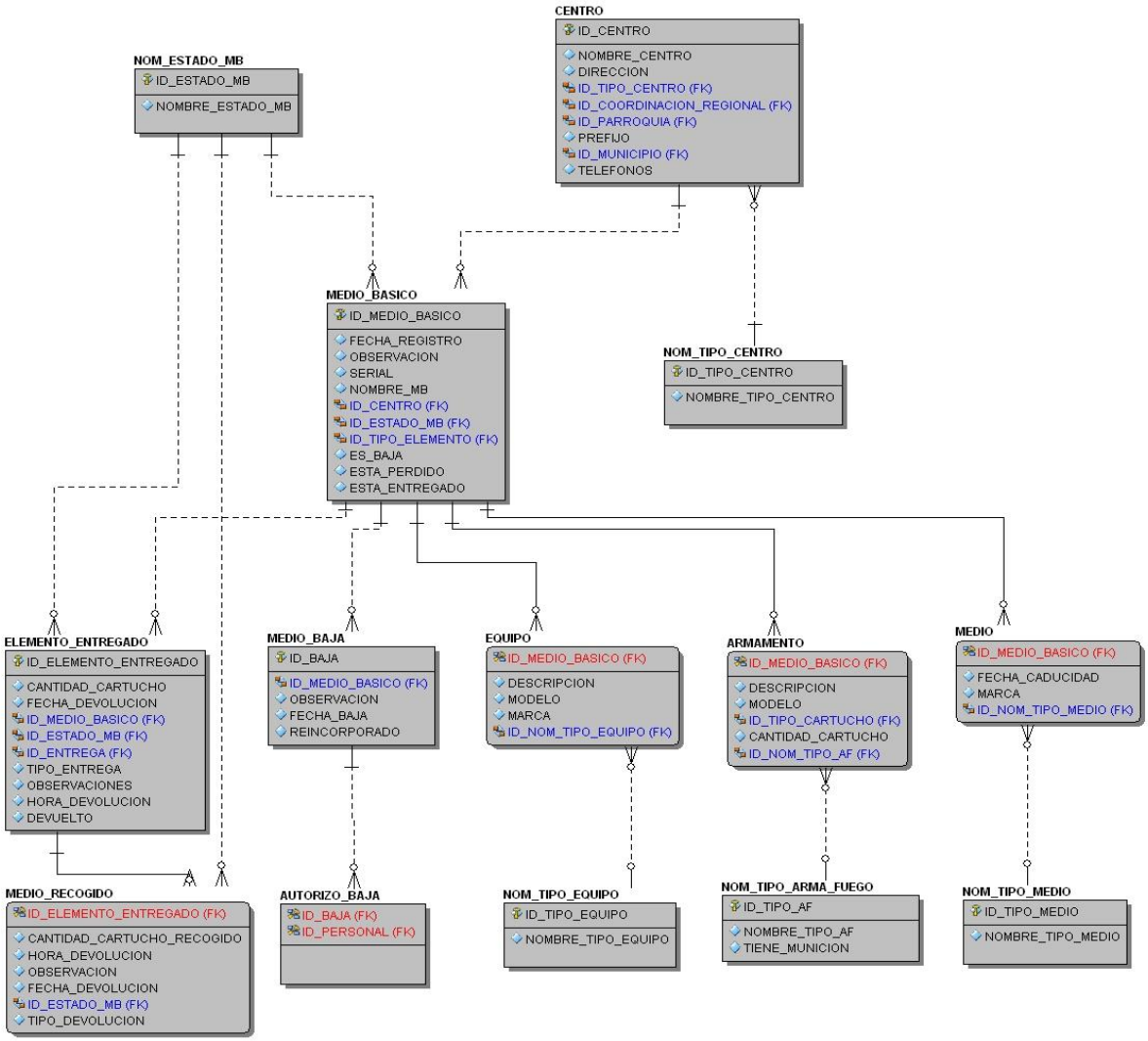


Figura 2.2 Diagrama entidad relación del módulo Control de armamentos, equipos y medios de seguridad

El modelo de datos del módulo Control de armamento, equipos y medio de seguridad, se representa en el **Anexo 2.2**.

2.3.3 Diseño de la capa de acceso a datos

Dentro de la capa de acceso a datos se diseñan todas las funcionalidades que están ligadas directamente con el gestor de base de datos, funcionalidades que son necesarias y complementan las funcionalidades de la capa de negocio. Como se explicó anteriormente, esta capa aísla a la capa de negocio de la tecnología de persistencia, por lo que todo su diseño se basa en el objetivo principal de que exista el menor acoplamiento posible entre nuestra aplicación y el gestor de base de datos Oracle. En el cumplimiento de este requisito un papel muy importante lo tiene el uso del framework ORM Hibernate. Este framework además de facilitar un fácil mapeo entre el lenguaje relacional de la base de datos y el lenguaje de objetos de nuestra aplicación, brinda una serie de interfaces básicas muy útiles a la hora de diseñar los métodos más comunes que se realizan sobre los datos persistidos.

En el diseño de esta capa se definen una serie de interfaces que, contienen las funcionalidades necesarias relacionadas con la persistencia y recuperación de datos del medio de almacenamiento, además de ser las encargadas de la comunicación con la capa de negocio, estas son: ElementoSeguridadProteccionDAO, BajaDAO, DevoluciónDAO, ElementoEntregadoDAO, EntregaDAO y PerdidaDAO. A la misma vez se definen las clases que implementarán las funcionalidades definidas en las interfaces, estas son: ElementoEntregadoDAOImpl, BajaDAOImpl, DevoluciónDAOImpl, EntregaDAOImpl, PerdidaDAOImpl y ElementoSeguridadProteccionDAOImpl. El uso de estas interfaces dentro de la aplicación es muy importante, ya que la independiza de la tecnología de persistencia que se usa en la actualidad para el proyecto, Oracle, o de algún cambio que esta tenga en el futuro. En el diseño propuesto los objetos del negocio están compuestos por interfaces y no por los objetos que implementan las funcionalidades sobre el gestor de base de datos. Este diseño permite que si hubiera cambios sobre la persistencia de datos, solo sería necesario implementar otro objeto que implementara las funcionalidades de la interface ahora con las nuevas peculiaridades del gestor de base de datos, y los cambios dentro de la capa de negocio serán nulos.

Las implementaciones de los DAOs heredan de la clase ***cu.uci.arbaweb.dataaccess.dao.impl.AbstractBaseDao*** proporcionada por ArbaWeb que a su vez extiende la clase hibernate3.support.HibernateDaoSupport e implementa métodos comunes para todos los DAOs como son findById, persist, delete, findByCriteria y findByExample.

En la Figura 2.3 se muestra el diseño de la capa de acceso a datos:

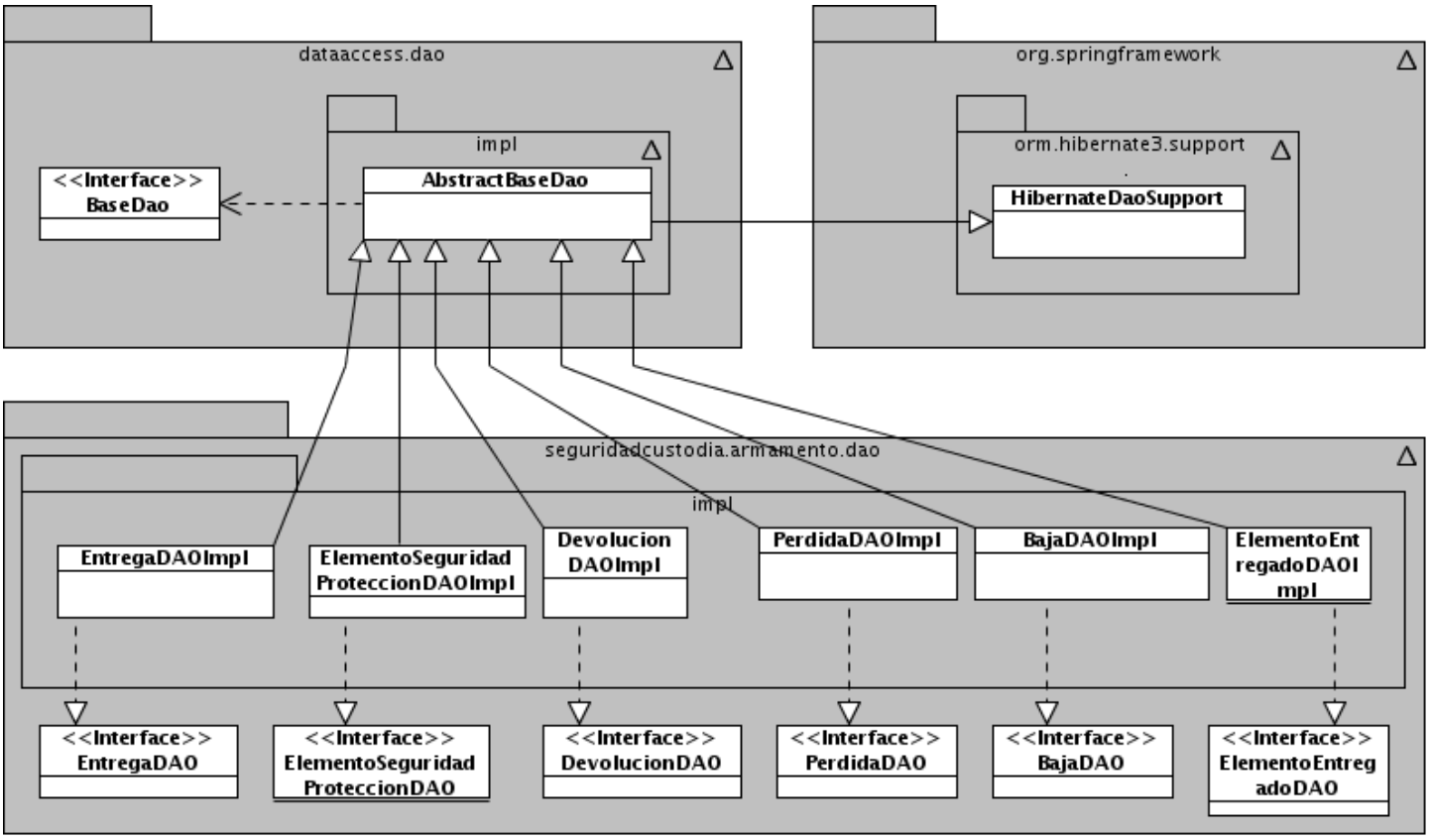


Figura 2.3 Estructura de la capa de acceso a datos

2.3.4 Diseño de la capa de negocio.

En la capa de negocio se diseñan todas las funcionalidades que, realiza el módulo Control de armamento, equipos y medios de seguridad, contiene toda la lógica de negocio. Para cubrir todas estas funcionalidades se definió durante el diseño de esta capa las siguientes interfaces: BajaManager, PerdidaManager, EntregaManager, DevoluciónManager, ElementoSeguridadProteccionManager. A la misma vez se definieron las clases que implementarán todas las funcionalidades contenidas en las interfaces, las mismas son: BajaManagerImpl, EntregaManagerImpl DevoluciónManagerImpl, ElementoSeguridadProteccionManager, PerdidaManagerImpl.

Los objetos de negocio (managers) fueron definidos de acuerdo al agrupamiento lógico de funcionalidades que respondan a procesos de negocio, los manager contienen las funcionalidades que representan el negocio sobre una o varias entidades del dominio.

Las funcionalidades que implementa los objetos del negocio, son expuestas a la capa de presentación a través de una fachada definida en el módulo de armamento: ArmamentoFacade. La fachada está compuesta por todas las interfaces definidas en el diseño y no por los objetos de negocio que implementan estas funcionalidades. Este diseño permite aislar a la fachada de las implementaciones. La fachada no implementa lógica de negocio, solo actúa como intermediario entre la capa de negocio y la capa de presentación, para implementar sus métodos solo hace falta saber qué manager tiene la información que necesita y utilizarla.

En la Figura 2.4 se muestra el resultado final del diseño de la capa de negocio

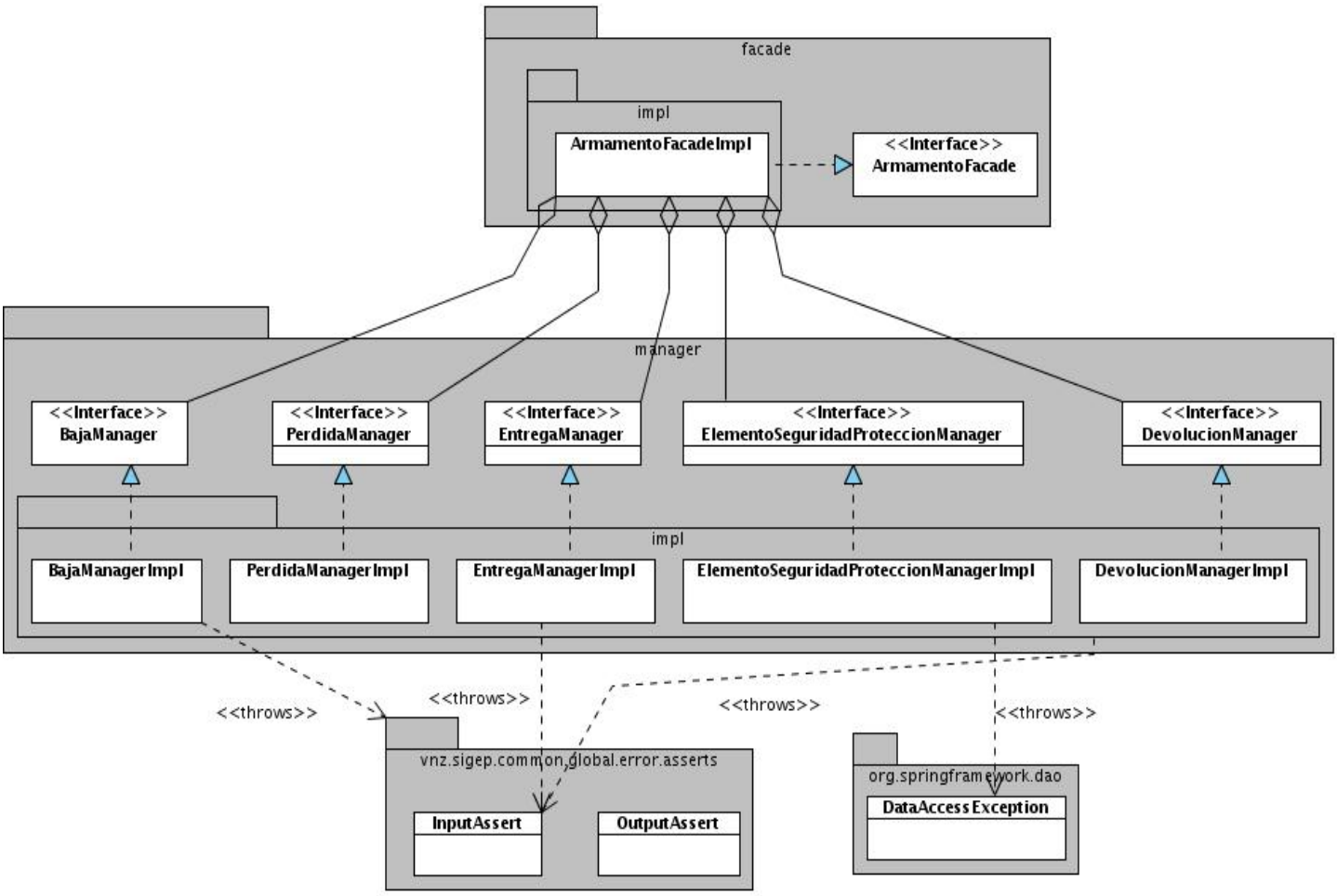


Figura 2.4 Estructura de la capa de negocio

2.3.5 Diseño de la capa de presentación.

En la construcción del diseño de la capa de presentación de la solución propuesta para el módulo de armamento, se integran un conjunto de tecnologías de la cual se debe tener un amplio dominio para realizar el diseño más eficiente y reutilizable posible, entre estas tecnologías tenemos: HTML, JSP, JavaScripts y Spring-MVC.

Durante del diseño de la capa de presentación, interviene un componente fundamental y de gran importancia para la realización de cada funcionalidad, el **controlador.** Los controladores se encuentran dentro del contenedor de servlets Tomcat y son los responsables de atender cada petición que llega desde el cliente, construir cada una de las páginas cliente con los datos que estas necesiten o recolectar datos desde la misma para realizar las distintas funcionalidades.

Para realizar el diseño de todas las funcionalidades del módulo armamento, la cuales fueron analizadas anteriormente ***(****2.2 Análisis de las funcionalidades****.),*** fueron divididas de acuerdo a las áreas de procesos que abarca el módulo que son: Gestión de Armamentos, Equipos y Medios de Seguridad, Gestión de Entregas, Gestión de Pérdidas y Gestión de Bajas. De forma general para cada área se define un controlador general que atiende la mayoría de las peticiones que se realizan en la misma, manipula la información necesaria y se encarga de construir la páginas clientes correspondiente a cada petición con los datos que estas necesiten, para esto cada método del controlador que lo necesite devuelve un ModelAndView; el Model contiene los datos necesarios y la View (la vista) representa la página jsp que se debe mostrar. Esta página jsp puede representar por sí sola una página cliente o en otros casos como se representa en los diagramas de clases del diseño puede integrarse con otras páginas jsp y formar la página cliente.

Los controladores generales definidos fueron: RegistrarArmamentosMultiActionController, EntregasEjecuciónMultiActionController, PerdidasMultiActionController, BajasMultiActionController, respectivamente. Además se definen otros controladores que atenderán otras peticiones de menor complejidad dentro del sistema como los controladores encargados de atender las peticiones de las tablas usadas, para cargar los datos requeridos desde la base de datos.

Los controladores definidos heredan de clases del módulo Spring-MVC del framework Spring, según las funcionalidades que realizarán; estas clases brindan implementaciones básicas para que los controladores realicen sus funciones. Las clases de Spring-MVC más utilizadas son: Controller, AbstractController MultiActionController, SimpleFormController, AbstractLoadTableController, AbstractLoadFilteredTableController, del paquete org.springframework.web.servlet.mvc.

Los controladores no ejecutan las tareas lógicas del negocio sino que las delegan en otras clases, con este objetivo los controladores están compuestos por objetos facade, implementando así el patrón Facade, con lo que se logra un modelo de alta cohesión.

Las facades usadas en este diseño son: la facade del módulo Armamento (ArmamentoFacade), y las facades definidas en el global de la aplicación que implementan funcionalidades comunes para todo el proyecto SIGEP (GlobalFacade y NomencladoresFacade).

Otro componente dentro del diseño de la capa de presentación son los ficheros JavaScripts. Los JavaScripts implementarán funciones del lado del cliente, enviarán y validarán datos de entradas y salidas hacia el contenedor de servlest Tomcat, además, utilizando el framework Dojo se encargarán de proporcionarles los aspectos estéticos y de formato, así como los componentes necesarios a la interfaz de usuario. Por lo general cada página cliente está asociada a un fichero JavaScripts.

2.4 Implementación módulo Control de armamento, equipos y medios de seguridad del SIGEP

2.4.1 Implementación de las entidades del dominio

Las entidades del dominio suelen tener muy poco comportamiento. Los métodos *equals(), hasCode()* y *toString()* deben ser implementados en la mayoría de los casos, ya que son frecuentemente utilizados. En este paso se debe refinar la definición de todos los atributos de las entidades de manera que queden listas en un buen por ciento para implementaciones reusables.

2.4.2 Implementación de la capa de acceso a datos.

La implementación de la capa de acceso a datos se basa en dos tareas fundamentales, la creación de los ficheros de mapeo de Hibernate (hbm.xml) y la implementación de las funcionalidades de los DAOs[5].

Los ficheros hbm.xml son usados por el framework ORM Hibernate, para mapear los campos entre las tablas del gestor de base de datos y los objetos del dominio de nuestra aplicación. Para la creación de estos importantes ficheros, es de gran ayuda el plug-in de Eclipse, Hibernate Tools que, acelera esta tarea al brindar generación de código en algunos momentos durante la confección de estos ficheros. Según la estructura de paquetes utilizada los ficheros. hbm se colocan en el paquete dao.impl.map.

[5] Data Access Object, Objeto que implementa la lógica de acceso a datos.

A continuación se muestra un fragmento del fichero Baja.hbm.xml, el cual mapea los campos entre la tabla Medio_Baja de la base de datos y la clase del dominio Baja, para ser usada por la aplicación

```
<hibernate-mapping
    package="vnz.sigep.seguridadcustodia.armamento.domain">
    <class name="Baja" table="MEDIO_BAJA">
        <id name="id" type="string">
            <column name="ID_BAJA" length="20"/>
            <generator
                class="vnz.sigep.common.global.util.dataaccess.id.PrefixPersistentGenerator">
                <param name="sequence">seq_BAJA</param>
                <param name="maxlo">1000</param>
            </generator>
        </id>

        <many-to-one name="ElementoSeguridadProteccion" class="ElementoSeguridadProteccion"
        fetch="join" lazy="false" outer-join="true">
            <column name="ID_MEDIO_BASICO" length="20" not-null="true" />
        </many-to-one>

        <property name="motivo" type="string">
            <column name="OBSERVACION" />
        </property>

        <property name="reincorporado" type="boolean">
            <column name="REINCORPORADO" precision="1" scale="0" />
        </property>
```

Figura 2.5 Configuración del fichero Baja.hbm.xml

Para la implementación de las funcionalidades de los objetos de acceso a datos se usa en primera opción los APIs Criteria y Example del Framework Hibernate. Estos APIs permiten la creación de consultas al gestor de base de datos de gran complejidad en la que intervienen varias tablas, de forma fácil para los desarrolladores sin la necesidad de introducir código en lenguaje SQL. En algunos casos en que estos APIs no son suficiente para darle cumplimiento a una funcionalidad se utiliza también el lenguaje de consultas de Hibernate(HQL), este lenguaje permite escribir consultas pero refiriéndose a las tablas de la base de datos como objetos con atributos (orientadas a objetos), por lo que permite mayor flexibilidad y mas libertades al implementador. Además de que siempre se pueden escribir las consultas al gestor de base de datos en lenguaje SQL. Siempre se recomienda el uso de las APIs mencionadas y del HQL porque permite escribir código de mayor comprensión, más sencillo y menos complejo, transparente para el resto del equipo de desarrollo y fácil de mantener.

Las implementaciones de los DAO se guardan en la carpeta dao.impl, ejemplos de implementaciones usando los APIs Criteria y Example del Framework Hibernate y el Lenguaje de consultas de Hibernate (HQL), se muestran en el **Anexo 2.4**.

Después de implementar los DAOs el siguiente paso es configurar el fichero de Spring, sigep-seguridadcustodia-armamento-dataaccess-context.xml. Esta configuración permite crear el objeto DAO en el contexto de Spring para luego ser inyectado a los objetos del negocio a través de la inyección de dependencia de Spring, además de inyectar a los objetos de acceso a datos el sessionFactory de Hibernate, que es el objeto que configura al framework Hibernate a partir de los ficheros de mapeo (hbm.xml) y el dialecto a utilizar, según el gestor de base de datos

En la Figura 2.6 se muestra un ejemplo de la configuración de un DAO en este fichero:

```
<bean id="bajaDAO"
    class="vnz.sigep.seguridadcustodia.armamento.dao.impl.BajaDAOImpl">
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>
```

Figura 2.6 Configuración de la clase BajaDAO en el fichero sigep-seguridadcustodia-armamento.dataaccess-context.xml

2.4.3 Implementación de la capa de negocio.

Durante esta actividad se implementan en los objetos de negocio los métodos definidos para cada una de las interfaces de los managers ajustándose a las funcionalidades previstas.

Cada método al ser implementado debe de verificar la integridad de los datos de entrada y de salida, e informar a la capa de interfaz de usuario de cualquier eventualidad a través de las excepciones definidas, las cuales serán capturadas por la capa de presentación y mostradas al usuario de una forma adecuada.

Las excepciones usadas en el módulo armamento son excepciones generales definidas por SIGEP y las de las librerías del lenguaje java, entre las usadas se encuentra: *DataAccessException* del paquete *org.springframework.dao* y *Exception* del paquete *java.lang* como excepción general para capturar excepciones lanzadas en la capa de acceso a datos como *java.sql.SQLException* y *org.hibernate.HibernateException.*

La fachada y los managers implementados se configuran en el fichero de configuración del contexto de Spring sigep-seguridadcustodia-armamento-business-context.xml. En este fichero se inician los objetos del negocio

en el contexto de Spring, se inyectan a cada uno de los managers a través de la inyección de dependencia de Spring los DAOs necesarios para realizar sus funcionalidades, así como la inyección a la fachada de los todos los manager que debe conocer para exponer todas las funcionalidades del módulo.

En las Figuras 2.7 y 2.8 se muestran ejemplos de la configuración de la fachada y del manager ElementoSeguridadProteccionManager en este fichero.

```
<bean id="elementoSeguridadProteccionManager"
    class="vnz.sigep.seguridadcustodia.armamento.manager.impl.ElementoSeguridadProteccionManagerImpl">
    <property name="elementoSeguridadProteccionDAO">
        <ref bean="elementoSeguridadProteccionDAO" />
    </property>
    <property name="entregaDAO">
        <ref bean="entregaDAO" />
    </property>

</bean>
```

Figura 2.7 Configuración del manager ElementoSeguridadProteccion en el fichero sigep-seguridadcustodia-armamento-bussiness-context.xml

```
<bean id="armamentoFacade"
    class="vnz.sigep.seguridadcustodia.armamento.facade.impl.ArmamentoFacadeImpl">
    <property name="elementoSeguridadProteccionManager">
        <ref bean="elementoSeguridadProteccionManager" />
    </property>
    <property name="entregaManager">
        <ref bean="entregaManager" />
    </property>
    <property name="bajaManager">
        <ref bean="bajaManager" />
    </property>
    <property name="personalManager">
        <ref bean="personalManager" />
    </property>
    <property name="devolucionManager">
        <ref bean="devolucionManager" />
    </property>
    <property name="perdidaManager">
        <ref bean="perdidaManager" />
    </property>
</bean>
```

Figura 2.8 Configuración de la fachada en el fichero sigep-seguridadcustodia-armamento-bussiness-context.xml

2.4.3.1 Transacciones

Las transacciones son un elemento de gran importancia durante la implementación de la capa de negocio, de ellas depende que los datos persistentes se mantengan en un estado consistente. Cada método definido en las interfaces de la capa de negocio generalmente se debe ejecutar como una transacción para garantizar la consistencia de la Información en la base de datos.

Para lograr esta tarea se usan las facilidades que brinda Spring para el manejo de funcionalidades como transacciones. Spring, para lograr la ejecución transaccional de las funcionalidades implementadas en los objetos de negocio, se integra con el framework Hibernate y utiliza el soporte para transacciones que brinda esta tecnología de persistencia.

La clase org.springframework.orm.hibernate3.HibernateTransactionManager une una sesión de Hibernate al hilo de ejecución e implementa los métodos getTransaction, commit y rollback definidos en la interface org.springframework.transaction.PlatformTransactionManager. A través de estos tres métodos se crea una transacción o se accede a la transacción en curso y se hace commit (aceptar) y rollback (fallar) a las transacciones. La instancia de HibernateTransactionManager requiere de una referencia al sessionFactory de Hibernate. Spring provee de un soporte para la gestión de transacciones de forma declarativa. Las transacciones se declaran en el fichero de configuración de su contexto y son aplicadas en forma de aspectos. De esta forma hay que escribir muy poco o ningún código de manejo de transacciones. Con la etiqueta ***<aop:advisor>*** se declara el pointcut (punto en que se aplicará el aspecto, generalmente una expresión regular que representa un conjunto de métodos de una clase). Con la etiqueta ***<tx:advice>*** se declara el aspecto específico o sea, cómo se va a aplicar la transacción.

En la Figura 2.9 se muestra un ejemplo de la aplicación de transacciones de forma declarativa a un objeto de negocio:

```
<aop:config>
    <aop:advisor
        pointcut="execution(* *..*ElementoSeguridadProteccionManagerImpl.*(..))"
        advice-ref="elementoSeguridadProteccionMgrAdvice" />
</aop:config>
<tx:advice id="elementoSeguridadProteccionMgrAdvice">
    <tx:attributes>
        <tx:method name="*" rollback-for="java.lang.Exception" />
    </tx:attributes>
</tx:advice>
```

Figura 2.9 Aplicación de transacciones al objeto ElementoSeguridadProteccionImpl

En el ejemplo se aplican las transacciones a todos los métodos del objeto ElementoSeguridadProteccionImpl. Con esta configuración los métodos de la clase ElementoSeguridadProteccionImpl se ejecutarán en contextos transaccionales sin siquiera ser conscientes de ello.

La definición del punto donde se aplica la transacción está escrita utilizando la sintaxis de AspectJ (Lenguaje de programación orientado a aspectos, construido como una extensión del lenguaje Java.). La expresión execution significa cuando el método sea ejecutado. La expresión entre paréntesis representa los métodos a los que se aplicará la transacción, en este caso: **...*ElementoSeguridadProteccionImpl.*(...). El primer * significa cualquier tipo de retorno; la expresión *...* ElementoSeguridadProteccionImpl significa cualquier clase cuyo nombre termine en ElementoSeguridadProteccionImpl y la expresión.*(...) significa cualquier método con cualquier parámetro. Al no especificarse el nivel de aislamiento de la transacción mediante el atributo isolation de la etiqueta <tx:method>, se toma el nivel de aislamiento por defecto del medio de almacenamiento, que en el caso de Oracle es lectura confirmada (read committed) que evita la lectura de datos sucios. Con el atributo rollback-for se definen las excepciones que, en caso de ser lanzadas, se desea que hagan fallar (rollback) la transacción.

2.4.4 Implementación de la capa de presentación

La implementación de la capa de presentación consta de dos momentos significativos: la implementación de los controladores y la construcción de la interfaz de usuario asociada a cada petición, junto con los ficheros JavaScripts.

2.4.5 Implementación de los controladores

La implementación de los controladores es de gran importancia porque manejan todo el flujo web de la aplicación. Atienden cada petición que llega desde el cliente y muestran las vistas necesarias, formatean los

datos de salidas que son utilizados para construir cada una de las páginas cliente y validan los datos de entrada de la aplicación que llegan desde el cliente para realizar las distintas funcionalidades.

2.4.6 Implementación de las páginas JSP

Como se explicó durante la realización del diseño de la capa de presentación, las páginas JSP construyen los documentos HTML que serán las páginas clientes que son mostradas al usuario. Por lo general cada página JSP construye una página cliente o, en ocasiones se juntan un número de ellas para formar una página cliente. Durante la implementación de las páginas JSP, se usan la tecnología JSTL (Java Standard Tag Library), conjunto de librerías de etiquetas simples y estándares muy útil a la hora de realizar ciertas operaciones sobre los datos que son enviados junto con la página JSP desde el controlador y las etiquetas de Spring. Para el diseño gráfico de las JSP usadas se utilizan las hojas de estilo en cascada (Cascading Style Sheets, CSS) definidas para cada uno de los componentes dentro de SIGEP.

2.4.7 Implementación de la lógica en el cliente

Aparejado a la implementación de las páginas JSP, se realiza la implementación de la lógica en el cliente; para esta actividad se usa el lenguaje JavaScript. Es durante esta implementación donde se realizan las validaciones del lado del cliente que sean necesarias y se programa el comportamiento de los componentes gráficos que se utilizan dentro de las páginas JSP, tales como: tablas, botones, pantallas emergentes de error o información al usuario y calendarios entre otros. Los componentes gráficos usados pertenecen a la biblioteca JavaScript DojoToolkit.

Se usan además las tecnologías AJAX y JSON-RPC para el envió de peticiones al servidor de una forma mucho más rápida.

2.4.8 Implementación de la funcionalidad Registrar Entrega

Dado el diseño de la capa de presentación, el flujo de esta funcionalidad comienza en la página de gestión de las entregas realizadas. Se decide registrar una nueva entrega, se construye la interfaz de usuario necesaria para registrar los datos de la entrega: fecha de entrega, hora de entrega, funcionario a quien se le hace la entrega y los datos de los medios de seguridad que serán entregados.

Las peticiones serán atendidas por el controlador RegistrarEntregaMultiActionController. Este controlador hereda de la clase de Spring-MVC org.springframework.web.servlet.mvc.MultiActionController que, extiende la capacidad para atender múltiples peticiones; en otras palabras equivale a implementar el flujo de varios Controller en un solo controlador a través de un fácil mapeo en el fichero servlet.xml de Spring, en el cual se mapea el nombre de la petición con el controlador que la va a atender y el nombre del método del controlador que atenderá esa petición en específico.

En la Figura 2.10 se muestra la configuración del controlador RegistrarEntregaMultiActionController dentro del fichero servlet.xml de Spring, la configuración de las facades que componen este controlador, las cuales serán inyectadas al controlador a través de la inyección de dependencia de Spring y la peticiones que este atiende, así como los métodos específicos para cada petición.

```
<bean id="registrarEntregaMultiActionController"
        class="vnz.sigep.seguridadcustodia.armamento.web.RegistrarEntregaMultiActionController">
        <property name="methodNameResolver">
      <ref bean="methodNameResolverEntrega" />
        </property>
        <property name="armamentoFacade">
            <ref bean="armamentoFacade" />
        </property>
        <property name="nomencladoresFacade">
            <ref bean="nomencladoresFacade" />
        </property>
    </bean>
    <bean id="methodNameResolverEntrega"
        class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver">
        <property name="mappings">
            <props>
            <prop key="/obtenerCantidadPerdidasActivasPorPersonal.htm">
     obtenerCantidadPerdidasActivasPorPersonal
            </prop>
            <prop key="/registrarEntrega.htm">obtenerPáginaRegistrarEntrega</prop>
            <prop key="/obtenerElementosEntregadosByEntrega.htm">
            obtenerElementosEntregadosByEntrega
            </prop>
        <prop key="/conocerLlevaMunicion.htm">conocerLlevaMunicion</prop>
<prop key="/obtenerFuncionario.htm">obtenerFuncionario</prop>
    <prop key="/obtenerCantidadPerdidasActivasPorPersonal.hmt">
        obtenerCantidadPerdidasActivasPorPersonal
            </prop>
```

```
    <prop key="/modificarEntrega.htm">obtenerPáginaModificarEntrega</prop>
            <prop key="/obtenerEntregaModificar.htm">obtenerEntrega_a_Modificar</prop>
            <prop key="/obtenerTipoElementos.htm">obtenerTipoElementos</prop>
    <prop key="/obtenerEstadoElemento.htm">obtenerEstadoElemento</prop>
            <prop key="/persistirEntrega.htm">persistirEntrega</prop>
                    <prop key="/persistirEntregaModificada.htm">
                            persistirEntregaModificada
                    </prop>
        <prop key="/obtenerSerialesTipoElementoSeguridadProteccion.htm">
                    obtenerSerialesTipoElementoSeguridadProteccion
            </prop>
      </props>
            </property>
□       </bean>
```

Figura 2.10 Configuración del controlador RegistrarEntregaMultiActionController

En la Figura 2.11 se muestra el código donde el controlador devuelve el ModelAndView necesario para construir la interfaz de usuario de registrar entrega.

```
public ModelAndView obtenerPáginaRegistrarEntrega(
                    HttpServletRequest request, HttpServletResponse response)
                    throws Exception {
            Map model = new HashMap();
            Date fecha = new Date();
            SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
            model.put("fecha", format.format(fecha));

            return new ModelAndView("registrarEntrega", "model", model);    }
```

Figura 2.11 Método ModelView del controlador RegistrarEntregaMultiActionController

La página cliente de *Registrar Entrega* realiza múltiples peticiones al contenedor de servlets Tomcat, en busca de datos para facilitar la actividad de registrar la entrega. Una vez seleccionada la denominación del nuevo medio de seguridad que será entregado (Arma de fuego, Equipo o medio), se realiza una petición al contenedor de servlets Tomcat mediante la tecnología AJAX en busca de los tipos de la denominación que haya sido seleccionada, con estos datos se llena el select que representa el campo de Tipo; una vez seleccionado el tipo del elemento que será entregado, se realiza una nueva petición en busca de los números de serial de los medios de este tipo que están disponibles en la base de datos para ser mostrados en otro select y por último, después de haber sido seleccionado el elemento a través de su número de serie, se realiza otra petición en busca del estado del mismo para ser mostrado. En caso de que el estado del elemento no sea el óptimo, se le muestra un mensaje de alerta al usuario.

En la Figura 2.12 se muestra el diagrama de secuencia para esta funcionalidad.

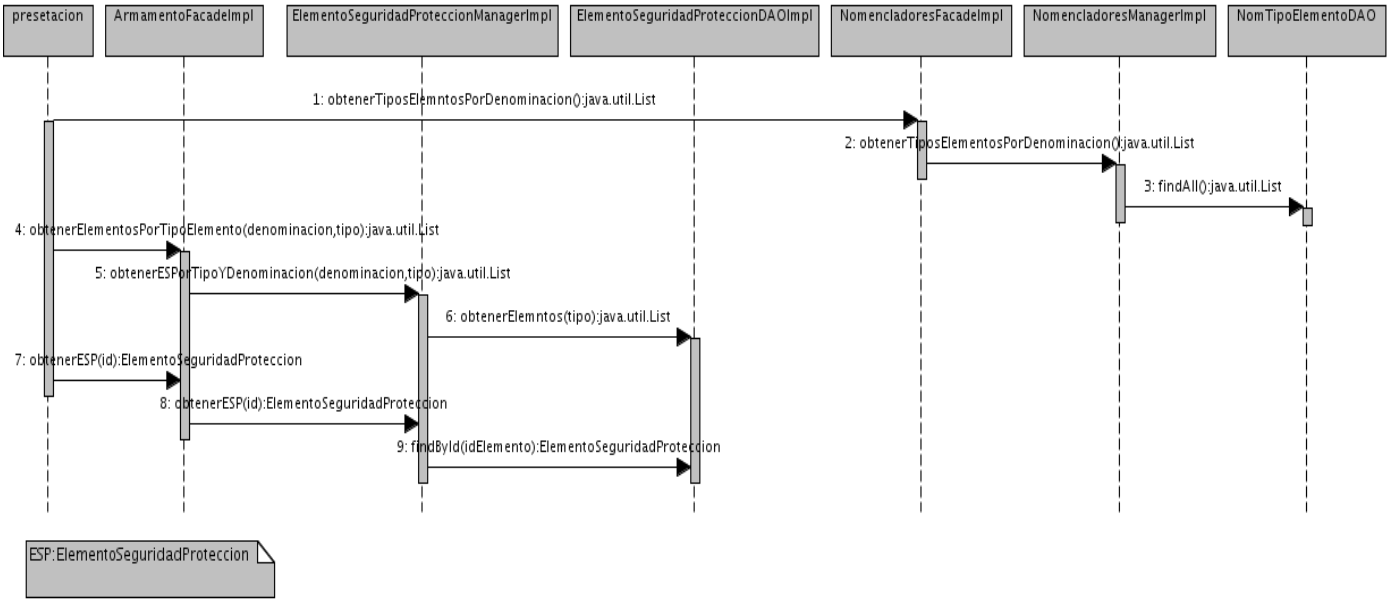


Figura 2.12 Diagrama de Secuencia de una Nueva Entrega

En el **Anexo 2.5** se muestra la implementación del método persistirEntrega en el controlador RegistrarEntregaMultiActionController, el cual será invocado una vez haber ingresado los datos de una nueva entrega correctamente.

2.4.9 Implementación de la funcionalidad Registrar Devolución

Para registrar una devolución, se selecciona la entrega en ejecución que se desea culminar en la interfaz de usuario de gestión de entregas. Una vez seleccionada la entrega, se pasa a la devolución de forma individual de cada uno de los medios de seguridad que fueron entregados. Los medios de seguridad entregados pueden ser registrados como devueltos o como perdidos, durante el proceso de devolución.

Las peticiones necesarias para la realización de esta funcionalidad serán atendidas por los controladores: EntregaEjecuciónMultiActionController para crear la interfaz de usuario de registrar devolución, RegistrarDevolucionSimpleFormController para registrar un medio entregado como devuelto y el controlador PerdidasMultiActionController si se necesita registrar algún medio como perdido.

El controlador RegistrarDevolucionSimpleFormController dentro del proceso de registrar un medio como devuelto se encarga tanto de atender la petición para crear la interfaz de usuario de devolver medio de seguridad, como de la petición para registrar la devolución del medio en la base de datos. Este controlador hereda de la clase *org.springframework.web.servlet.mvc.SimpleFormController.* Esta clase, como su nombre lo indica, es muy útil para trabajar con formularios. La implementación de un controlador de este tipo implica la implementación de otro componente, el objeto command, que representa el objeto mostrado en el formulario y contendrá los atributos que presenta el formulario. Es de gran importancia porque se encarga de mostrar datos inicialmente en los campos del formulario que lo requieran y de recoger los datos del formulario, mediante los métodos set que serán implementados en el command para cada uno de los atributos. El SimpleFormController permite extender una serie de implementaciones básicas (métodos), muy útiles para atender y realizar las peticiones antes mencionadas: el método referenceData(), es llamado antes de que el formulario sea procesado y se encarga de agregar el modelo de datos que el formulario necesita utilizar; el método initBinder() permite registrar editores personalizados para ciertos campos de nuestro objeto command, como es el caso de los campos fechaDevolucion y HoraDevolucion que son de tipo *java.util.Date* pero el objeto *request* devuelve para estos campos un String por lo que, la transformación lo realiza el editor personalizado declarado aquí; el método onSubmit() se encargará del registro en la base de datos una vez que se le haya hecho submit() al formulario; el método formBackingObject() se ejecuta la primera vez al cargarse la página en ese momento se utiliza generalmente para inicializar los valores de los campos del formulario que lo requieran utilizando el objeto command, la segunda vez que se ejecuta el método es cuando se le da submit() al formulario y en esa oportunidad se utiliza para implementar las transformaciones necesarias al objeto *command* antes de ejecutar el método onSubmit().

En las siguientes figuras se muestran las interfaces de usuario que intervienen en el registro de una devolución:

Figura 2.13 Pantalla de Registrar Devolución

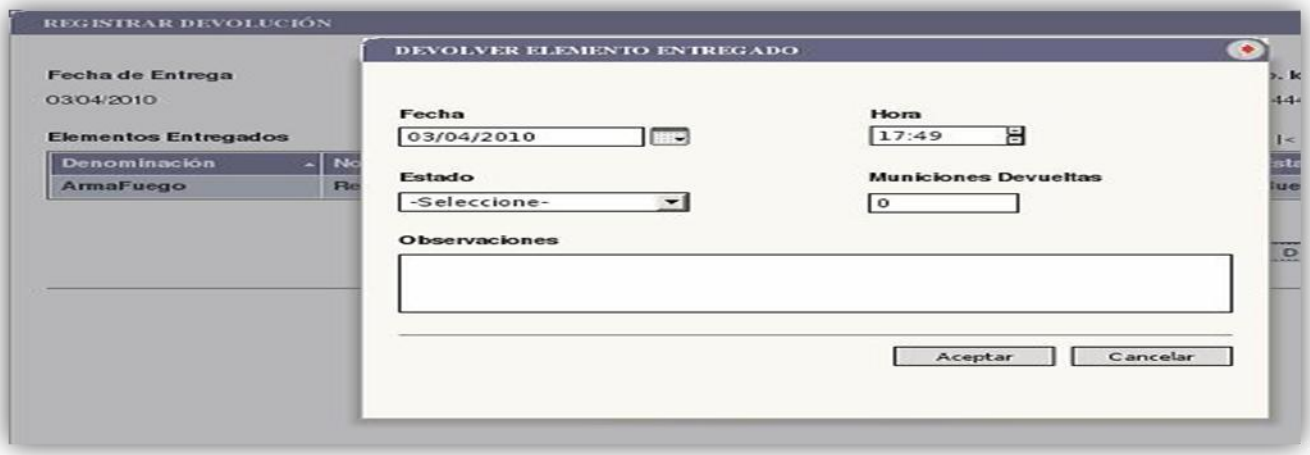


Figura 2.14 Pantalla Devolver Elemento Entregado

En el **Anexo 2.6** se encuentra la implementación de la clase RegistrarDevolucionSimpleFormController.

CAPÍTULO 3: VALIDACIÓN DEL SOFTWARE

3.1 Introducción

En este capítulo trataremos las pruebas de aceptación del cliente realizadas para la validación del módulo Control de armamento, equipos y medios de seguridad del proyecto SIGEP. Las pruebas se desarrollarán en dos tipos: pruebas de aceptación parcial del cliente, finalizando cada iteración; y pruebas de aceptación final del cliente, al efectuarse cada entrega de las versiones correspondientes. Los artefactos, condiciones necesarias y documentos rectores de esta actividad, así como todo lo relacionado a su completo y efectivo cumplimiento se encuentran en el “Plan de Prueba de Aceptación de SIGEP” referenciada en (ARECIO)

A continuación se abordan los aspectos más significativos de este plan y la recolección y valoración de no conformidades y solicitudes de cambio de las pruebas de aceptación parcial y final del cliente.

3.2 Pruebas de aceptación parcial del cliente (validaciones)

Las pruebas de aceptación parcial del cliente son las encargadas de ir asegurando, paulatinamente y al final de cada iteración, el cumplimiento de los Procesos Elementales del Negocio, los Casos de Uso, los prototipos y lo planteado en el proyecto técnico. El alcance de dichas pruebas deberán ser acordadas con quince días de antelación por parte del equipo de desarrollo.

3.2.1 Objetivos generales de las pruebas

Los fines de esta prueba son:

- Asegurar que los objetivos trazados para la confección del producto previamente definido, así como el producto en sí, cumplan con el entregable dado a los clientes
- Asegurar el cumplimiento del alcance definido para cada prueba.
- Detectar la mayor cantidad posible de No Conformidades en el tiempo planificado.
- Documentar y analizar todos los detalles y resultados obtenidos.
- Garantizar que el criterio de aceptación esté dentro del rango de criterios permisiblemente tolerables.

3.2.2 Fases para la realización de las pruebas

Las pruebas se realizan de forma manual y se prueban todas las funcionalidades implementadas en el módulo según la metodología del cliente.

Estas pruebas se harán teniendo en cuenta los siguientes pasos:

- Organización de los escenarios de pruebas y capacitación del equipo de pruebas.
- Realización de pruebas de funcionalidad.
- Conciliación de los documentos entregables resultados de las pruebas.
- Discusión y aprobación del los cambios.
- Aprobación y firma de los mismos.

3.2.3 Pruebas funcionales

La prueba de funcionalidad se enfoca en requerimientos para verificar que se corresponden directamente a casos de usos o funciones y reglas del negocio. Los objetivos de estas pruebas son verificar la aceptación de los datos, el proceso, la recuperación y la implementación correcta de las reglas del negocio. Este tipo de prueba se basa en técnicas de caja negra que consisten en verificar la aplicación y sus procesos interactuando por medio de la interfaz de usuario y analizar los resultados obtenidos.

Objetivo de la prueba

Asegurar la funcionalidad apropiada de los módulos, incluyendo el flujo de trabajo, entrada de datos, proceso y recuperación.

Técnica

Ejecutar cada proceso o función usando datos válidos, no válidos y casos críticos, para verificar lo siguiente:

- ¿Se obtienen los resultados esperados cuando se usan datos válidos?
- ¿Cuando se usan datos no válidos y casos críticos se despliegan los mensajes de error o advertencia apropiados?
- ¿Se aplica apropiadamente cada regla del negocio?

Criterio de aceptación

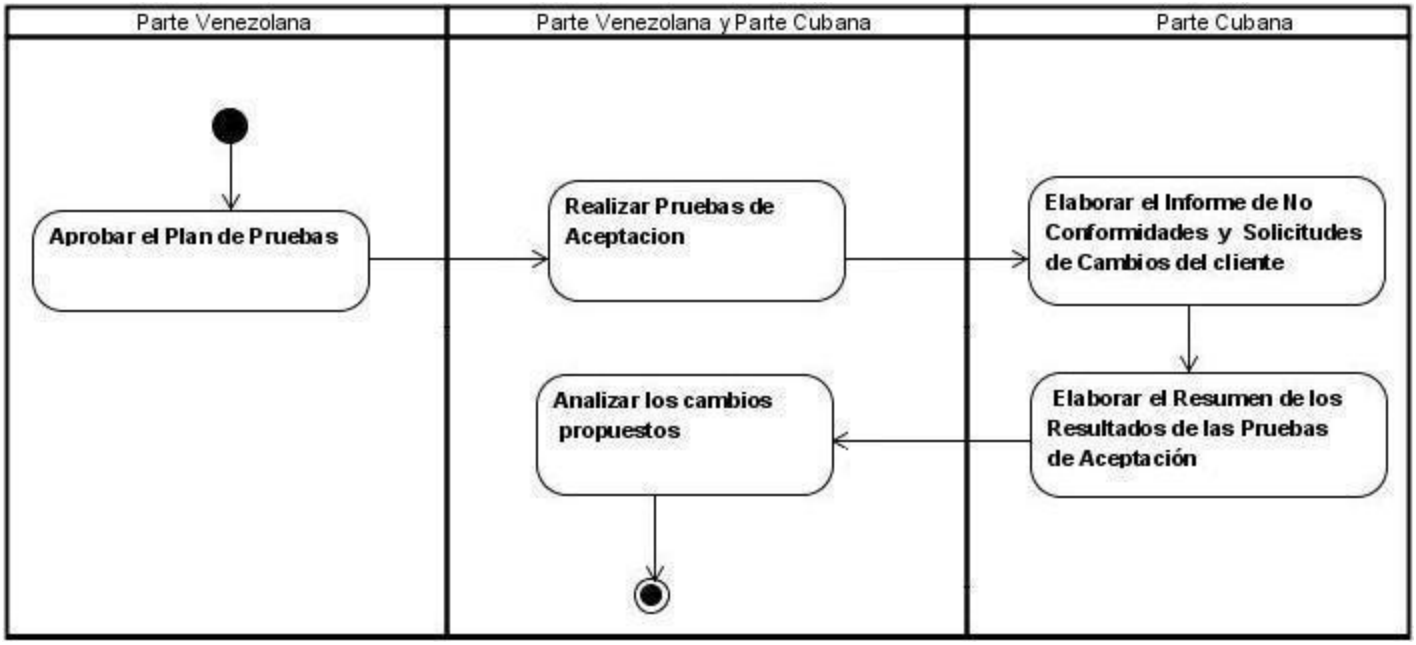
Todas las pruebas planificadas se concluyeron. Todos los defectos encontrados han sido debidamente identificados, notificados y trazadas las estrategias pertinentes para su erradicación. Las solicitudes de cambios han sido valoradas y asumidas.

3.2.4 Participantes en la prueba de aceptación parcial

PARTICIPANTE	CARGO	ESPECIALISTA
José Piñango	Técnico Superior (TSU)	
Luis Agüero	Técnico Superior (TSU)	
Carlos Garrido	Técnico Superior (TSU)	
Sharon DiGiusto	Abogado	
Gregory Rojas	Director de Custodia	X
Jose Piñango	Subdirector mínima de Carabobo	X

Tabla 1: Personas y cargo que ocupan, encargadas de la aceptación parcial del módulo.

3.2.5 Flujo de trabajo



3.2.6 Descripción del flujo de trabajo

Las pruebas de aceptación parcial del cliente se inician con la aprobación y firma del Plan de Pruebas que incluye el plan de trabajo detallado, la relación de los datos del personal que intervendrán en las pruebas y la selección de los módulos a probar. De manera opcional se puede asumir el paso donde se introduce a los

especialistas funcionales el uso de las funcionalidades de los módulos a probar.

Garantizadas todas las condiciones técnicas y verificadas por la dirección representante de la parte proveedora, se puede dar inicio al desarrollo de las pruebas de los módulos planificados. Para ello, intervienen el Especialista funcional cliente correspondiente y el Probador de forma directa con el software y como apoyo, el resto del equipo de desarrollo, esclareciendo las dudas de las funcionalidades que lo requieran. Durante las pruebas se verificará el cumplimiento de los Procesos Elementales del Negocio, los Casos de Uso y los prototipos correspondientes validados en la etapa de Captura de Requisitos, así como las funcionalidades de cada uno de los módulos planificados.

Las no conformidades del cliente que surjan durante las pruebas serán recogidas en un Informe de No Conformidades, las cuales serán analizadas con posterioridad por el equipo de desarrollo, donde se clasificará y valorará su respectiva respuesta. Las solicitudes de cambios que aparezcan serán recolectadas en su respectivo Informe de Solicitudes de Cambio, las cuales serán analizadas y llevadas a la mesa de negociaciones entre ambas partes; las aceptadas y pactadas pasarán a ser nuevos requisitos. Los resultados serán contemplados en el Resumen de los Resultados de las Pruebas y discutidos con las partes interesadas llegando a un acuerdo de aprobación y firma de dicho documento. De esta forma concluye el ciclo de las pruebas de aceptación parcial del cliente, genérico para cualquier caso que se aplique.

3.2.7 Acción de recolección y valoración de no conformidades y solicitudes de cambio

DESCRIPCIÓN	TIPO DE OBSERVACIÓN	ORIGEN	ESTADO
En la pantalla donde se registran las armas de fuego, quitar el campo "descripción" como requerido	Solicitud de Cambio	Calisoft	Resuelto
Mostrar un Historial con las armas de fuego, equipos y medios que han sido dados de baja	Nuevo Requisito	Calisoft	Resuelto
Permitir reintegrar un arma de fuego, equipo o medio que ha sido dado de baja	Nuevo Requisito	Calisoft	Resuelto
En la pantalla donde se registran las armas de fuego, poner el campo "modelo" como requerido	Solicitud de Cambio	Calisoft	Resuelto
Mostrar una alerta cuando un arma de fuego, equipo o medio no deba ser entregado por el estado en el que se encuentra (mal estado)	Nuevo Requisito	Calisoft	Resuelto

Imprimir un recibo como constancia de la devolución y la entrega	Nuevo Requisito	Calisoft	Resuelto
En los detalles del elemento entregado, mostrar las observaciones de la entrega y las observaciones de la devolución	Nuevo Requisito	Calisoft	Resuelto
Mostrar una alerta cuando se intente entregar algo a algún funcionario que haya perdido algo anteriormente	Nuevo Requisito	Calisoft	Resuelto
Reporte de Entregas y Devoluciones	Nuevo Requisito	Calisoft	Resuelto
Mover el módulo completo para el subsistema Seguridad y Custodia	Nuevo Requisito	Calisoft	Resuelto

Tabla 2: Descripción, tipo de observación, origen y estado de las solicitudes de cambio y no conformidades.

La prueba fue realizada por Calisof, empresa encargada de verificar la calidad en los proyectos de la UCI. De forma general los resultados fueron muy satisfactorios, a pesar de las no conformidades arrojadas. Las inquietudes o dificultades encontradas son de tipo Solicitud de Cambios y ninguna de tipo No Conformidad. Entre las inquietudes encontradas ocho, se clasifican según el alcance de la solicitud de cambios en Nuevo Requisito, que implica plantearse un nuevo requisito en la solución pactada y reajustar la misma.

3.3 Pruebas de aceptación final del cliente

Las pruebas de aceptación final del cliente son las encargadas de asegurar el cumplimiento de los Procesos Elementales del Negocio, los Casos de Uso, lo aceptado en los prototipos y lo planteado en el proyecto técnico.

3.3.1 Objetivos generales de las pruebas

- Ejecutar pruebas de software en un ambiente real y bajo la supervisión del equipo de desarrollo.
- Probar el rendimiento de la aplicación.
- Detectar casos críticos.
- Determinar la factibilidad de despliegue del software desarrollado.

- Valorar el impacto de la solución de software para el sistema penitenciario.

3.3.2 Fases para la realización de las pruebas

Todas las pruebas se realizarán de forma manual y se probarán todas las funcionalidades implementadas en el módulo según la metodología del cliente.

Estas pruebas se harán teniendo en cuenta los siguientes pasos:

- Organización de los escenarios de pruebas y capacitación del equipo de pruebas.
- Realización de pruebas de funcionalidad.
- Realización de las pruebas de interfaz de usuario.
- Se realizaran pruebas de regresión para chequear que los errores ya corregidos no se vuelvan a repetir y/o no se dañen funcionalidades que funcionaban correctamente.
- Se realizarán Pruebas de Seguridad y Control de Acceso con el objetivo de validar la protección de la aplicación sensible a entradas no deseadas.
- Realizar las pruebas de integridad de los datos y la base de datos.
- Se realizarán las pruebas de diseño informacional
- Realizar las pruebas de referencia cruzada
- Aplicación de la lista de chequeo y los principios de calidad.
- Conciliación de los documentos entregables resultados de las pruebas, aprobación y firma de los mismos.

3.3.3 Participantes en la prueba piloto por la parte cubana

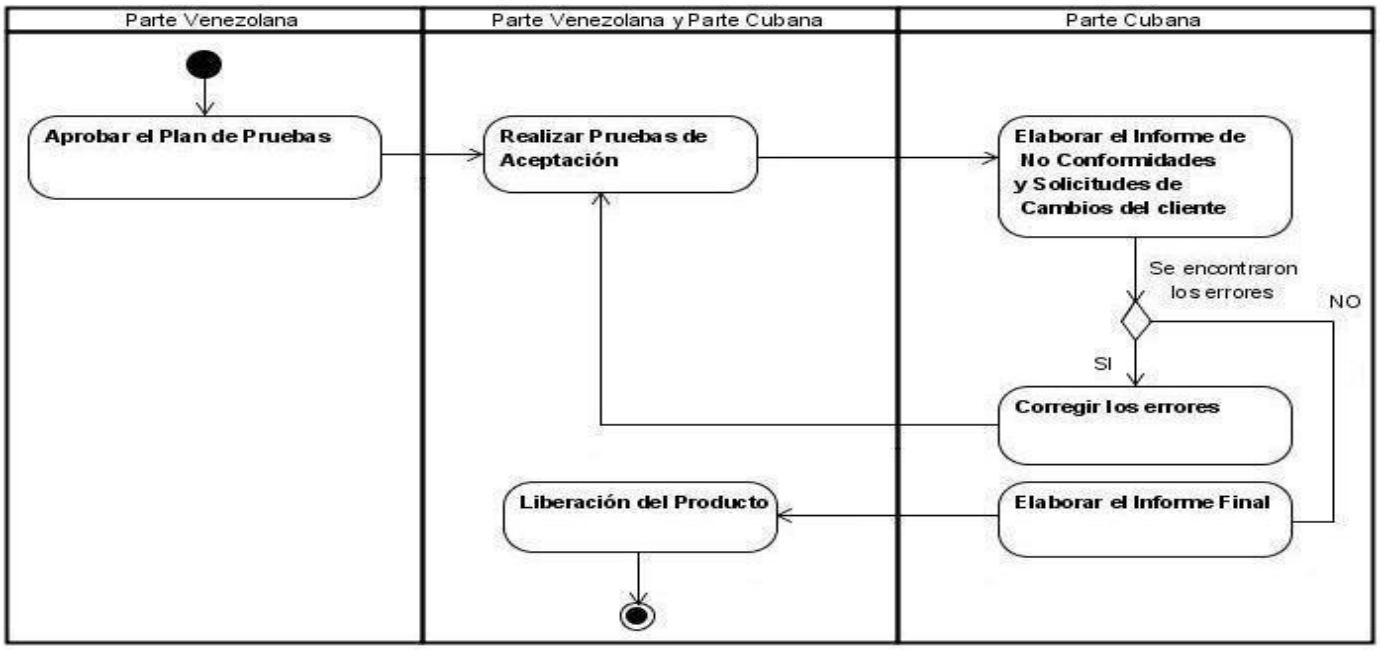
ESPECIALISTA	ROL
Coronel. Julio Serra Santiesteban	Gerente del Proyecto Humanización Penitenciaria. Componente cubano
Ing. Arturo César Arias Orizondo	Líder de subproyecto de software
Ing. Yandry Alberto Terry	Administrador de Base de Datos del SIGEP
Ing. Aracelis Reina Betancourt Cruz	Analista de software del SIGEP
Ing. Javier López del Castillo Caymares	Analista de software del SIGEP

Marisleidy Mora Castillo	Analista de software del SIGEP
Lic. Eddy Manuel Infante Alonso	Diseñador de Base de Datos
Ing. Maikel Pérez Martínez	Desarrollador de software del SIGEP
Eivys Hernández Suárez	Desarrollador de software del SIGEP
Ing. Geiser Arecio Pérez Rivas	Especialista de calidad. Calisoft

Tabla 3: Personas y rol que desempeñaban en el proyecto, que participaron en las pruebas piloto por la parte cubana

3.3.4 Flujo de trabajo

El flujo de trabajo de las pruebas de aceptación final del cliente se muestra en la figura que aparece a continuación y que muestra en un diagrama de actividades cómo interactúan cada una de las partes, venezolana y cubana, para validad el módulo realizando las actividades mostradas en el orden que exigen las flechas.



3.3.5 Descripción del flujo de trabajo

Las pruebas de aceptación final del cliente se inician con la aprobación y firma del Plan de Pruebas que incluye el plan de trabajo detallado, la relación de los datos del personal que intervendrán en las pruebas y la

selección de los módulos a probar. De manera opcional se puede asumir el paso donde se introduce a los especialistas funcionales al uso de las funcionalidades de los módulos a probar. Garantizadas todas las condiciones técnicas y verificadas por la dirección representante de la parte proveedora, se puede dar inicio al desarrollo de las pruebas de los módulos planificados.

Para ello, intervienen el Especialista funcional, el Cliente y el Probador de forma directa con el software y como apoyo, el resto del equipo de desarrollo, esclareciendo las dudas de las funcionalidades que lo requieran. Durante las pruebas se verificará el cumplimiento de los Procesos Elementales del Negocio, los Casos de Uso y los prototipos validados, así como las funcionalidades de cada uno de los módulos.

Las no conformidades del cliente que surjan durante las pruebas serán recogidas en un Informe de No Conformidades, las cuales serán analizadas y valorada su solución, que le será asignada al equipo de desarrollo para ser ejecutada. La respuesta se realiza de manera inmediata, brindado una nueva solución al cliente. Las solicitudes de cambios que aparezcan serán recolectadas en su respectivo Informe de Solicitudes de Cambio, las cuales serán analizadas y llevadas a la mesa de negociaciones entre ambas partes; las aceptadas y pactadas pasarán a ser nuevos requisitos.

Al concluir cada ciclo se reiniciará con una nueva revisión de la aplicación, partiendo de las No Conformidades pendientes a solución y de las solicitudes aceptadas, verificándose la solución dada y la integridad del sistema. Terminada la revisión se elabora un informe final sobre el resultado de las pruebas y se libera el producto a través de un Acta de Aceptación.

3.3.6 Acción de recolección y valoración de no conformidades y solicitudes de cambio

DESCRIPCIÓN	TIPO DE OBSERVACIÓN	ESTADO
El campo “Detalles” de la pérdida no se muestra correctamente.	No Conformidad	Resuelta
Al registrar una devolución, estando un elemento “Devuelto” se mostró como atrasada, al refrescar la pantalla se quitó la alarma.	No Conformidad	Resuelta
Cuando se registra la devolución, el estado en que es devuelto el elemento no actualiza el estado del elemento.	No Conformidad	Resuelta
Cuando se entrega un arma de fuego, no se habilita	No Conformidad	Resuelta

el campo cantidad de municiones.		
En el menú de Armamentos, hay URLs que en ocasiones no cargan la página correspondiente.	No Conformidad	Resuelta

Tabla 4: Descripción, tipo de observación y estado de las no conformidades.

Los resultados expuestos en la tabla anterior corresponden a la prueba piloto realizada al proyecto SIGEP, una vez concluida la implementación de su versión 2.0. Las inquietudes encontradas fueron de tipo No Conformidad ya que se originaron por una incorrecta implementación de un requisito previamente pactado. No se encontraron inquietudes de tipo Solicitud de Cambio. En general se encontraron 5 No Conformidad dentro del módulo de Armamento, las cuales fueron correctamente resueltas por el equipo de desarrollo encargado. Los resultados fueron satisfactorios y quedaron satisfechos los usuarios, especialistas e informáticos de la institución.

CONCLUSIONES

Se realizó el diseño, implementación y pruebas del módulo Control de armamentos, equipos y medios de seguridad del subsistema de Seguridad y Custodia de la versión 2.0 de la solución SIGEP.

Se utilizaron las herramientas previstas, se siguieron los flujos de trabajo definidos en el proyecto y se cumplió la arquitectura del sistema.

Todo lo anterior garantizó la integración del módulo a la solución sin comprometer la estabilidad de los módulos ya implementados y de la solución en su conjunto.

Se logró cubrir todas las funcionalidades definidas para el módulo y con los artefactos generados se cumplen con los objetivos trazados en el presente trabajo y para las tareas del proyecto.

Se realizaron pruebas de aceptación tanto parcial como final con el cliente de las cuales se obtuvieron resultados satisfactorios.

El módulo ha demostrado en el período de implantación y soporte, estabilidad en los requisitos y en la implementación realizada.

RECOMENDACIONES

Las pruebas realizadas en el Sistema Penitenciario de Seguridad Mínima de Carabobo se ejecutaron solamente con el volumen de datos generados en este centro. Por esta razón se recomienda que se realicen pruebas de rendimiento incorporándole datos de otros sistemas penitenciarios y/o en mayor tiempo de prueba para garantizar la integridad del módulo.

BIBLIOGRAFÍA

ALBET, S.A. Descripción del Producto SIGEP V 2.0. s.l. : ALBET.

ALEXANDER, CHRISTOPHER, et al. Design Patterns. Oxford University Press (Ed.), 1977.

ARECIO PR, GEISER. *Plan de Prueba de Aceptación de SIGEP.* CH : ALBET,UCI., 2007

ARIAS ORIZONDO, ARTURO CÉSAR. Visión del sistema de gestión penitenciaria. actualización. CH : ALBET.

BAUER, CHRISTIAN y KING, GAVIN. Hibernate in action. Greenwich, Manning Publications Co (Ed.),2005.

CALIMANO MENESES, YADIRA y MORA CASTILLO, MARISLEIDY. Modelación del Negocio y Levantamiento de requisitos de los procesos: requisas, novedades y contingencias y control de armamentos. CH : UCI, 2008.

GALLARDO, DAVID et al. Eclipse in Action. s.l. : Manning Publications, 2003.

KAISLER. *Oracle Database Documentation Library.* s.l. : ORACLE CORPORATION, 2005.

LARMAN, CRAIG. UML y Patrones, Introducción al Análisis y Diseño Orientado a Objetos. México : Prentice Hall Hispanoamericana, SA., 1999.

MCLAUGHLIN, BRETT. Building Java Enterprise Applications Volume I: Architecture. s.l. : O'Reilly 2002. 0-569-00123-1.

MEYERS, NATHAN. Java Programming in Linux. 2000. Wait Group.

OSSORIO, MANUEL. Diccionario de Ciencias Jurídicas, Políticas y Sociales. Guatemala : Datascan, S.A.

PIMENTEL GONZÁLEZ, LUIS ALBERTO. Documento de Arquitectura de Software. CH : ALBET, 2007.

PIMENTEL GONZALEZ, LUIS ALBERTO y PÉREZ RIVERO, IÓSEV. ArBaWeB: Arquitectura Base sobre la Web. CH : UCI, 2007.

RALPH E., JOHNSON. *Journal of Object-Oriented Programming.* s.l. : Components, Frameworks, Patterns., 1988.

SNYDER, BRUCE, DE VELDE, THOMAS y DUPUIS, CHRISTIAN. Beginning Spring Framework 2. Indianapolis, Willey Publishing (Ed.), 2007. p. 507

WALLS CRAIG y BREINDENBACH RAYN. String in Action. Greenwich : Manning Publications Co.

WALLS, CRAIG y BREINDENBACH, RYAN. Spring in Action. Second Edition. Greenwich, Manning Publications Co (Ed.), 2008. p. 765.

WILLEY. Professional Java Development with the Spring Framework. 2005.

WILLIAMSON, ALAN, et al. Programación Java Server J2EE Edition. 1.3. s.l. : ANAYA.

GLOSARIO

A

Arquitectura: Conjunto de patrones y abstracciones coherentes que guían la construcción del software informático.

F

Frameworks: Conjunto de APIs y herramientas destinadas a la construcción de un determinado tipo de aplicaciones de manera generalista.

H

HQL: Hibernate Query Language(Lenguaje de Consulta de Hibernate).

I

IDE: Integrated Development Enviroment (Ambiente Integrado de Desarrollo).

M

Mapear: Técnica de programación para hacer coincidir o convertir datos entre distintos sistemas

(Ejemplo Object-Relational Mapping).

Método: Operación que pueden modificar el estado de un objeto u obtener datos sobre el mismo.

O

Objeto: Instancia de una clase encargada de realizar acciones en la ejecución de un programa.

P

Persistir: Almacenar en la base de datos la información referente a algunos objetos.

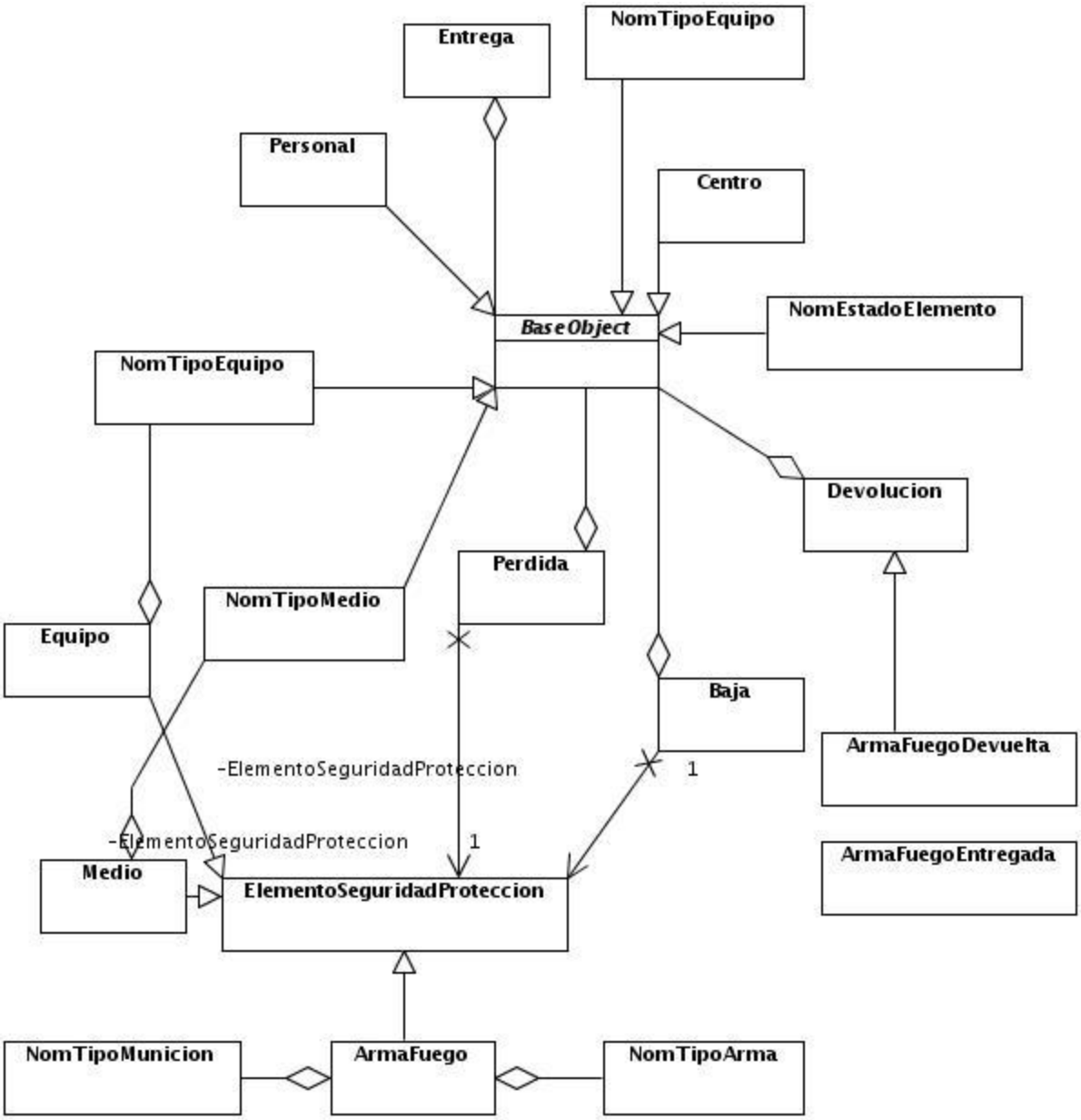
Plataforma: Tecnología básica del software de un ordenador que define como funciona y que otro tipo de software se puede emplear con él.

S

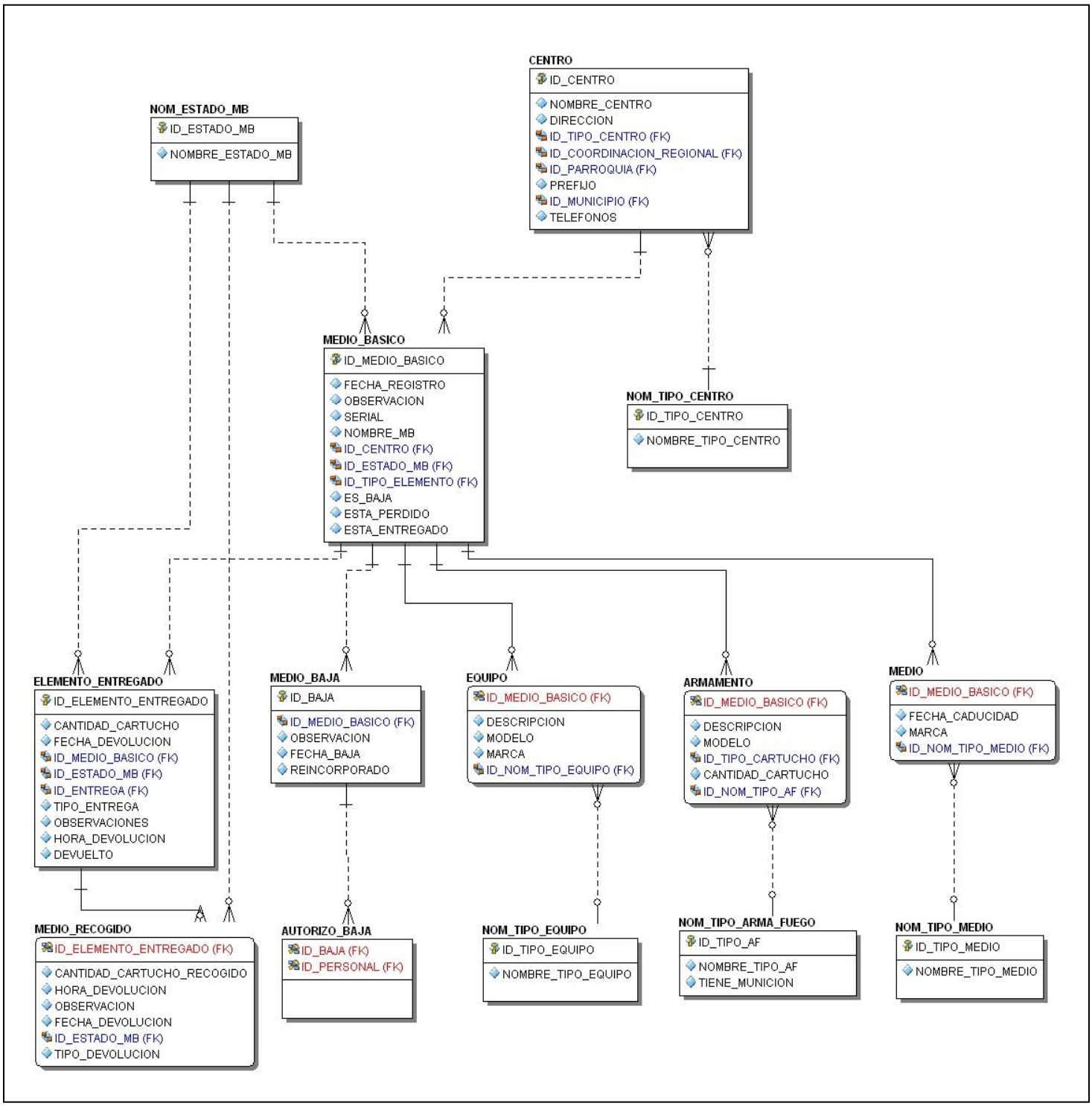
SIGEP: Sistema de Gestión Penitenciaria.

ANEXOS

2.1 Modelo de Dominio

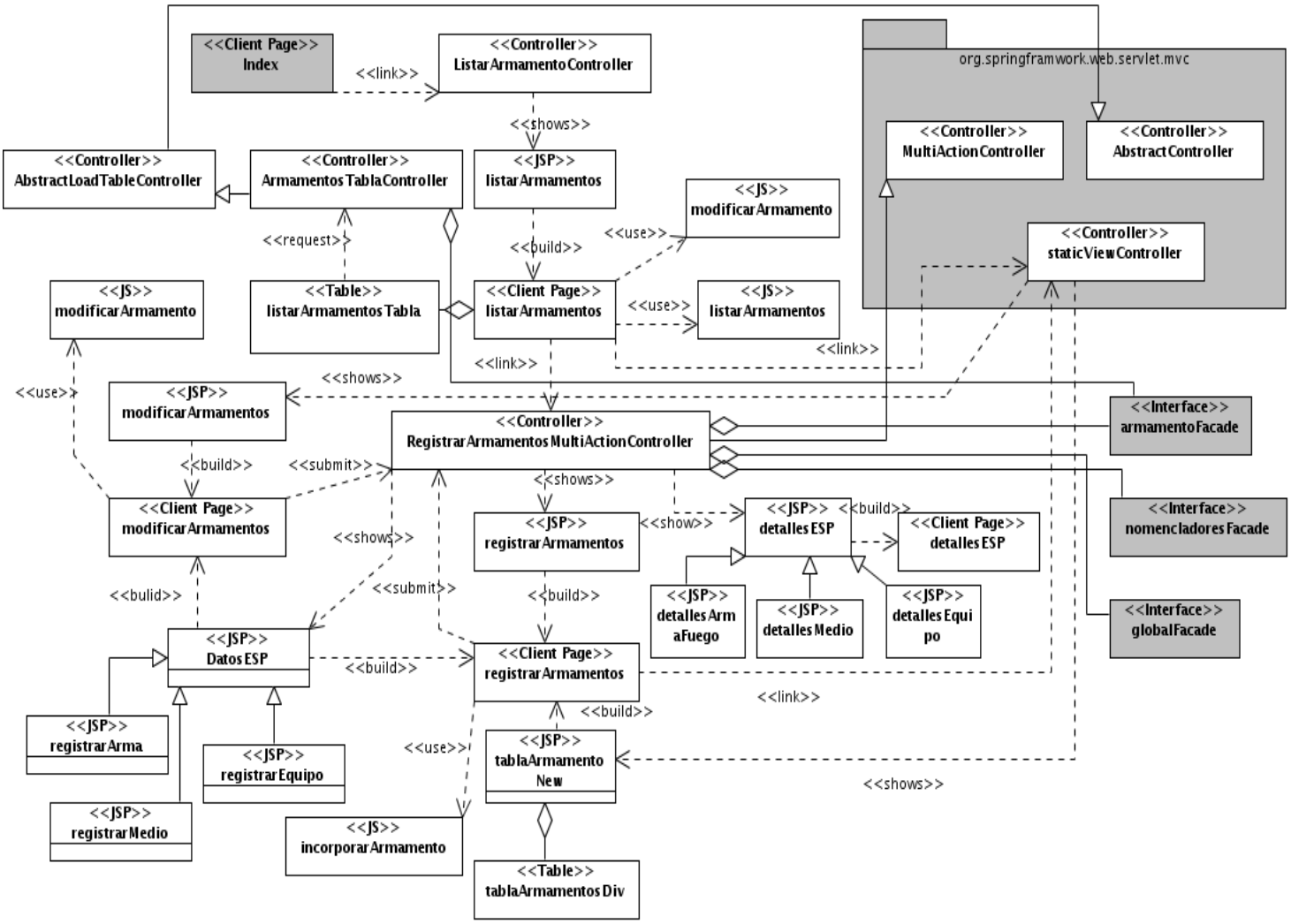


2.2 Diagrama Entidad Relación de la base de datos.

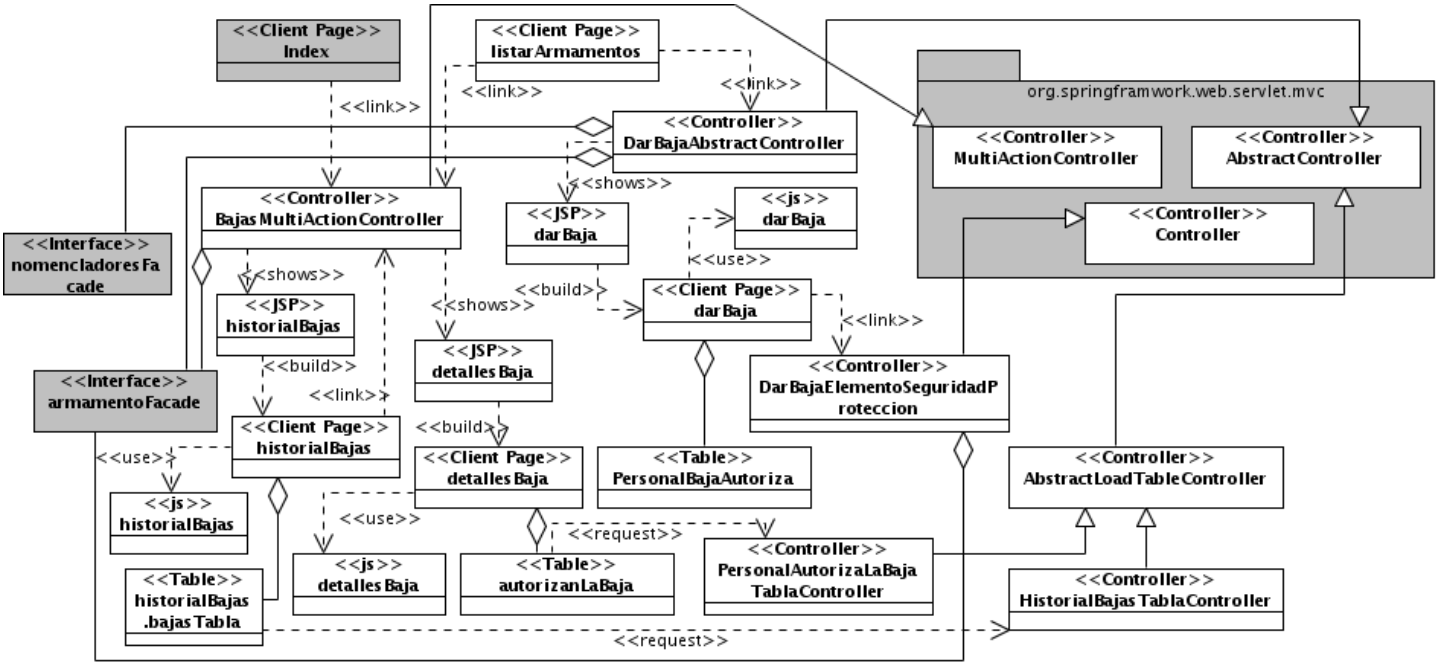


2.3 Diseño de la Capa de Presentación del módulo Control de armamento, equipos y medios de seguridad.

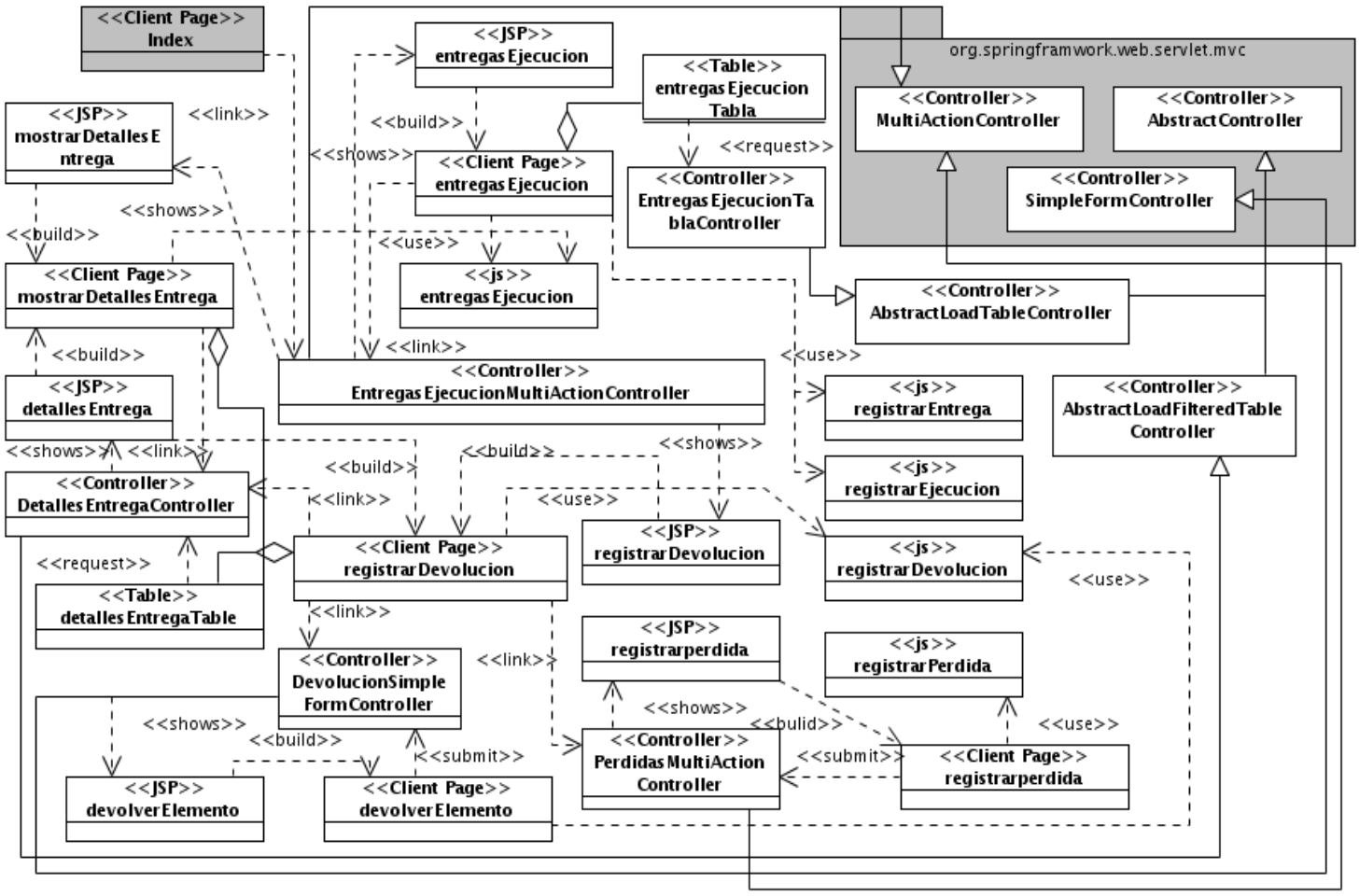
2.3.1 Gestión de armamentos, equipos y medios de seguridad.

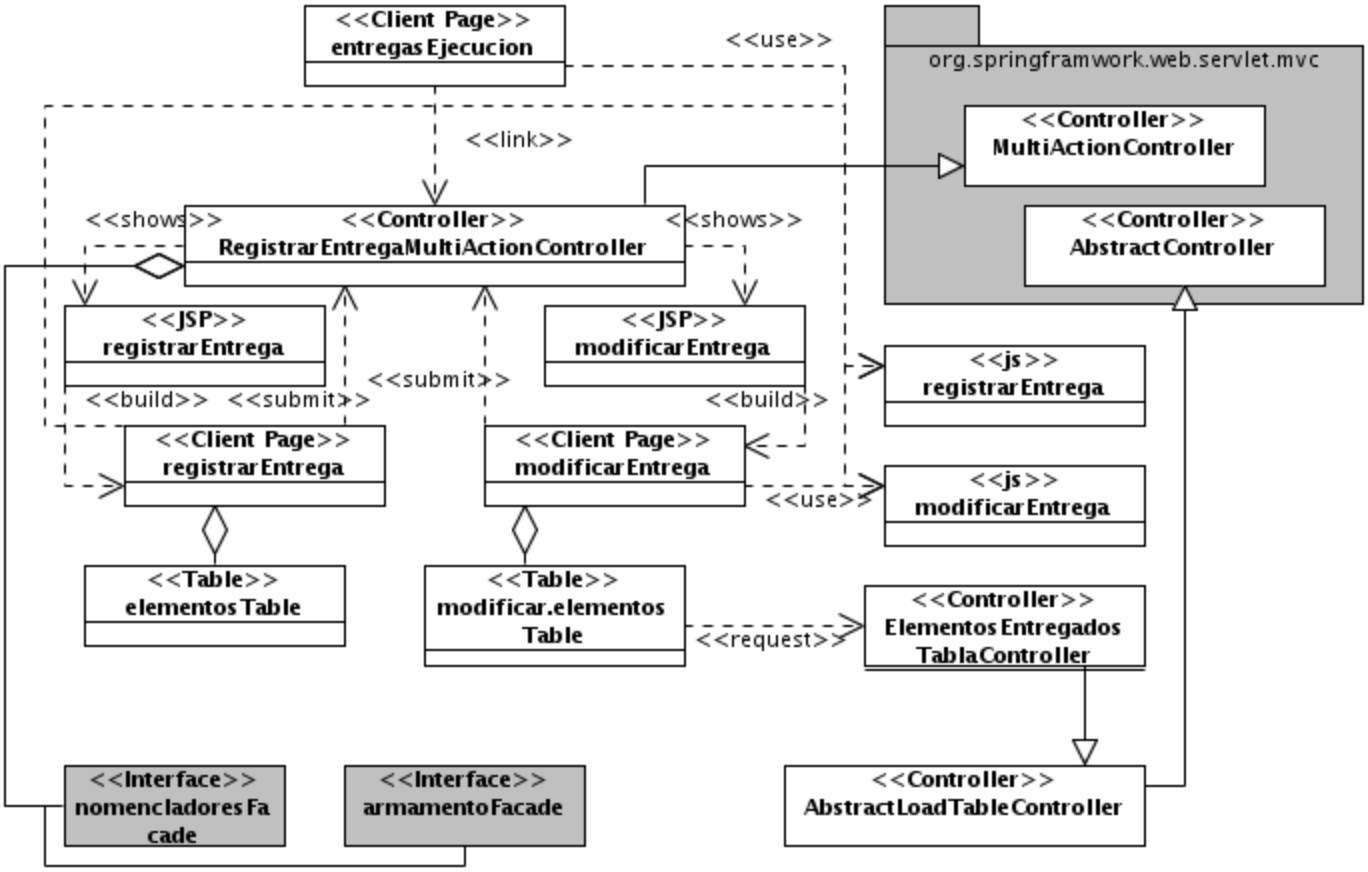


2.3.2 Gestión de bajas

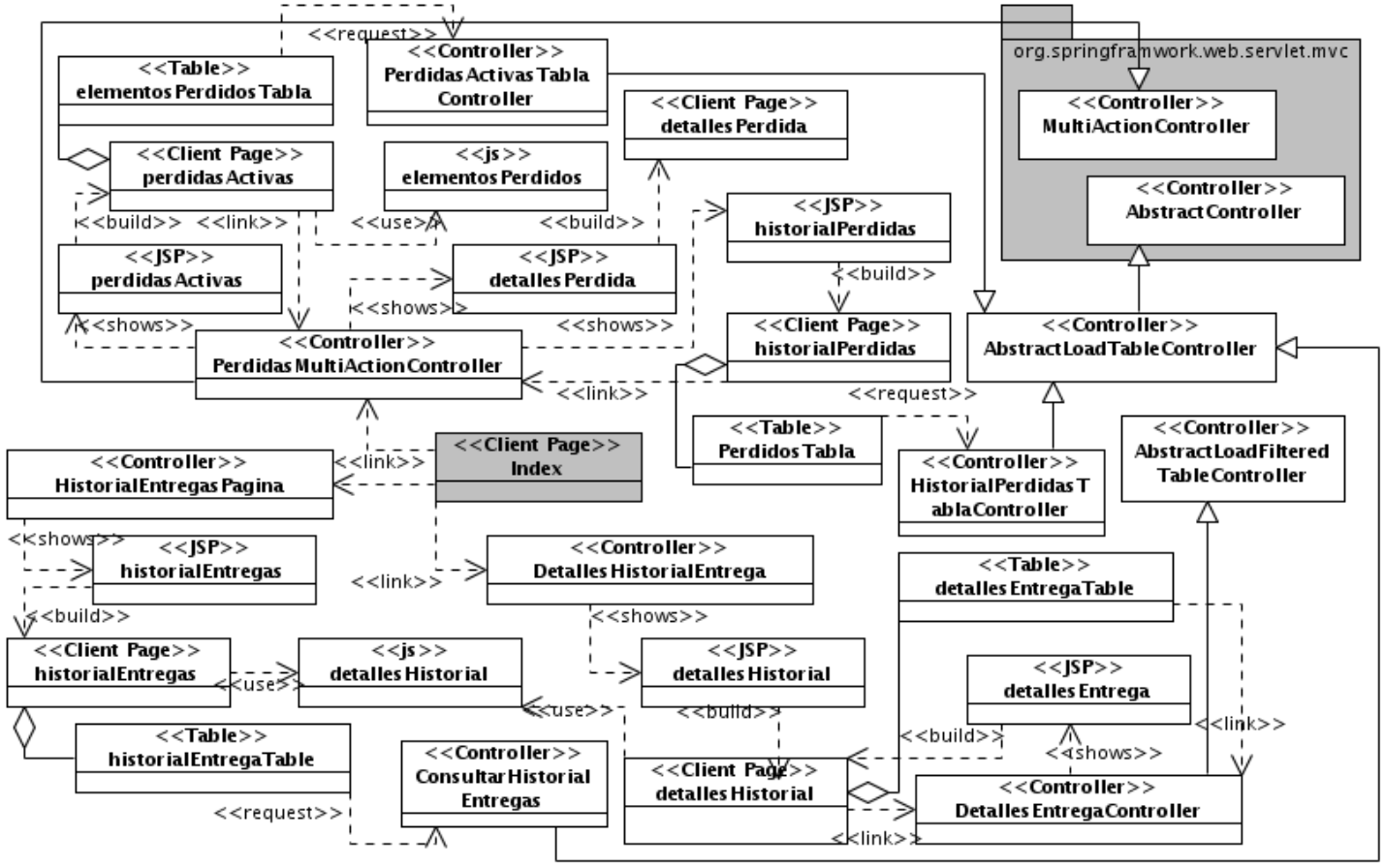


2.3.3 Gestión de entregas




<<Client Page>>
entregasEjecucion
<<use>>
org.springframwork.web.servlet.mvc
<<Controller>>
MultiActionController
<<Controller>>
AbstractController
<<link>>
<<shows>>
<<Controller>>
RegistrarEntregaMultiActionController
<<shows>>
<<JSP>>
registrarEntrega
<<JSP>>
modificarEntrega
<<submit>>
<<build>>
<<submit>>
<<build>>
<<js>>
registrarEntrega
<<Client Page>>
registrarEntrega
<<Client Page>>
modificarEntrega
<<use>>
<<js>>
modificarEntrega
<<Table>>
elementosTable
<<Table>>
modificar.elementos
Table
<<request>>
<<Controller>>
Elementos Entregados
TablaController
<<Interface>>
nomencladores Fa
cade
<<Interface>>
armamentoFacade
<<Controller>>
AbstractLoadTableController

2.3.4 Gestión de pérdidas



2.4 Ejemplo de consultas usando el API Criteria y el lenguaje HQL de Hibernate (fragmentos del EntregaDAOImpl).

```
public class EntregaDAOImpl extends AbstractBaseDAO<Entrega, String> implements
        EntregaDAO {

    protected EntregaDAOImpl() {
        super(Entrega.class);
    }
    private Map<String, String> getMap() {
        Map<String, String> map = new HashMap<String, String>();
        map.put("primerNombre", "personal.nombre1");
```

```
            map.put("primerApellido", "personal.apellido1");
            return map;
        }
        @SuppressWarnings("unchecked")
        public List<ElementoEntregado> obtenerElementosEntregados(
                final String idEntrega, final int inicio, final int cantidad,
                final List<String> idExcluidos) {
            return getHibernateTemplate().executeFind(new HibernateCallback() {
                public Object doInHibernate(Session session) {
                    StringBuilder q = new StringBuilder(
                            "select elementos from Entrega e join e.elementosEntregados as
elementos where e.id = :idEntrega");
                    if ((idExcluidos != null) && !idExcluidos.isEmpty())
                        q.append(" and elementos.id not in (:ids)");
                    Query query = session.createQuery(q.toString());
                    query.setFirstResult(inicio).setMaxResults(cantidad);
                    query.setParameter("idEntrega", idEntrega);
                    if ((idExcluidos != null) && !idExcluidos.isEmpty())
                        query.setParameterList("ids", idExcluidos);
                    return query.list();
                }
            });
        }
        @SuppressWarnings(value = { "unchecked" })
        public List<Entrega> obtenerEntregasEjecución(final int inicio,
                final int cantidad, final List<PropertyFilter> filtrar)
                throws Exception {
            return getHibernateTemplate().executeFind(new HibernateCallback() {
                public Object doInHibernate(final Session session)
                        throws HibernateException, SQLException {
                    String s = "id_entrega in (select ee.id_entrega from elemento_entregado ee
where ee.id_medio_básico in(select m.id_medio_básico from medio_básico m where m.esta_perdido = 0) and
ee.id_elemento_entregado not in (select m.id_elemento_entregado from medio_recogido m))";
                    Criteria criteriaEntrega = session
                            .createCriteria(Entrega.class);
```

```
                criteriaEntrega.add(Restrictions.sqlRestriction(s));
                criteriaEntrega.setFirstResult(inicio).setMaxResults(cantidad);
                criteriaEntrega.createAlias("personal", "personal",
                        CriteriaSpecification.INNER_JOIN);
                CriteriaUtils.aplicarFiltros(criteriaEntrega, null, filtrar,
                        getMap());
                return criteriaEntrega.list();
            }
        });
    }
```

2.5 Implementación del método persistirEntrega

```
public void persistirEntrega(HttpServletRequest request,
            HttpServletResponse response) throws Exception {
        List<ElementoEntregado> elementosAdd = null;
        /*
         * Obteniendo valores de la entrega
         */
        String idpersonal = request.getParameter("idFuncionario");
        String fecha = request.getParameter("fecha_entrega");
        String hora = request.getParameter("hora_entrega");
        SimpleDateFormat sdfFecha = new SimpleDateFormat("dd/MM/yyyy");
        SimpleDateFormat sdfHora = new SimpleDateFormat("HH:mm");
        elementosAdd = new ArrayList<ElementoEntregado>();
            sdfFechaDev = new SimpleDateFormat(
                    "dd/MM/yyyy");
            SimpleDateFormat sdfHoraDev = new SimpleDateFormat("HH:mm");
            Date fechaDev = sdfFechaDev.parse(jsonElementosAdd
                    .getJSONObject(i).getString("fechaDev"));
            Date horaDev = sdfHoraDev.parse(jsonElementosAdd.getJSONObject(
                    i).getString("horaDev"));
            ElementoSeguridadProteccion refElementoEntregado = armamentoFacade
                    .obtenerElementoSeguridadProteccion(jsonElementosAdd
```

```
                                        .getJSONObject(i)
                                        .getString("idElementoSegProt"));
                        /*
                         * si el elmeneto seguridad y proyección es un arma hay que
                         * ponerle la cantidad de municiones                          */

                        ElementoEntregado elementoEntregado = null;
                        if (refElementoEntregado instanceof ArmaFuego) {
                                NomTipoArmaFuego tipoAr = nomencladoresFacade
                                                .obtenerTipoArmaFuego(jsonElementosAdd
                                        .getJSONObject(i).getString("idtipoElemento"));
                                elementoEntregado = new ArmaFuegoEntregada();
                                elementoEntregado.setFechaDevolucion(fechaDev);
                                elementoEntregado.setHoraDevolucion(horaDev);
                                elementoEntregado.setObservaciones(jsonElementosAdd
                                                .getJSONObject(i).getString("observaciones"));
                                elementoEntregado.setNomEstado(refElementoEntregado
                                                .getNomEstadoElemento());
                                ((ArmaFuegoEntregada) elementoEntregado)
                                                .setElementoSeguridadProteccion(refElementoEntregado);
                        if (tipoAr.getLlevaMunicion()) {
                                        ((ArmaFuegoEntregada) elementoEntregado)

                .setCantidadMuniciones(Integer.parseInt(jsonElementosAd.getJSONObject(i)
.getString("cantMuniciones")));         }
                        } else {
                                elementoEntregado = new ElementoEntregado();
                                elementoEntregado.setFechaDevolucion(fechaDev);
                                elementoEntregado.setHoraDevolucion(horaDev);
                                elementoEntregado.setObservaciones(jsonElementosAdd
                                                .getJSONObject(i).getString("observaciones"));
                                elementoEntregado.setNomEstado(refElementoEntregado
                                        .getNomEstadoElemento());
                                elementoEntregado
```

```
                                        .setElementoSeguridadProteccion(refElementoEntregado);
                    }
                    elementosAdd.add(elementoEntregado);
                }
                Entrega entrega = new Entrega();
                entrega.setHoraEntrega(horaEntrega);
                entrega.setFechaEntrega(fechaEntrega);
                entrega.setElementosEntregados(elementosAdd);
                Personal personal = armamentoFacade.obtenerPersonal(idpersonal);
```

2.6 Implementación de la clase RegistrarDevolucionSimpleFormController

```
public class DevoluciónSimpleFormController extends SimpleFormController {

    private ArmamentoFacade armamentoFacade;
    private NomencladoresFacade nomencladoresFacade;
    @Override
    protected ModelAndView onSubmit(HttpServletRequest request,
            HttpServletResponse response, Object command, BindException errors)
            throws Exception {
        DevoluciónCommand devoluciónCommand = (DevoluciónCommand) command;
        // creando estado
        NomEstadoElemento nomEstadoElemento = nomencladoresFacade
                .obtenerEstadoElemento(devoluciónCommand.getIdEstado());

        // creando elemento entregado
        ElementoEntregado elementoEntregado = armamentoFacade
                .obtenerElementoEntregado(devoluciónCommand
                        .getIdElementoEntregado());

        devoluciónCommand.setElementoEntregado(elementoEntregado);

        devoluciónCommand.setEstado(nomEstadoElemento);
        // persisitiendo devolución
```

```
        armamentoFacade.registrarDevolución(devoluciónCommand.getDevolución());
        return null;
}

@Override
protected Map referenceData(HttpServletRequest request) throws Exception {

        Map map = new HashMap();

        List<NomEstadoElemento> estados = nomencladoresFacade
                        .obtenerEstadoElemento();
        map.put("estados", estados);
        Date date = new Date();
        map.put("timeActual", date.getTime());
        return map;

}
@Override
protected Object formBackingObject(HttpServletRequest request)
                throws Exception {
        DevoluciónCommand devoluciónCommand = new DevoluciónCommand();
        String idElementoEntregado = request
                        .getParameter("idElementoEntregado");
        if (StringUtils.hasText(idElementoEntregado)
                        && !isFormSubmission(request)) {
                ElementoEntregado elementoEntregado = armamentoFacade
                                .obtenerElementoEntregado(idElementoEntregado);
                if ((elementoEntregado.getElementoSeguridadProteccion() instanceof ArmaFuego)
                                && ((ArmaFuego) elementoEntregado
                                                .getElementoSeguridadProteccion())
                                                .getNomTipoArmaFuego().getLlevaMunicion()) {
                        devoluciónCommand.setCantidadMuniciones(0);
                } else {
                        devoluciónCommand.setCantidadMuniciones(-1);
```

```
                }
        } else {
                int municiones = Integer.parseInt(request
                                .getParameter("cantidadMuniciones"));
                if (municiones >= 0) {
                        devoluciónCommand.setDevolución(new ArmaFuegoDevuelta());
                }
        }
        return devoluciónCommand;
}

@Override
protected void initBinder(HttpServletRequest request,
                ServletRequestDataBinder binder) throws Exception {
        CustomDateEditor editorFecha = new CustomDateEditor(
                        new SimpleDateFormat("dd/MM/yyyy"), true);
        CustomDateEditor editorHora = new CustomDateEditor(
                        new SimpleDateFormat("HH:mm"), true);

        binder.registerCustomEditor(Date.class, "fechaDevolucion", editorFecha);
        binder.registerCustomEditor(Date.class, "horaDevolucion", editorHora);
}
public ArmamentoFacade getArmamentoFacade() {
        return armamentoFacade;
}
public void setArmamentoFacade(ArmamentoFacade armamentoFacade) {
        this.armamentoFacade = armamentoFacade;
}
public NomencladoresFacade getNomencladoresFacade() {
        return nomencladoresFacade;
}
public void setNomencladoresFacade(NomencladoresFacade nomencladoresFacade) {
        this.nomencladoresFacade = nomencladoresFacade;
}
```