Universidad de las Ciencias Informáticas

Tema: Diseño e implementación de los módulos Pertenencias y Medidas Disciplinarias del SIGEP Venezuela.

Trabajo de Diploma

Autor(es): Ludmila Yirenis Manso Rodríguez.
Maydalis Pizarro Barata.

Tutor: Ing. Haydee Fernández Díaz.

La Habana, mayo 2010

DECLARACIÓN DE AUTORÍA

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

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

_____ _____

Maydalis Pizarro Barata **Ludmila Yirenis Manso Rodríguez**

_____

Ing. Haydee Fernández Díaz

DATOS DE CONTACTO

Graduada en el año 2007 como Ingeniera en Ciencias Informáticas en la UCI.

Ha participado en eventos como UCIENCIA a nivel de base.

Opta actualmente por la categoría docente de Instructor y cursa la maestría de Gestión de Proyecto.

Se ha vinculado a los proyectos productivos en el rol de Subgerente de Proyecto y Planificadora.

Ha recibido los cursos del Diplomado Docencia e Innovación Universitaria, así como el curso de Gestión de Proyectos de ALBET.

DEDICATORIA

Maydalis Pizarro Barata

A mi mamá: Por haberme apoyado siempre, por estar orgullosa de mi, por brindarme su amor, confianza y apoyo , por enseñarme a ser la persona que soy en estos momentos, gracias por todo, te quiero mucho.

A mi papá: Por enseñarme a ver la vida de otra manera.

A Ricardo: Por confiar en mi y ser junto a mi madre los mejores guías que he tenido, gracias a ti estoy hoy aquí.

A mis abuelos: Por todo su amor incondicional, por creer en mí y por darme tantos momentos de felicidad.

A mi novio Luis Edgardo: A ti te doy las gracias por todo, por ser mi novio, mi compañero, mi amigo, mi apoyo en todo momento, por estar siempre presente, por enseñarme a ser mejor persona y una buena profesional, te amo.

A mis tias y primas: Por todo su apoyo y confianza.

A mi suegra Janet: Por su preocupación y apoyo en todo momento.

A mis viejas amigas: Miladys, Isol, Yailene, Yordenky, por creer en mí y por su sincera amistad.

A todos mis amigos de la universidad, a mis compañeros de grupo, sobre todo los del SIGEP, y a todos aquellos que de una forma u otra han estado conmigo los 5 años de carrera.

Ludmila Yirenis Manso Rodríguez

A mis padres: Por inculcarme sus ejemplos como seres humanos y como profesionales. En especial a mi mamá que ha sido mi principal fuente de inspiración todos estos años, te quiero mucho mami.

A mi abuela: Por hacerme saber siempre que estará orgullosa de mi, por estar ahí cuando me hace falta, por quererme tanto, te quiero mucho.

A mis suegros: Que me han apoyado en todo momento como si fueran mis padres.

A mi novio Osmany: Por todo su amor, por nunca dejarme caer, por ser mi brazo derecho, izquierdo mi todo, te amo tanto.

A Fedalia: La mamá de mi hermana que me ha ayudado en todo momento y se ha portado como si fuera mi madre.

A todos mis amigos, mis compañeros de aula y a todos lo que han querido que yo esté aquí ahora.

RESUMEN

Desde hace varios años la situación penitenciaria en la República Bolivariana de Venezuela se ha agudizado: el hacinamiento, la falta de clasificación de los reclusos y el retardo procesal son varios de los ejemplos que lo evidencian. Teniendo en cuenta estos problemas y la necesidad de contar con una herramienta que permita mantener actualizada la información referente al interno y a los centros penitenciarios, nace el proyecto de Humanización del Sistema Penitenciario y dentro de este, el Sistema de Gestión Penitenciaria (SIGEP). Entre los principales procesos a informatizar por el SIGEP se encuentran Pertenencias y Medidas Disciplinarias. Para la gestión de estos procesos se definieron ambos módulos dentro del subsistema: Seguridad y Custodia.

Este trabajo se basa en las actividades realizadas por los autores en los roles de diseñador e implementador como integrantes del proyecto SIGEP. En él se muestra el diseño y la implementación de la solución brindada por ellos para los módulos Pertenencias y Medidas Disciplinarias a partir de los requisitos definidos con el cliente y el análisis de la arquitectura especificada para el SIGEP.

Para darle cumplimiento a las funcionalidades esperadas por ambos módulos se realizó un diseño basado en patrones. Para la implementación de este diseño se utilizaron herramientas y tecnologías de la plataforma Java que exponen tendencias de la programación web y que en su mayoría son libres y de código abierto.

Como resultado de todas las actividades realizadas se obtuvo el diseño e implementación de ambos módulos, se integraron como componentes ejecutables al SIGEP y se validaron en un ambiente de trabajo real en un centro penitenciario en la República Bolivariana de Venezuela.

PALABRAS CLAVE

Pertenencias, Medidas Disciplinarias, SIGEP, Arquitectura, Diseño, Implementación.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Desde hace más de cuatro décadas el sistema penitenciario en la República Bolivariana de Venezuela confronta innumerables problemas, la violencia carcelaria nacional tiene sus orígenes en dos grupos de factores causales. El primero de ellos, atribuidos a la propia institución: infraestructura precaria, hacinamiento, falta de clasificación de los reclusos, el trato dado a los visitantes, el retardo procesal, traslado a los tribunales, corrupción, excesos y abusos de custodios y guardias, impunidad, sostenimiento del ocio, entre otros. El segundo, originado por los propios actores: tráfico y consumo de drogas, conflictos por control territorial y el tráfico de armas.

A principios del año 2005 se realizó en la República Bolivariana de Venezuela un censo nacional de la situación judicial de la población penitenciaria. Para este se elaboró una encuesta que recogía información sobre: datos del interno, datos sobre el delito, sobre el proceso judicial y la condición física de la infraestructura penitenciaria. Todo esto permitió contar con un acercamiento a la realidad del momento en que fue aplicado, sin embargo, no resolvió las necesidades en materia de gestión, información, comunicación y apoyo a la toma de decisiones en la Dirección Nacional de Servicios Penitenciarios (DNSP), además de carecer de mecanismos de actualización.

Para resolver estos problemas, teniendo en cuenta la difícil situación penitenciaria por la cual atraviesa República Bolivariana de Venezuela y la necesidad de contar con una herramienta que permita mantener actualizada la información relacionada a los internos y los centros penitenciarios, se decide construir un sistema informático que contribuya a soportar la toma de decisiones estratégicas por parte de la DNSP. A raíz de esto nace el proyecto de Humanización del Sistema Penitenciario que incluye un sistema informático para gestionar y automatizar los procesos penitenciarios conocido como Sistema de Gestión Penitenciaria (SIGEP).

Una parte importante del SIGEP sería gestionar las pertenencias, indisciplinas y medidas disciplinarias, ya que la información que se maneja en los centros de reclusión se lleva de manera manual. No existe un modelo estándar para llevar el control de los objetos entregados en depósito por los privados de libertad a su ingreso en los establecimientos penitenciarios, lo que conlleva

que a la hora de entregar las pertenencias a los reclusos que van a salir en libertad o a los familiares del individuo una vez fallecido dentro del establecimiento, no haya concordancia entre las pertenencias y el documento físico o entre el recluso y sus datos en la ficha de las pertenencias.

La implantación de las medidas disciplinarias a un recluso es un proceso que no tiene la misma eficacia en todas las instituciones, ya que aunque son las mismas medidas disciplinarias, en todos los centros no se registran de igual forma. Además de esto no se tiene un control sobre todas las indisciplinas cometidas por el recluso y no existe un reglamento que relacione las faltas cometidas por el individuo con las medidas disciplinarias.

A partir de lo descrito anteriormente se plantea como **problema**: ¿Cómo gestionar de manera organizada las pertenencias y las medidas disciplinarias de los reclusos en las sedes de la DNSP?

Como propuesta de solución al problema en cuestión se propone como **objetivo** desarrollar los módulos de Pertenencias y Medidas Disciplinarias de los reclusos en las sedes de la DNSP, teniendo en cuenta los requerimientos, arquitectura y tecnologías definidas en el SIGEP. Por lo que, los resultados esperados son los modelos de diseño e implementación validado de los módulos Pertenencias y Medidas Disciplinarias y los módulos como componentes ejecutables integrados al SIGEP.

En consecuencia, el **objeto de estudio** del presente trabajo son Procesos de Gestión de las pertenencias y medidas disciplinarias en los sistemas penitenciarios y el **campo de acción** Gestión de pertenencias y medidas disciplinarias en las sedes de la DNSP de la República Bolivariana de Venezuela.

En función de los objetivos y de las actividades a realizar se definen las siguientes **tareas de investigación**:

- Análisis de la documentación necesaria para la realización del estado del arte sobre la gestión de las pertenencias y medidas disciplinarias de los reclusos en las sedes penitenciarias.

- Selección de la tecnología necesaria y factible para la implementación del sistema.
- Elaboración del modelo de diseño.
- Implementación que facilite la gestión de las pertenencias y medidas disciplinarias de los reclusos.
- Realización de las pruebas unitarias.

El documento consta de 3 capítulos más anexos.

En el **CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA** se realiza un estudio de algunas soluciones de software existentes relacionadas con los Sistemas Penitenciarios tanto a nivel nacional como internacional. De igual forma se hace referencia a la metodología de desarrollo de software, las principales herramientas que se utilizaron, así como la plataforma en la cual se trabajó.

En el **CAPÍTULO 2: ARQUITECTURA DEL SISTEMA** se explica la arquitectura utilizada para desarrollar los módulos Pertenencias y Medidas Disciplinarias. Se describe el modelo de la arquitectura, las capas por las que está compuesta y se realiza un análisis de las actividades y pautas generales para el diseño e implementación de los módulos tratados dentro del SIGEP.

En el **CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA** se presentan fragmentos de diagramas y código fuente explicativos del diseño e implementación de los módulos Pertenencias y Medidas Disciplinarias.

En los **ANEXOS** se muestra la documentación generada como resultado de los procesos de diseño e implementación de los módulos Pertenencias y Medidas Disciplinarias.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

Como parte de la problemática en este capítulo se abordan algunos conceptos del Sistema Penitenciario Venezolano, los cuales serán referenciados en el desarrollo del trabajo. Se realiza el análisis de antecedentes de software para sistemas penitenciarios en la región de Latinoamérica y las tecnologías, herramientas y tendencias de desarrollo a utilizar en el diseño e implementación de las funcionalidades de los módulos Pertenencias y Medidas Disciplinarias del Sistema Penitenciario Venezolano, en el marco del desarrollo del Sistema de Gestión Penitenciaria (SIGEP).

1.2. Sistemas Penitenciarios

Sistema penitenciario se define 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. [1]

Un Sistema penitenciario es también "un conjunto de normas legislativas o administrativas encaminadas a determinar los diferentes sistemas adoptados para que los penados cumplan sus penas. Se encamina a obtener la mayor eficacia en la custodia o en la readaptación social de los delincuentes. [2]

1.3. Sistemas penitenciarios y sistemas informáticos

El precario funcionamiento de las instituciones y las dificultades en la estandarización de los procesos penitenciarios son características comunes de los sistemas penitenciarios de América Latina. Aunque, en algunos de ellos existen sistemas informáticos que sirven de apoyo a la gestión de los procesos en los centros penitenciarios y en la toma de decisiones a nivel centralizado.

Para el desarrollo del SIGEP se hizo un análisis de algunos sistemas automatizados existentes que fueron valorados como posible solución o parte de la solución para realizar este trabajo, se

analizaron teniendo en cuenta las funcionalidades que brindan con el fin de identificar los principales beneficios y limitantes que un sistema como este pueden producir. A continuación se abordan características de algunos de ellos.

1.3.1. Sistema penitenciario del gobierno de Cuba

El sistema penitenciario nacional comienza el desarrollo de la informatización a partir del 1989 con la automatización de los datos principales del recluso y algunos aspectos de control penal. A raíz del cumplimiento de la orden 43/99 del Viceministro Primero del Ministerio del Interior, se crea un Sistema Automatizado para el Control del Recluso (SACORE). Este sistema fue diseñado y programado sobre la tecnología existente en el MININT y fue implantado a partir de enero 2003 por lo que tiene 6 años en explotación, contiene como módulos principales Control Penal, Reeducación Penal y el Orden Interior. El SACORE se caracteriza por: garantizar respuestas inmediatas a las solicitudes de información de los diferentes órganos e instituciones del estado (Jefatura del MININT, Ministerio de Justicia, Tribunales, Fiscalías, MINED, INDER, FMC, MINFAR), recoger prácticamente la totalidad de la información de los reclusos en todas las especialidades, tener más de 200 reportes impresos, permitir la recuperación dinámica a partir de una solicitud de búsqueda, emitir partes de forma automatizada y permitir el traslado automático de todos los datos del recluso al nivel nacional.

Además del SACORE se desarrollaron el Sistema Automatizado de Incidencias de la Dirección de Establecimiento Penitenciaria (SAIDEP) y el Sistema Administración de Capacidades de la Dirección de Establecimiento Penitenciaria (SACDEP).

1.3.2. Sistema penitenciario del gobierno de Ecuador

El Sistema Nacional de Gestión Penitenciaria "SIGPEN", permite recopilar y controlar la información operativa que se genera en el Centro de Rehabilitación Social. SIGPEN funciona en una plataforma informática operativa en Internet, en la que se tiene la información inherente a los internos y al área administrativa de los 36 reclusorios del país. El SIGPEN es un sistema que gestiona la información referente al interno, así como los datos personales, la situación jurídica, el expediente médico y laboral, los movimientos (salidas, traslados, visitas y recepción de celda), la calificación del comportamiento y la evaluación psicológica, social, de peligrosidad y de trabajo.

Además genera reportes sobre la libertad controlada, prelibertad, conducta, fuga e información del censo. El sistema no solo es de uso administrativo sino también orientador, ya que con él se pueden fijar políticas específicas para algunos temas.

1.3.3. Sistema penitenciario del gobierno de Panamá

Como resultado de un proyecto financiado por las Naciones Unidas y el Gobierno Español en coordinación con la Dirección General de Sistemas Penitenciario de Panamá (DGSP), surge a principios del año 1997 un sistema para la gestión penitenciaria, con la finalidad de mantener almacenado en una base de datos los registros de los internos que están detenidos en los centros penales a nivel nacional. Es un sistema centralizado, con una sede que se enlaza con otras unidades operativas distribuidas en una parte de los centros penitenciarios, con el resto se mantiene un enlace mediante el sistema de actualización por disco de tres y media.

El proceso se inicia con la captura de la información de los datos personales del interno, antecedentes médicos, datos socio-económicos y jurídicos y algún otro dato de importancia, por parte del personal ubicado en el centro penal. La información se registra automáticamente en una base de datos Oracle, que radica en la sede central y la cual está disponible para los diferentes departamentos que tienen acceso al sistema. Esto garantiza que se refleje de forma inmediata los cambios en el expediente del interno tales como traspasos de autoridad, traslado de un centro a otro, libertades, diligencias médicas, jurídicas y la realización del cómputo automático de la sentencia.

Las limitaciones se evidencian en los centros que no poseen enlace aún con la dirección central lo que debe llevar a una transformación de la red externa, reestructuración de la red interna de la sede, la dotación de Internet a la institución, la actualización de toda la información de los centros penitenciarios y actualización de los equipos.

1.3.4. Ventajas y Limitaciones

Los sistemas informáticos del gobierno de Cuba, Ecuador y Panamá tienen como principal limitante la definición de los procesos penitenciarios que los fundamentan y la infraestructura física sobre la cual se implantan. En la mayoría de los casos la falta de conectividad entre los centros

penitenciarios imposibilita la instalación de sistemas centralizados que garanticen la integridad e inmediatez de la información. La mayoría de estos sistemas implementan términos de la legislación vigente en los países para los cuales fueron definidos lo que los convierte en sistemas no portables a otro sistema penitenciario. A pesar de sus muchas limitaciones estos sistemas sirven de apoyo a los procesos de sus sistemas penitenciarios y funcionan como herramientas de trabajo a nivel local e institucional.

1.3.5. Sistema penitenciario venezolano

Intramuros y Extramuros son los dos posibles regímenes para el cumplimiento de condenas según las leyes venezolanas. Para cada individuo el sistema judicial decide el régimen adecuado tomando en cuenta historial judicial, salud y precedentes delictivos. Los cambios de regímenes se expresan a través del otorgamiento o revocatoria de Fórmulas Alternativas de Cumplimiento de Pena (FACP).

Extramuros es un régimen abierto donde los internos que se encuentran bajo cumplimiento de medidas restrictivas de libertad, pueden interactuar con la comunidad. Los centros de Extramuros son las Unidades Técnicas de Supervisión y Orientación (UTSO) y los Centros de Residencia Supervisada (CRS), donde el individuo no se encuentra a tiempo completo porque generalmente está insertado al trabajo y a la sociedad directamente.

Régimen Intramuros o cerrado se refiere a los centros que albergan a los individuos que se encuentran bajo cumplimiento de medidas privativas de libertad. Los centros penitenciarios de intramuros son los Establecimientos Penitenciarios en los cuales el individuo se encuentra a tiempo completo hasta el cumplimiento de la condena o un cambio de régimen.

El sistema penitenciario venezolano actual comparte las características básicas de los sistemas penitenciarios latinoamericanos. La información de los individuos privados de libertad se colecta de diferentes formas, mayormente en papel, salvo algunas iniciativas aisladas que utilizan hojas de cálculo, con el fin de facilitar el trabajo de los funcionarios de cada penal, sin que medien

formatos estándares para ello. No existe registro de sistema informático precedente que sirva como apoyo a los procesos penitenciarios, ni a la toma de decisiones a nivel centralizado.

Con el desarrollo del SIGEP se espera como resultado un sistema informático centralizado que estandarice e implemente los procesos fundamentales en el sistema penitenciario venezolano y se convierta en herramienta de trabajo para funcionarios en los centros penitenciarios y para la toma de decisiones a nivel institucional, solucionando así las problemáticas planteadas anteriormente.

Teniendo en cuenta la estructura actual del sistema penitenciario venezolano se identificaron las áreas que el SIGEP informatizaría y se conformaron los sistemas y subsistemas que este contendría. En la Figura 1.1 se muestran los sistemas y subsistemas del SIGEP y las relaciones de dependencia entre ellos. [1]

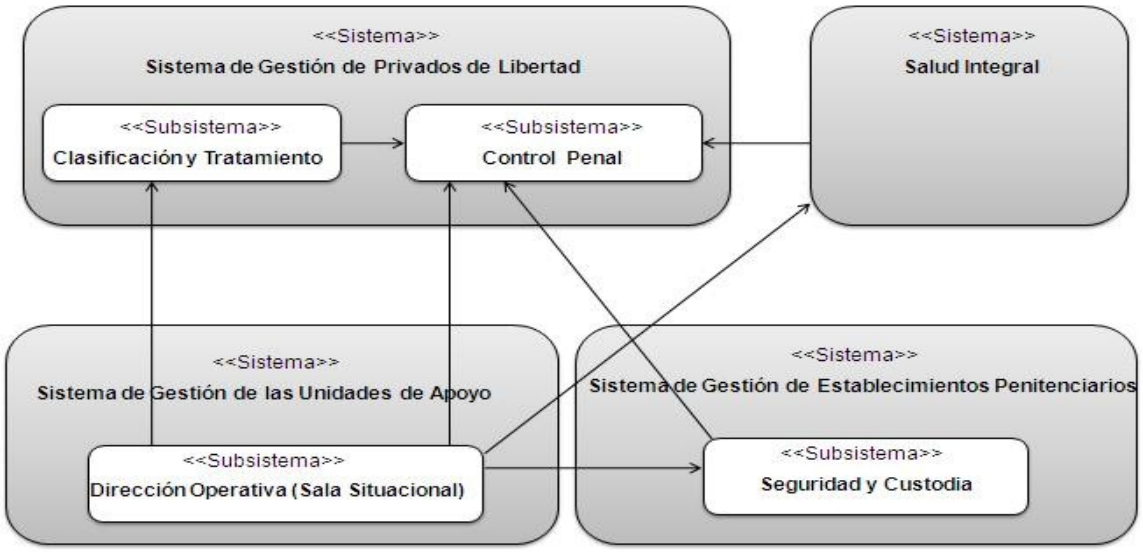


Figura 1.1 Sistemas y subsistemas que integran el SIGEP y sus relaciones de dependencia

1.4. Seguridad y Custodia

Este subsistema permitirá controlar las actividades de custodia a los reclusos y de orden interior de los establecimientos penitenciarios incluyendo los módulos de requisas y decomisos, faltas y

medidas disciplinarias, registro de pertenencias, novedades y contingencias, salidas transitorias, traslados interpenal o trasferencias, planificación y ejecución de visitas familiares e institucionales, ubicación, gestión de cupos, control de armamentos.

El Registro de Pertenencias permite llevar el control de los objetos entregados en depósito por los privados de libertad a su ingreso en los establecimientos penitenciarios, así como la devolución de estos a las personas autorizadas.

Las Faltas y Medidas Disciplinarias permite registrar las faltas disciplinarias cometidas por los privados de libertad. Las faltas disciplinarias se categorizan en función de su gravedad y son reflejadas en el expediente penitenciario. Las medidas disciplinarias se imponen a partir de las faltas disciplinarias registradas que no hayan sido sancionadas; de igual manera son reflejadas en el expediente del individuo.

1.4.1. Proceso de Pertenencias

Cuando el individuo entra al establecimiento penitenciario se le recogen las pertenencias que trae consigo, registrando la cantidad y fecha en la que se guardaron las mismas en el depósito del centro. Cuando el recluso hace entrega de sus pertenencias declara a la persona que, en caso de fallecimiento o de no existir en el centro un depósito, se le haga entrega de las pertenencias.

1.4.2. Proceso de Faltas y Medidas Disciplinarias

Las faltas disciplinarias cometidas por un individuo dentro del centro penitenciario se clasifican en leve, grave, y muy grave de acuerdo a las categorías siguientes: agresión física con golpes hacia otro interno, agresión física con objetos contundentes hacia un funcionario, agresión verbal hacia otro interno, daños a las instalaciones físicas y bienes nacionales, decomiso de arma blanca, intento de fuga, entre otras. Teniendo en cuenta las faltas cometida por el recluso, se le aplica una medida disciplinaria la cual es registrada en el expediente penitenciario.

Teniendo en cuenta las faltas cometidas por el recluso, se le aplica una medida disciplinaria la cual es registrada en el expediente penitenciario.

1.5. Tecnologías y Herramientas utilizadas en el desarrollo

La herramientas, tecnologías y flujos de procesos utilizados en la realización del SIGEP fueron definidas por el grupo de arquitectura, por lo cual en este epígrafe se realiza una identificación de cada una de ellas y la utilidad que proporcionan en el trabajo del diseño e implementación de los módulos de Pertenencias y Medidas Disciplinarias.

1.5.1. Metodología de Desarrollo

Rational Unified Process (RUP)

Es la forma disciplinada de asignar tareas y responsabilidades en una organización de desarrollo, siguiendo como objetivos asegurar la producción de software de calidad que satisfagan las necesidades de los usuarios finales y clientes, dentro de plazos y presupuestos establecidos. RUP es un proceso que se caracteriza por ser dirigido por casos de uso, centrado en la arquitectura e iterativo incremental. Identifica una serie de roles para los trabajadores, que son los responsables de generar una serie de artefactos y realizar una serie de actividades; se pueden encontrar entre los principales roles el ingeniero de requisitos, diseñador, arquitecto, implementador, jefe de proyecto y probador. RUP divide el proceso de desarrollo en ciclos, teniendo un producto al final de cada ciclo. Cada ciclo se divide en cuatro Fases: inicio, elaboración, construcción y transición. Cada fase concluye con un hito bien definido donde deben tomarse ciertas decisiones.

Los flujos de trabajo representados en la Figura 1.2 plantean la realización de un gran número de actividades y la elaboración de un amplio conjunto de artefactos, que por lo general los proyectos de desarrollo de software se comprometen a desarrollar pero que no cumplen en su totalidad debido a la carencia de tiempo o a que descubren que no eran necesarios. RUP indica que al inicio del proyecto se realice una adecuación de cada flujo de trabajo de manera que se produzcan solo los artefactos y se realicen las actividades que tienen un propósito dentro del proyecto.

A continuación se describen los flujos de trabajo de mayor interés para la realización del trabajo de diploma, según los roles que desarrollaron los autores.

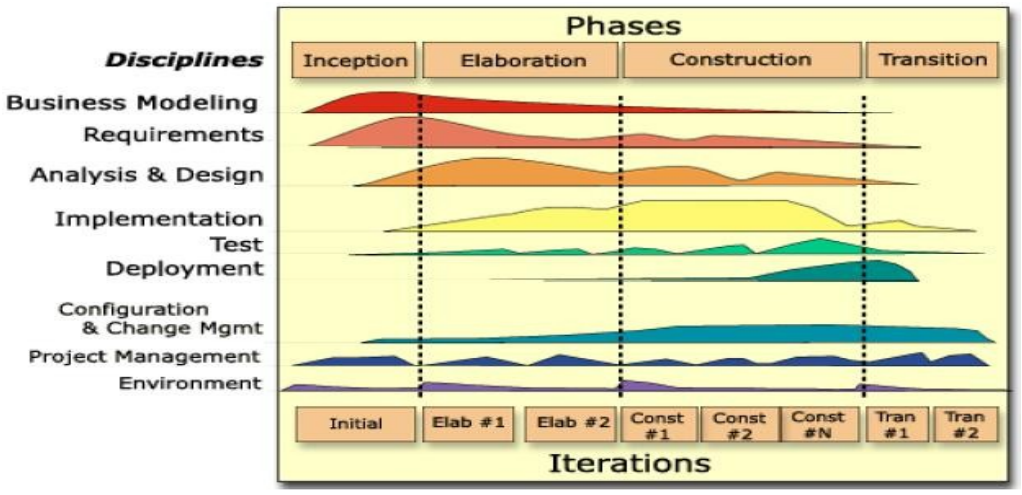


Figura 1.2 Fases y flujos de trabajo en RUP.

Análisis y Diseño

Los objetivos del análisis y diseño son: transformar los requisitos al diseño del futuro sistema, desarrollar una arquitectura para el sistema, adaptar el diseño para que sea consistente con el entorno de implementación. De manera general el propósito de este flujo de trabajo es traducir los requisitos a una especificación que describe cómo implementar el sistema. El análisis consiste en obtener una visión del sistema que se preocupa de ver qué hace, de modo que solo se interesa por los requisitos funcionales. Por otro lado el diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, en definitiva cómo cumple el sistema sus objetivos. En esta disciplina se generan una serie de artefactos como: Modelo de Análisis, Modelo de Diseño, Modelo de Datos y Modelo de Despliegue. [3]

Para el SIGEP se realizó un ajuste donde los principales artefactos generados son: Modelo de Diseño y el Modelo de Datos.

Implementación

En este flujo de trabajo se implementan las clases y objetos en ficheros fuente, binarios y ejecutables. Además se deben hacer las pruebas de unidad: cada implementador es responsable

de probar las unidades que produzca. El resultado final de este flujo de trabajo es un sistema ejecutable. La estructura de todos los elementos implementados forma el modelo de implementación. La integración debe ser incremental, es decir, en cada momento sólo se añade un elemento. De este modo es más fácil localizar fallos y los componentes se prueban más a fondo. En fases tempranas del proceso se pueden implementar prototipos para reducir el riesgo. Su utilidad puede ir desde ver si el sistema es viable desde el principio hasta probar tecnologías o diseñar la interfaz de usuario. Los prototipos pueden ser exploratorios (desechables) o evolutivos. Estos últimos llegan a transformarse en el sistema final. [3]

1.5.2. Patrones de diseño del software

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectivo y reusable. El objetivo es agrupar una colección de soluciones de diseño que son válidas en distintos contextos y que han sido aplicadas con éxito en otras ocasiones. Los patrones son soluciones de sentido común que deberían formar parte del conocimiento de un diseñador experto. Además facilitan la comunicación entre diseñadores, pues establecen un marco de referencia (terminología, justificación). Por otro lado, facilitan el aprendizaje al programador inexperto, pudiendo establecer parejas problema-solución. A continuación se hace referencia a algunos de los patrones que se utilizaron en la propuesta de solución técnica:

- **Model-View-Controller (Modelo-Vista-Controlador, MVC)**: Divide una aplicación en tres componentes, el modelo, las vistas y los controladores. El modelo contiene la funcionalidad y los datos del sistema, las vistas muestran información al usuario y los controladores manejan la interacción del usuario. Las vistas y controladores comprenden la interfaz de usuario. MVC permite crear diferentes vistas para el mismo modelo incluso en tiempo de ejecución.

- **Facade (Fachada)**: Simplifica el acceso a un conjunto de objetos proporcionando uno, llamado fachada, que los clientes pueden usar para comunicarse con el conjunto. Reduce el número de objetos con los que tiene que interactuar un cliente proporcionando un menor acoplamiento y facilitando el cambio de componentes sin afectar a sus clientes.

- **Front-Controller (Controlador Frontal)**: El patrón propone utilizar un controlador como el punto inicial de contacto para manejar las peticiones del usuario en una aplicación, incluyendo la invocación de los servicios de seguridad, la elección de una vista apropiada, el manejo de errores, y el control de la selección de estrategias de creación de contenido.

- **Data Access Object (DAO)**: Centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Sus principales beneficios son: reduce la complejidad de los objetos de negocio; al abstraerlos de la implementación real de la comunicación con la fuente de datos y permite una migración más fácil de fuente de datos.

- **Controller (Controlador)**: Este patrón propone asignar la responsabilidad de controlar el flujo de eventos de un sistema, a clases específicas llamadas controladores. Los controladores no ejecutan las tareas sino que las delegan en otras clases, con las que mantiene un modelo de alta cohesión.

- **Strategy (Estrategia)**: Se utiliza cuando se necesitan diferentes variantes de un algoritmo o cuando una clase define muchos comportamientos los cuales se manifiestan como definiciones condicionales múltiples de sus operaciones. Este patrón propone encapsular los algoritmos en diferentes clases, eliminando las costosas definiciones de comportamiento multicondicionales. Brinda gran flexibilidad para futuras extensiones y tiene como desventaja que puede producir una gran explosión en el número de objetos del sistema.

1.5.3. Estándares de codificación

Las convenciones de código son importantes para los programadores por un gran número de razones, ya que el 80% del coste del código de un programa va a su mantenimiento. Mejoran la lectura del software, permitiendo entender código nuevo mucho más rápido y más a fondo. En el desarrollo del SIGEP se incluyen las definidas para el lenguaje Java por Sun Microsystems. Para la documentación de las clases se utilizan las convenciones de Javadoc adaptadas para el SIGEP.

1.5.4. Plataforma de desarrollo

Lenguaje de programación

El lenguaje de programación Java desarrollado por Sun Microsytems presenta características muy favorables; es un lenguaje libre pudiendo ser utilizado el compilador y la máquina virtual de forma gratuita. Entre sus características se encuentran: es un lenguaje simple, orientado a objetos e independiente de la arquitectura. El lenguaje en sí mismo tiene un modelo de objetos más simple y elimina herramientas de bajo nivel que suelen inducir a muchos errores, como la manipulación directa de punteros memoria. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

JEE

JEE es una tecnología que apunta simplificar el diseño y desarrollo de aplicaciones empresariales portables, robustas, escalables y seguras utilizando Java como lenguaje. No solo proporciona un conjunto de APIs sino también una infraestructura en tiempo de ejecución, esta infraestructura la proporciona un tipo especial de aplicaciones. Las aplicaciones desplegadas con la tecnología JEE están estandarizadas, siguen una guía de especificación estrictas, están escritas en Java y se pueden desplegar en cualquier servidor. JEE define tres tipos de componentes que un programador puede desarrollar Servlets, JSP y Enterprise Java Beans.

Java-Servlets

Los servlets son objetos que se crean dentro del contexto de un contenedor de servlets, proporcionan un servicio de petición–respuesta basada en un determinado protocolo. Los Java-Servlets proporcionan un método para escribir programas del lado del servidor, su uso común es la generación de páginas web dinámicas.

Java Server Pages (JSP)

Java Server Pages (JSP) es la forma simplificada y rápida de crear contenido web dinámico. La tecnología JSP permite el desarrollo rápido de aplicaciones web que son de servidor y plataforma independiente. La tecnología Java Server Pages permite poner segmentos de código servlet directamente dentro de una página HTML estática. Cuando el navegador carga una página JSP

se ejecuta el código del servlet y el servidor de aplicaciones crea, compila, carga y ejecuta un servlet en segundo plano; para ejecutar los segmentos de código servlet y devolver una página HTML o imprimir un informe XML.

JDBC

Java Database Connectivity (JDBC) API es el estándar de la industria de la conectividad de base de datos independiente entre el lenguaje de programación Java y una amplia gama de bases de datos como la base de datos SQL y otras fuentes de datos tabulares (hojas de cálculo o ficheros planos). JDBC generaliza las funciones más comunes de acceso a los datos, abstrayendo los detalles específicos de un proveedor de determinada base de datos. [4] El resultado es un conjunto de clases e interfaces que pueden ser utilizadas con cualquier gestor que tenga un controlador JDBC apropiado.

1.5.5. Frameworks

Un Framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Spring Framework

Es potente en cuanto a la gestión del ciclo de vida de los componentes y fácilmente ampliable. Ofrece un ligero contenedor de bean para los objetos de la capa de negocio, DAOs, repositorio de Datasources JDBC y sesiones Hibernate. Spring proporciona:

- Una potente gestión de configuración basada en JavaBeans, aplicando los principios de Inversión de Control (IoC). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Estas definiciones de beans se realizan en lo que se llama el contexto de aplicación. [5]
- Una capa genérica de abstracción para la gestión de transacciones; permitiendo gestores de transacción añadibles (pluggables) y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel. [5]

- Una capa de abstracción JDBC que ofrece una significativa jerarquía de excepciones (evitando la necesidad de obtener de SQLException los códigos que cada gestor de base de datos asigna a los errores), simplifica el manejo de errores y reduce considerablemente la cantidad de código necesario. [5]
- Integración con Hibernate, JDO e iBatis SQL Maps en términos de soporte a implementaciones DAO y estrategias con transacciones. Especial soporte a Hibernate añadiendo convenientes características de IoC y solucionando muchos de los comunes problemas de integración de Hibernate. [5]
- Funcionalidad AOP, totalmente integrada en la gestión de configuración de Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa. [5]
- Un framework MVC (Model-View-Controller) construido sobre el núcleo de Spring. Este framework es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista. De cualquier manera una capa modelo realizada con Spring puede ser fácilmente utilizada con una capa web basada en cualquier otro framework MVC, como: Struts, WebWork o Tapestry. [5]

Toda esta funcionalidad puede usarse en cualquier servidor JEE y la mayoría de ella ni siquiera requiere su uso. El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reutilizables, no atados a servicios JEE específicos. La arquitectura en capas de Spring ofrece mucha flexibilidad y todas las funcionalidades están construida sobre los niveles inferiores. Spring está estructurado por varios módulos como AOP, ORM, DAO, JMX, JCA, JMS, Web, Context, MVC, Portlet MVC, Remoting y Core. Los más utilizados en la implementación del SIGEP son:

- **MVC:** El paquete Web 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.

- **ORM:** El paquete ORM provee capas de integración para APIs de mapeo objeto - relacional, incluyendo: JDO, Hibernate e iBatis. Además brinda algunos servicios como la integración con el mecanismo de transacciones declarativas de Spring y el manejo transparente de excepciones.

Framework Hibernate

Hibernate es un framework ORM para la plataforma Java disponible además para la plataforma .NET con el nombre de Nhibernate. Es un poderoso objeto de alto rendimiento, persistencia relacional y servicio de consulta. Permite desarrollar clases persistentes siguiendo un lenguaje orientado a objetos incluyendo: la asociación, herencia, polimorfismo, la composición y las colecciones. Además de expresar consultas en su propia extensión de SQL portátil (HQL), así como en SQL nativo. Ofrece sofisticadas opciones de consulta, puede escribir SQL de fricción, orientado a objetos HQL (Hibernate Query Language) o crear los criterios programáticos y consultas de ejemplo. Se adapta a cualquier proceso de desarrollo, ya sea comenzando un diseño desde cero o trabajando con una base de datos existente y apoyará cualquier arquitectura de aplicaciones.

AJDT (AspectJ Development Tools)

Es un Plug-in que provee herramientas para el desarrollo orientado a aspectos, usando AspectJ Framework. Algunos aspectos de la aplicación del sistema, tales como el manejo de errores, las normas de aplicación y función de las variaciones son muy difíciles de implementar en forma modular. El resultado es que el código está enredado en un sistema y conduce a la calidad, productividad y problemas de mantenimiento. Aspect Oriented Software Development permite la modularización limpia de estas preocupaciones transversales, tales como la comprobación de errores y la manipulación, la sincronización, el contexto de un comportamiento sensible, optimizaciones de rendimiento, control de registro, la depuración de apoyo y multi-objeto de protocolos.

1.5.6. Gestor de Base de Datos

Oracle

Oracle es básicamente un herramienta cliente/servidor para la gestión de base de datos. Es un manejador de base de datos relacional que hace uso de los recursos del sistema informático en todas las arquitecturas de hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información. Es el conjunto de datos que proporciona la capacidad de almacenar y acudir a estos de forma recurrente con un modelo definido como relacional. Además es una suite de productos que ofrece una gran variedad de herramientas.

1.5.7. Entorno integrado de desarrollo

Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido". Es una plataforma de desarrollo Open Source basada en Java, desarrollado por IBM cuyo código fuente fue puesto a disposición de los usuarios. En sí mismo Eclipse es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (plug-in).

El SIGEP utiliza una serie de plug-ins, que permite agilizar el desarrollo de la implementación a los programadores. A continuación se hace referencia de los mismos:

Hibernate Tools: Constituye un conjunto de herramientas para facilitar el uso del frameworks Hibernate. Las principales funcionalidades que brinda son:

- **Editor de Mapeo:** Un editor de archivos de mapeo Hibernate XML, brinda el apoyo a la auto-realización y el resaltado de sintaxis. Soporta auto-semántica de finalización de nombres de clase, de propiedad, nombres de campos, nombres de tabla y columna de nombres. [6]
- **Consola:** La perspectiva de Hibernate Console permite configurar las conexiones de base de datos, proporciona una visualización de las clases y sus relaciones y permite ejecutar consultas HQL interactiva en contra de su base de datos y navegar en los resultados de la consulta. [6]
- **Ingeniería inversa:** La característica más poderosa de Hibernate Tools es una herramienta de ingeniería inversa de bases de datos que puede generar las clases del modelo de dominio y los archivos de mapeo de Hibernate. [6]

- **Asistentes (wizards):** Proporciona varios asistentes, incluyendo asistentes para generar rápidamente la configuración de archivos Hibernate (cfg.xml) y configuraciones de la consola de Hibernate. [6]

Spring IDE v2.0 (2007): Complemento para el desarrollo de aplicaciones que utilicen Spring desde Eclipse, contiene un editor de ficheros XML, que permite autocompletado, además de un validador y visor de los beans configurados. Permite la búsqueda y visualización en forma de grafo de todos los beans y dependencias. [7]

SDE v4.4.1 (2007): Smart Development Environment (SDE) para Eclipse proporciona la plataforma de desarrollo unificado de desarrollador de Java para la captura de requisitos, diseño de software, diseño de base de datos, generar código, la aplicación de software y generar informes. SDE permite la realización todos los tipos de diagramas UML en Eclipse. [8]

Subclipse v1.0.1: Subclipse es un plug-in de Eclipse que proporciona la funcionalidad para interactuar con un servidor de Subversion, y para manipular un proyecto en un servidor de Subversion en el entorno de Eclipse, permitiendo operaciones de sincronización, actualización, entre otras. Facilita el trabajo en colectivo, ya que en el intervienen un conjunto de desarrolladores trabajando en el mismo proyecto. [9]

- **Subversion**: Subversion es un sistema de gestión de configuración que permitir que varios miembros de un equipo puedan comprobar la última versión del código de un repositorio, hacer sus cambios en el código, y luego confirmar de nuevo los archivos al mismo tiempo. Permite al programador saber si existen otras modificaciones entre su última descarga y carga posterior.

1.5.8. Biblioteca

Dojo Toolkit

Es una biblioteca JavaScript de código abierto, que contiene APIs y controles (widgets) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Provee un conjunto de componentes de interfaz gráfica de usuario como calendarios y menús, que se pueden insertar de manera sencilla en páginas HTML.

1.5.9. Herramienta de modelado

Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Facilita una excelente interoperabilidad con otras herramientas CASE y la mayoría de los IDEs. Permite generar código inverso, código desde diagramas, documentación y dibujar todos los tipos de diagramas. Visual Paradigm ofrece dos modalidades: el UML Edition y Smart Development Environment (SDE) para la integración con IDEs de desarrollo como el eclipse.

ER/Studio

Embarcadero ER/Studio es una herramienta para el diseño de base de datos, que brinda productividad en su diseño, generación y mantenimiento de aplicaciones; desde un modelo lógico de los requerimientos de información, hasta el modelo físico perfeccionado para las características específicas de la base de datos diseñada. ER/Studio soporta principalmente bases de datos relacionales SQL y bases de datos que incluyen Oracle, Microsoft SQL Server, Sybase. El mismo modelo puede ser usado para generar múltiples bases de datos, o convertir una aplicación de una plataforma de base de datos a otra.

1.5.10. Contenedor Web (Apache Tomcat)

Apache Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Sun Microsystems. Dado que fue escrito en Java, funciona en cualquier sistema operativo que disponga de una máquina virtual Java. Puede funcionar como servidor web por sí mismo, es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad y su nivel de utilización ha aumentado debido a su contrastada estabilidad.

1.6. Conclusiones

A partir del análisis realizado a sistemas similares en el área geográfica, en el que se va a implementar finalmente el SIGEP, se concluyó que no son una solución portable para el desarrollo de los procesos Pertenencias y Medidas Disciplinarias, muy ligados al proceder específico del Sistema Judicial y Penitenciario venezolano. Las características de los procesos analizados, como parte del núcleo del SIGEP, reflejaron la importancia de la informatización de los mismos para el Sistema Penitenciario. Basado en el hecho de que SIGEP es una aplicación web de gestión empresarial cuyo equipo de desarrollo es numeroso, se utiliza una adecuación para el proyecto de la metodología de desarrollo de software RUP, basándose la realización del mismo en los flujos de trabajo: Diseño e Implementación. Las tecnologías a utilizar contienen tendencias recientes de la programación sobre la plataforma JEE como la programación orientada a aspectos y la inyección de dependencias. Además los frameworks utilizados se consideran de referencia en su ámbito como lo es Spring como framework de aplicación e Hibernate como framework de soporte para la capa de acceso a datos. El IDE Eclipse con su sistema de plug-ins es una herramienta provechosa pues agiliza del trabajo de implementación con estos frameworks y otros abordados en el capítulo. La aplicación se desarrolla con herramientas y tecnologías en su mayoría libres y multiplataforma. Esto trae como consecuencia positiva una reducción notable del costo del sistema por concepto de licencias de software y soporte.

CAPÍTULO 2: ARQUITECTURA DEL SISTEMA

2.1. Introducción

En el siguiente capítulo se describe la arquitectura utilizada para desarrollar los módulos de Pertenencias y Medidas Disciplinarias, la cual está regida por la arquitectura base del SIGEP, descrita en el Documento de Arquitectura de Software del mismo, dentro del cual se enmarcan los módulos antes mencionados. La misma está basada en los estilos arquitectónicos Cliente-Servidor y en Capas, específicamente tres capas lógicas fundamentales bien definidas. Cuenta con la capa Presentación, Lógica de Negocio y Acceso a Datos. Cada capa puede ser modificada tanto como sea necesario sin provocar cambios en las demás. Una capa no tiene dependencias con la capa superior, cada una depende solamente de la fachada que le permite comunicarse con la capa inmediata inferior. Esta dependencia entre capas es normalmente a través de fachadas, asegurando que el acoplamiento sea el más bajo posible y la abstracción del funcionamiento de la capa inferior, sea casi total. A continuación se realizará un análisis más detallado de la Arquitectura Cliente-Servidor y Arquitectura en capas, además se explicará cada una de las capas propuestas anteriormente, comenzando de abajo hacia arriba.

2.1.1. Arquitectura Cliente-Servidor

La arquitectura cliente servidor se encuentra dentro de la clasificación de estilo, llamada-retorno. El cliente y el servidor generalmente están localizados en diferentes sistemas, sin embargo pueden encontrarse en el mismo. El cliente es la entidad que hace la petición por un servicio. El servidor es la entidad que provee el servicio correspondiente a la petición. El servicio debe procurar el resultado, el cual es retornado.

2.1.2. Arquitectura en capas

La plataforma JEE define un modelo de programación encaminado a la creación de aplicaciones basadas en capas. Típicamente una aplicación puede tener cinco capas diferentes:

- Capa de cliente: Representa la interfaz de usuario que maneja el cliente.

- Capa de presentación: Representa el conjunto de componentes que generan la información que se representará en la interfaz de usuario del cliente. Típicamente se creará a través de componentes basados en Servlets y JSP.

- Capa de lógica de negocio: Contiene los componentes de negocio reutilizables. Normalmente se forman a partir de componentes EJB (Enterprise Java Beans).

- Capa de integración: En esta capa se encuentran componentes que permiten hacer más transparente el acceso a la capa de sistemas de información. Por ejemplo este es el lugar idóneo para implementar una lógica de objetos de acceso a datos, DAO (Data Access Objects).

- Capa de sistemas de información: Esta capa engloba los sistemas de información: bases de datos relacionales, bases de datos orientadas a objetos, sistemas heredados, etc.

Las ventajas para la solución informática son palpables. Al tener las capas separadas existe poco acoplamiento entre las mismas, de modo que es mucho más fácil hacer modificaciones en ellas sin que interfieran en las demás. Todo esto redunda en la obtención de mejoras en cuanto a mantenibilidad, extensibilidad y reutilización de componentes.

Capa de Acceso a Datos

La capa de acceso a datos es la encargada de persistir, consultar, actualizar y eliminar los objetos de dominios recibidos de la capa lógica. Esta capa contiene los objetos que encapsulan la lógica de acceso a datos (DAO) e interfaces brindadas para ser accedida desde la capa de negocio. Las implementaciones de los DAOs extienden clases de soporte del framework Spring para el uso de este patrón usando el framework ORM Hibernate, mientras que las interfaces se mantienen independientes de Spring e Hibernate.

Capa de Lógica de Negocio

Es la encargada de procesar toda la información que será persistida en la base de datos o mostrada en la capa de presentación. La definición de arquitectura del sistema propone que los objetos de dominio no presenten ningún tipo de lógica de negocio, dejando caer esta

responsabilidad sobre los objetos de negocio, permitiendo usar a los objetos de dominio como Transfer Objects que se mueven entre las capas arquitectónicas de la aplicación.

Capa de Presentación

Es la encargada de interactuar con el usuario, obtener, validar y enviar los datos al servidor. Debido al uso de Spring MVC estará compuesta por tres tipos de objetos: Modelo, Vista y Controlador.

- **Controlador**: Es el responsables de procesar las entradas del usuario en forma de peticiones HTTP (Hyper Text Transfer Protocol), invocando las funcionalidades necesarias, expuestas por la capa de lógica de negocio y devolviendo un modelo requerido para ser mostrado.

- **Modelo**: Contiene los datos resultantes de la ejecución de la lógica de negocio, los cuales deberían ser mostrados en la respuesta.

- **Vistas**: Son las encargadas de mostrar los datos del modelo que han sido suministrados por un controlador como respuesta a una petición. La forma de mostrar el modelo podrá ser con diferentes tipos de vistas como JSP (Java Server Page), HTML (Hyper Text Markup Languaje), documentos PDF, documentos de Excel e imágenes.

2.2. Estructura de paquetes del SIGEP

En la estructura de paquetes del SIGEP se muestran todos los subsistemas, dentro de los cuales se encuentran los diferentes módulos correspondientes a cada uno de ellos. La arquitectura del sistema posee una estructura de paquetes que divide tanto lógica como físicamente las tres capas arquitectónicas y junto a ello los recursos que se generan.

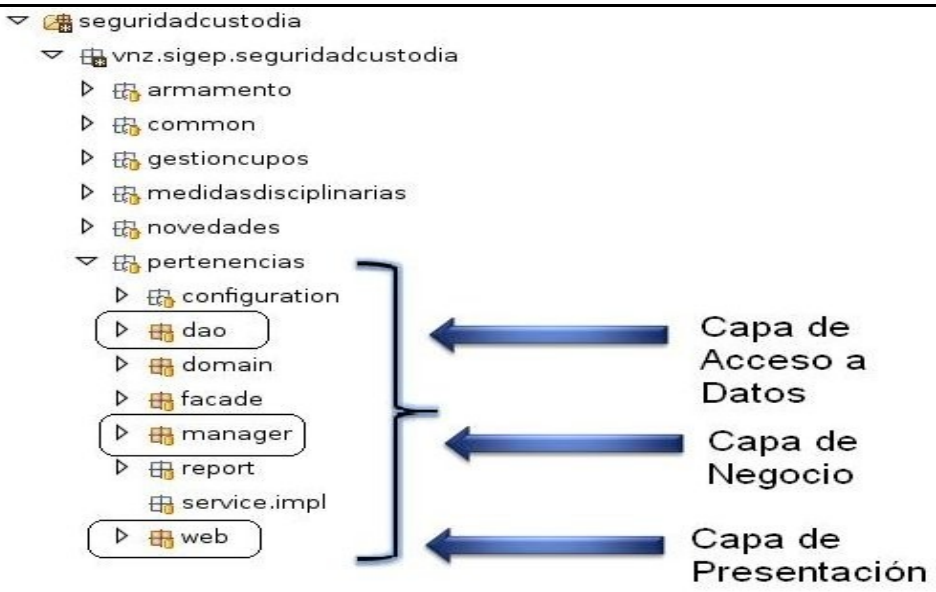


Figura 2.1 Estructura de paquetes del módulo Pertenencias

Esta estructura de paquete permite mantener una buena organización de las actividades a realizar dentro de las diferentes capas. La capa raíz llamada SIGEP, agrupa todos los subsistemas de la aplicación, dentro de los cuales se encuentran los diferentes módulos, estructurados de forma tal que estén presente las tres capas lógicas. El paquete web contendrá todos los recursos que se generen dentro de la capa de presentación, ya sean Controllers, Commands, Validator, PropertyEditors, el paquete manager contiene la implementación del negocio y el paquete dao posee la implementación del acceso a datos, los map, mock y los test. El paquete configuration contendrá las configuraciones del módulo, en forma de mapeo de peticiones, controladores y vistas. El paquete facade permitirá la comunicación entre la capa de Negocio y la de Presentación.

2.3. Actividades del diseño e implementación de un módulo del SIGEP

El desarrollo de un módulo del SIGEP generalmente supone una división del trabajo por los roles que lo ejecutarán, así cada capa puede ser diseñada e implementada de forma paralela a otras y el tiempo de desarrollo se acorta en dependencia de la eficiencia del trabajo en equipo. El diseño

e implementación de un módulo es desencadenado por la Descripción del Prototipo de Interfaz de Usuario (PIU) y el Proceso Elemental de Negocio correspondiente. Esta documentación es la especificación de lo que el módulo debe hacer, establecido en acuerdo mutuo y en forma de contrato entre el cliente y el equipo de desarrollo durante la captura de requisitos. Tomando como partida esta documentación, las actividades a realizar, por el diseñador (es) y el implementador (es) son: análisis de las funcionalidades, diseño del dominio, diseño del Modelo de Datos, diseño de la capa de negocio, diseño de la capa de acceso a datos, diseño de la capa de interfaz de usuario, implementación de las entidades del dominio, implementación de las interfaces de los managers, implementación de la interfaz de la fachada, implementación de la capa de acceso a datos, implementación de los controladores, implementación de las páginas JSP y la implementación de la lógica en el cliente. A continuación se detalla cómo se ejecutan cada una de dichas actividades:

2.3.1. Análisis de las funcionalidades

Dada la documentación generada en la captura de requisitos, se realiza en un taller de trabajo el análisis de las funcionalidades a implementar, para obtener como resultado el entendimiento común de las funcionalidades que el sistema provee 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. A partir del flujo básico de navegación y las pautas generales definidas para la aplicación se definen las interfaces gráficas.

2.3.2. Diseño del dominio

El diseño del dominio constituye una entrada principal a las restantes actividades de diseño. En este punto 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. Además se identifican los nomencladores. Las clases del dominio se definen en el paquete domain del 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 2.2 se muestra el diagrama de clases del dominio del módulo de Pertenencias.

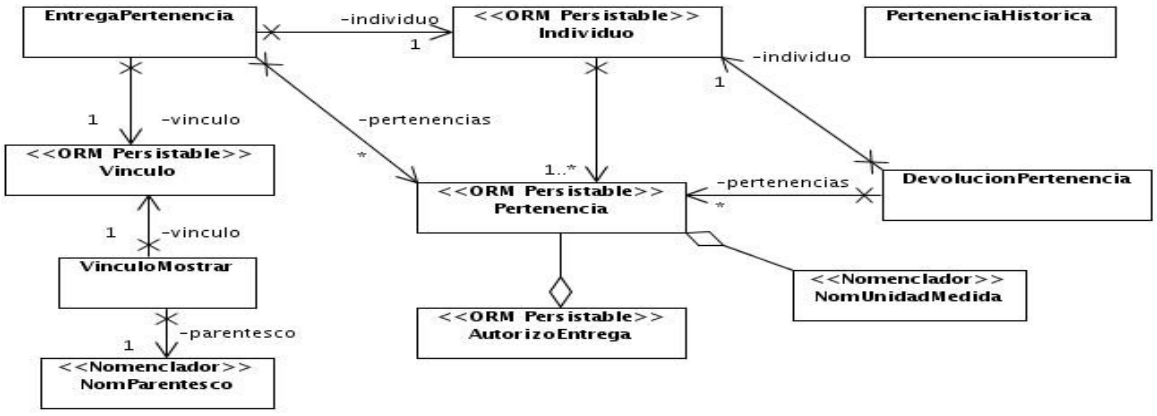


Figura 2.2 Diagrama del dominio del módulo Pertenencias

2.3.3. Diseño del modelo de datos

Al desarrollar esta actividad se obtiene como resultado de cada módulo el modelo de datos. 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 el almacenamiento y recuperación de la información ocurra de manera adecuada y que se cumplan las restricciones identificadas, a través de la integridad referencial. En la Figura 2.3 se muestra el diagrama lógico de la base de datos correspondiente al módulo Pertenencias, generado a partir de la definición de las entidades persistentes. El modelo de datos es refinado en la medida en que se avanza en las actividades de diseño porque sea necesario persistir nuevos elementos o se decida la utilización de procedimientos almacenados, vistas, restricciones y/o funciones definidas en el gestor de base de datos.

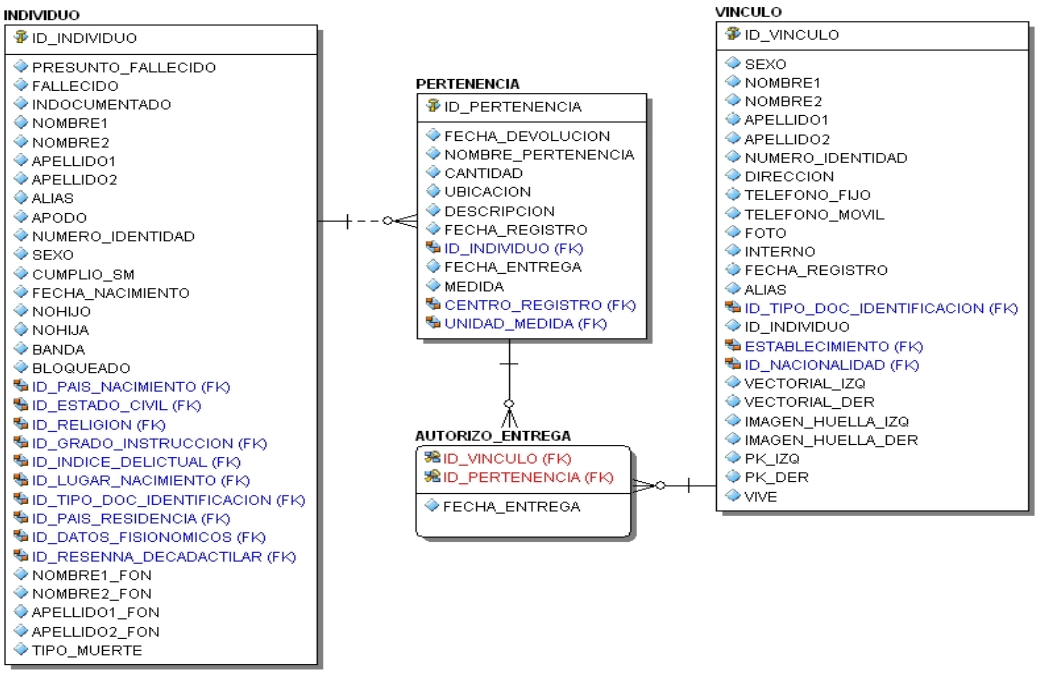


Figura 2.3 Diagrama Entidad Relación del Módulo de Pertenencias

2.3.4. Diseño de la capa de negocio

El diseño de la capa de negocio abarca una serie de clases imprescindibles para cubrir las funcionalidades que se definen en el módulo. Por cada módulo se definirá una o más fachadas que agrupen los métodos de negocio implementados en los manejadores o Managers, en caso de que se requiera. 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 cuales son las únicas clases en la aplicación que tendrán lógica de negocio mientras que las fachadas se limitarán solamente a agrupar las funcionalidades que serán expuestas a capas superiores. En la Figura 2.4 se muestra el diagrama de clases correspondiente al diseño de la capa de negocio del módulo Pertenencias a manera de ejemplo de lo explicado.

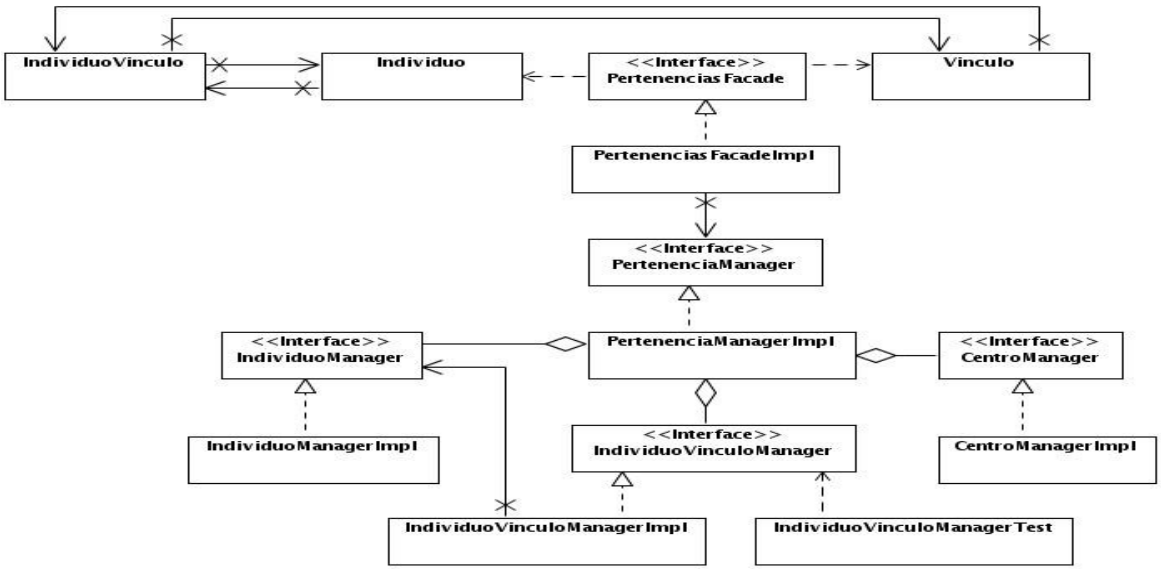


Figura 2.4 Diseño de la capa de negocio del módulo Pertenencias

2.3.5.Diseño de la capa de acceso a datos

El diseño de la capa de acceso a datos está estrechamente relacionado con la capa de negocio, debido a que las funcionalidades de dicha capa aparecen como necesidades de la capa de negocio. Las interfaces de la capa de acceso a datos definen las funcionalidades necesarias relacionadas con la persistencia y recuperación de datos del medio de almacenamiento. Las implementaciones de los DAOs heredan de la clase ***cu.uci.arbaweb.dataaccess.dao.impl.AbstractBaseDAO*** que a su vez extiende la clase ***org.springframework.orm.hibernate3.support.HibernateDaoSupport*** e implementa métodos comunes para todos los DAOs como son ***findById***, ***persist***, ***delete***, ***findByCriteria*** y ***findByExample***. En la Figura 2.5 se muestra el resultado del diseño de la capa de acceso a datos del módulo Pertenencias.

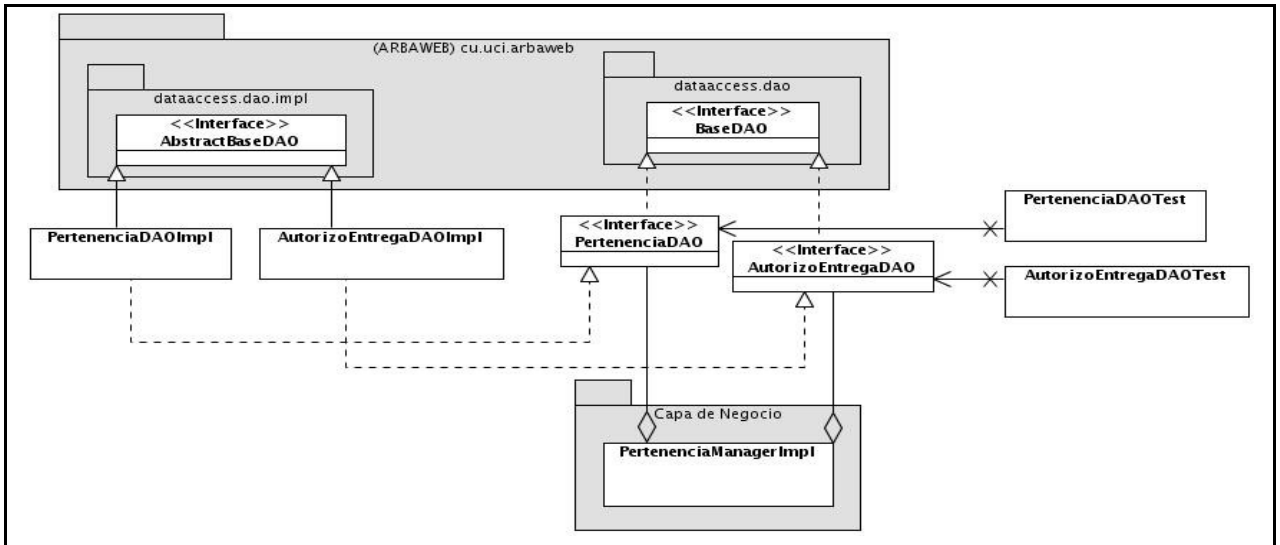


Figura 2.5 Diseño de la capa de acceso a datos del módulo Pertenencias

2.3.6.Diseño de la capa de interfaz de usuario

En el diseño de la capa de interfaz de usuario se definen las vistas y el flujo de navegación, comenzando por las posibles peticiones del usuario y los controladores que las van a atender . El diseño de los controladores por lo general implica diseñar otros componentes que cumplen funciones en el flujo de Spring-MVC como son validadores (validator), objetos de respaldo (command) y property editors. Se definen componentes del cliente, como son clases JavaScript, documentos HTML, imágenes, etcétera. Spring-MVC Framework, es una implementación del patrón arquitectónico llamado MVC (Modelo Vista Controlador), el cual se utilizará en esta capa de presentación. Su uso es de gran importancia y beneficio debido a que este framework provee implementaciones bases de controladores que realizan funciones comúnmente necesarias en aplicaciones web. Ejemplo de esto es la clase ***org.springframework.web.servlet.mvc.SimpleFormController*** que implementa el flujo de un formulario y ***org.springframework.web.servlet.mvc.AbstractWizardFormController*** que implementa el flujo de un asistente. Al definir los controladores se desarrollan estas clases

añadiendo las funcionalidades específicas del formulario o asistente particular, por lo que aumenta la productividad.

2.3.7. Implementación de las entidades del dominio

Las entidades del dominio pueden tener muy poco comportamiento. En la mayoría de los casos los métodos *equals()*, *hasCode()* y *toString()* deben ser implementados, ya que son utilizados con frecuencia. En la implementación de las entidades del dominio se debe refinar la definición de los atributos de las entidades de modo que queden listas en un buen por ciento para implementaciones reusables.

2.3.8. Implementación de las interfaces de los managers

En esta actividad se implementan los método para cada una de las interfaces de los managers, ajustándose a las funcionalidades previstas. Todo método implementado tiene que verificar la integridad de los datos e informar a la capa de interfaz cualquier eventualidad a través de las excepciones definidas. En caso de que sea necesario se implementa la publicación y manipulación de eventos. Los managers implementados se configuran en el fichero de configuración del contexto de Spring correspondiente a la capa de negocio del módulo. El fichero se halla en el paquete configuration y tiene por nombre: sigep-[subsistema]-[módulo]-business-context.xml. Un ejemplo de la implementación del método ***registrarEntregaPertenencias*** que pertenece al manager ***PertenenciaManagerImpl*** se encuentra representado en el Anexo 1, el muestra como debe ser implementado un manager en el SIGEP.

2.3.8.1. Pruebas unitarias a los managers

Por cada método implementado del manager se debe realizar una prueba de caja blanca que verifique la veracidad del código. La clases de pruebas de los manager se definen en el paquete ***manager.impl.test*** y que heredan directa o indirectamente de la clase ***org.springframework.test.AbstractDependencyInjectionSpringContextTests*** de Spring y sobreescribir el método ***protected String [] getConfigLocations ()*** definiendo la ubicación física de los ficheros de configuración del contexto de Spring en los que se halla el objeto que se está probando y sus dependencias, ya sean implementaciones reales o falsas.

Estas pruebas se benefician de la técnica de inyección de dependencias de Spring y se basan en el framework de pruebas JUnit.

2.3.9. Implementación de la interfaz de la fachada

Para la implementación de la interfaz de la fachada se necesita solo conocer los managers en los cuales se encuentran implementadas las funcionalidades necesarias para comunicarlas a la capa de interfaz; de esta manera, la fachada no tiene ninguna lógica de negocio, solo es experta del lugar donde radica la información que necesita y así la utiliza. El uso de la fachada reduce el número de objetos con los que tiene que interactuar esta, ya que simplifica el acceso de la capa de presentación a la capa de negocio. Seguidamente se muestra un fragmento de la implementación de la interfaz de la fachada PertenenciasFacadeImpl, que refleja la ausencia de lógica interna en la implementación del método ***registrarEntregaPertenencias***.

```
public class PertenenciasFacadeImpl implements PertenenciasFacade {

    private PertenenciaManager pertenenciaManager;
    private NomencladoresManager nomencladoresManager;

public void setNomencladoresManager(NomencladoresManager nomencladoresManager) {
              this.nomencladoresManager = nomencladoresManager;
       }
  …

public void registrarEntregaPertenencias(EntregaPertenencia entrega)
                    throws Exception {
              pertenenciaManager.registrarEntregaPertenencias(entrega);}
```

2.3.10. Implementación de la capa de acceso a datos

Esta actividad comprende fundamentalmente la creación de los ficheros de mapeo de Hibernate (hbm.xml) y la implementación de los objetos de acceso a datos. Los ficheros hbm se ubican en el paquete ***dao.impl.map***. Para la creación de estos ficheros se utiliza el plug-in de Eclipse Hibernate Tools, herramienta que desarrolla considerablemente la productividad en esta actividad. La implementación de los DAOs se centra en el uso de los APIs Criteria y Example del Framework Hibernate. Estos APIs son una eficaz herramienta para la creación de consultas complejas de manera relativamente fácil pero, en caso de no satisfacer las necesidades del implementador, se utiliza el HQL (Lenguaje de consultas de Hibernate), el cual permite mayor flexibilidad y, en último

caso, SQL. El uso de las APIs antes mencionadas y del HQL es recomendada porque permite escribir código más sencillo, transparente y tolerante al cambio. Además de esto la implementación de las DAOs está acoplada de forma mínima al gestor de base de datos que, en el caso de SIGEP, es Oracle pero en ocasiones se apoya en el uso de funciones o procedimientos almacenados que residen en este, como fue explicado en la actividad Diseño de la capa de acceso a datos. Los DAOs implementados se configuran en el fichero **sigep-[subsistema]-[módulo]-dataaccess-context.xml**. En el siguiente ejemplo se muestra la configuración de estos objetos, a los cuales se inyecta el sessionFactory de Hibernate.

```
<bean id="pertenenciaDAO"
 class="vnz.sigep.seguridadcustodia.pertenencias.dao.impl.PertenenciaDAOImpl">
    <property name="sessionFactory">
        <ref bean="sessionFactory"/>
    </property>
</bean>
```

2.3.10.1.Pruebas unitarias a los DAOs

La pruebas unitarias a los DAOs se realizan de manera similar a las pruebas de los manager. Las pruebas de los DAOs se definen en el paquete ***dao.impl.test*** del módulo y heredan de la clase ***org.springframework.test.AbstractTransactionalDataSourceSpringContextTests*** de Spring. Cuando heredan de esta clase, los casos de prueba se ejecutan en un contexto de transacciones a las que se puede hacer *rollback* o *commit* en dependencia de los intereses de cada prueba. De esta manera se logra probar las funcionalidades de acceso a datos sin realizar afectaciones a los datos en la base de datos.

2.3.11. Implementación de los controladores

Los controladores que manejan el flujo web de la aplicación se programan partiendo del diseño realizado de la capa de programación. Estos implementan directa o indirectamente la interfaz ***org.springframework.web.servlet.mvc.Controller*** para insertarse en el flujo de Spring-MVC. A través de su fachada estos componentes son los encargados de la comunicación con la capa de negocio. Siendo los responsables de validar los datos de entrada de la aplicación, formatear los datos de salida y gestionar el flujo web, mostrando las vistas correspondientes. Se

configuran en el fichero del contexto de Spring correspondiente a la capa de presentación: **sigep-[subsistema]-[módulo]-servlet.xml**. Un ejemplo de la implementación de un controlador, mostrada en el Anexo 4 es ***RegistrarEntregaPertenenciasController***, que gestiona el flujo del asistente para el registro de entregar pertenencias a un vinculo externo que ha sido autorizado para ello.

2.3.12. Implementación de las páginas JSP

Las páginas JSP construyen documentos HTML con los datos que el controlador le envía. En la programación de las JSP se utiliza principalmente la biblioteca de etiquetas JSTL (Java Standard Tag Library) y las etiquetas de Spring. En el SIGEP el diseño gráfico de las JSP se realiza a través del uso de estilos que radican en el fichero *css*, donde se encuentran todos los estilos de la aplicación.

2.3.13. Implementación de la lógica en el cliente

Para la implementación de la lógica en el cliente se programa, utilizando el lenguaje JavaScript, validaciones en el cliente y comportamiento de componentes gráficos como calendarios, tablas, botones y pantallas emergentes de error o información al usuario. Además, desde el cliente se lanzan peticiones al servidor usando AJAX y JSON-RPC. En el SIGEP se utilizan los componentes de la bibliotecas javascript de componentes gráficos DojoToolkit (Dojo 0.42) los cuales se localizan en el módulo widget de Dojo.

2.4. Conclusiones

En este capítulo al realizar la descripción de la arquitectura del sistema SIGEP, se logra una mejor compresión de la solución desarrollada. La arquitectura basada en capas independientes estructuralmente posibilita distribuir el equipo de trabajo por roles, propiciando la programación de las capas simultáneamente. La descripción de las principales actividades del flujo de trabajo, constituyen una guía orientada a la obtención de resultados parciales. En cada una de las actividades se ejemplifica el uso de los frameworks y tecnologías mencionadas en el Capítulo 1 y la forma de integrarse para lograr el diseño y la implementación de los módulos: Pertenencias y Medidas Disciplinarias.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

3.1. Introducción

En el presente capítulo se da solución a los módulos Pertenencias y Medidas Disciplinarias conforme a su Diseño e Implementación. Se expone una muestra representativa de algunos diagramas e implementaciones y de las funcionalidades distintivas de cada módulo, utilizando para la presentación, la secuencia de actividades definidas para la realización de los módulos dentro del SIGEP. Se presenta el resultado de las validaciones de los módulos, en sesiones de trabajo presenciales con especialistas funcionales y usuarios finales.

3.2. Actividades y aspectos relevantes en el desarrollo de los módulos Pertenencias y Medidas Disciplinarias

En el Capítulo 2 de este documento se expone la secuencia de actividades llevadas a cabo en el diseño e implementación de los módulos Pertenencias y Medidas Disciplinarias, generando como artefactos los Modelos de Diseño e Implementación adaptados al proyecto SIGEP a partir de la definición de los artefactos homónimos de la metodología RUP, analizadas en el Capítulo 1. Dentro de las actividades realizadas se explican las soluciones no triviales y tecnologías que se usaron para cada situación encontrada, partiendo del análisis de las funcionalidades a implementar. En el cuerpo del documento está anexada y referenciada toda la documentación generada en el proceso de diseño.

3.3. Análisis de las funcionalidades

Las funcionalidades a implementar, descritas en los documentos Control de Pertenencia: Descripción de Funcionalidades y Sanciones Disciplinarias: Descripción de Funcionalidades, generada en la captura de requisitos para los módulos Pertenencias y Medidas Disciplinarias son:

Módulo Pertenencia:

1. **Registrar pertenencias a un individuo**: registra las pertenencias de un individuo cuando entra al centro penitenciario.

2. **Modificar pertenencias a un individuo**: modifica las pertenencias registradas de un individuo.
3. **Listar pertenencias de un individuo (en el expediente)**: guarda en el expediente del individuo el listado de sus pertenencias.
4. **Eliminar pertenencias a un individuo**: elimina las pertenencias registradas de un individuo.
5. **Autorizar entrega a vínculo externo**: autoriza la entrega de las pertenencias de un individuo a los vínculos externos.
6. **Devolver pertenencias a un individuo**: devuelve las pertenencias registradas al individuo.
7. **Entregar pertenencias a un vinculo externo autorizado**: entrega las pertenencias del individuo al vinculo externo.

Módulo Medidas Disciplinarias:

1. **Consultar detalles de sanción disciplinaria**: consulta los detalles de la sanción disciplinaria aplicada al individuo.
2. **Registrar sanción disciplinaria**: registra la sanción disciplinaria aplicada al individuo.
3. **Consultar histórico de sanciones disciplinarias del individuo (en el expediente)**: guarda en el expediente del individuo el listado de las sanciones disciplinarias aplicadas.
4. **Histórico de indisciplinas cometidas en el penal**: consulta el histórico de las indisciplinas cometidas por el individuo en el penal.
5. **Registrar indisciplinas**: registra las indisciplinas cometidas por el individuo en el penal.
6. **Detalles indisciplina**: permite mostrar los detalles de las indisciplinas cometidas por el individuo en el penal.
7. **Mostrar indisciplinas no sancionadas (en el expediente)**: muestra las indisciplinas que no han sido sancionadas en el expediente del individuo.

3.4. Diseño del dominio

En el modelo del dominio del módulo Medidas Disciplinarias se realiza una representación de sus elementos principales encabezados por la aplicación de medias disciplinarias que se desglosa en indisciplina y sanción disciplinaria, las cuales se registran en el sistema. Además aporta nomencladores útiles para la programación de la interfaz, como NomClasificacionIndisciplina que clasifica la indisciplina cometida por el recluso según la categoría.

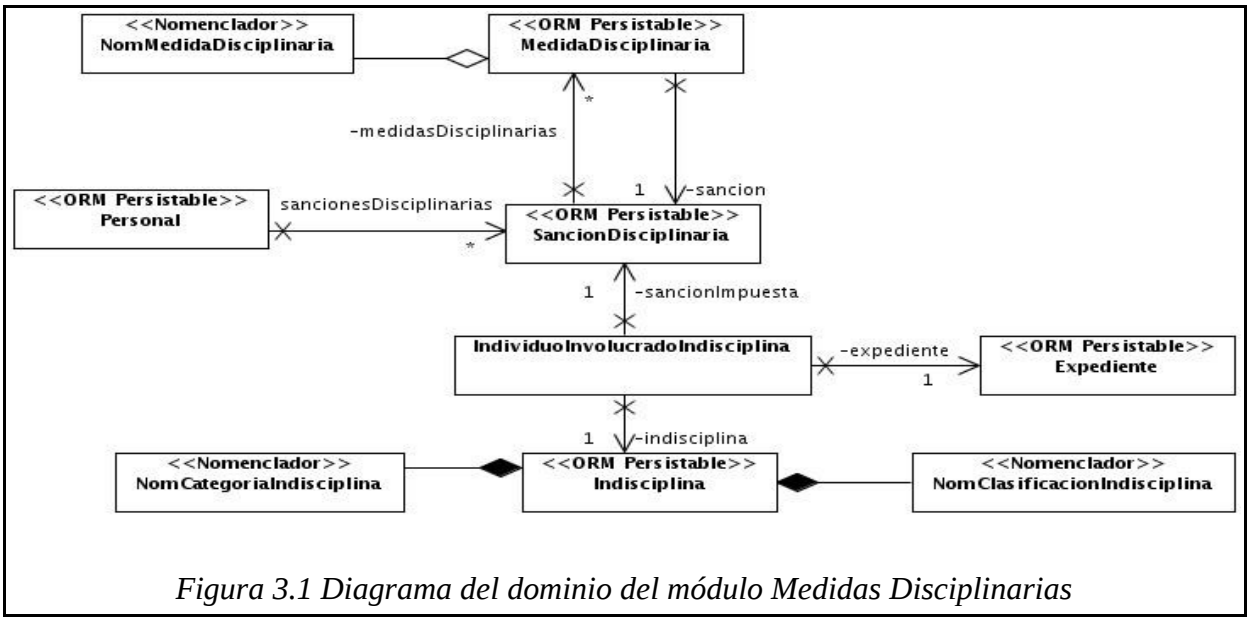


Figura 3.1 Diagrama del dominio del módulo Medidas Disciplinarias

El modelo del dominio representado en la Figura 2.2 muestra las entidades del módulo Pertenencias, además incluye entidades gestionadas por el módulo Datos Personales, agregándole atributos importantes para el módulo en particular. Para la programación de la interfaz se añaden nomencladores que son de gran ayuda para la misma, como ***NomUnidadMedida*** y ***NomParentesco***.

3.5. Diseño del modelo de datos

En el modelo de datos del módulo Medidas Disciplinarias uno de sus casos de uso prioritario es el registro de las indisciplinas de un individuo. Para lograr que el registro se efectivo, se realizan una secuencia de validaciones relacionadas con el tipo de categoría y clasificación, nombre de la falta

disciplinaria, la fecha, hora, lugar donde ocurrió el hecho y la descripción del mismo. En la Figura 3.2 se refleja el modelo de datos con las relaciones existentes entre las tablas sanción con medida aplicada e indisciplina; sanción también está relacionada con el expediente y con el personal a cargo de aplicar la medida, además de las tablas que nomenclan por cada tipo de categoría y clasificación y por el nombre de la medida disciplinaria.

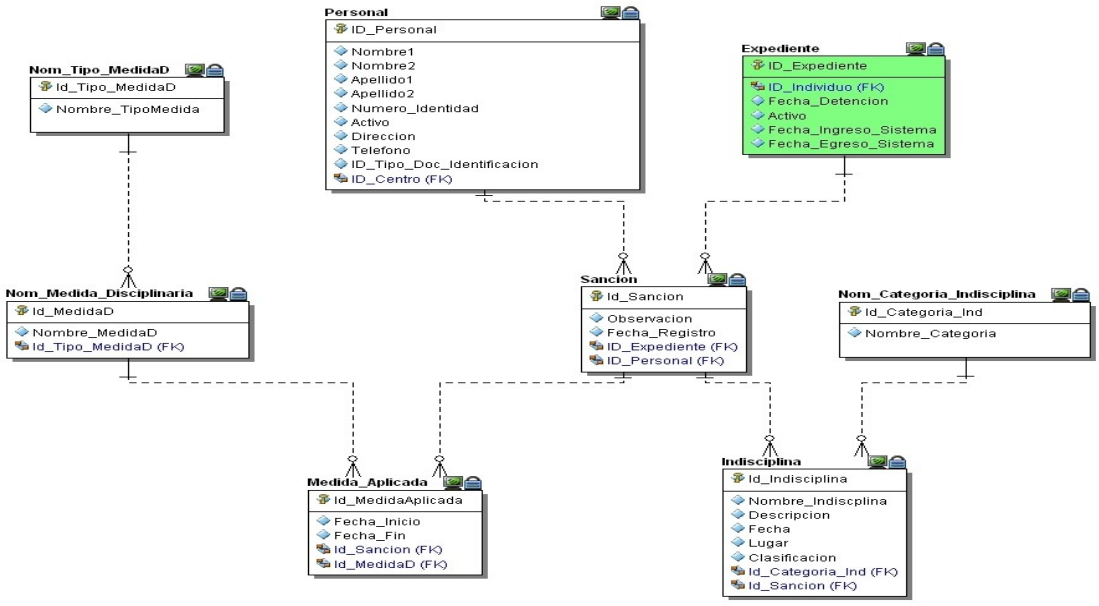


Figura 3.2 Diagrama entidad relación del módulo Medidas Disciplinarias

Por su parte, el módulo Pertenencias al igual que Medidas Disciplinarias tiene como unos de sus casos principales, el registro de las pertenencias de los individuos cuando entran al establecimiento penitenciario. Para la cual se realizan una serie de validaciones relacionadas con la pertenencia, la fecha, cantidad, la unidad de medida y el depósito, para poder realizar un registro efectivo. En el Figura 2.3 se representa el modelo de datos con las relaciones entre las tablas individuo con pertenencias, esta a sus vez con autorizo de entrega y esta última con el vinculo del interno.

3.6. Diseño e implementación de la capa de negocio

Para el diseño de la capa de negocio de los módulos de Pertenencias (Ver Anexo 7) y Medidas Disciplinarias (Ver Anexo 8) se realizó un análisis de las principales restricciones del negocio y a partir de esto el aspecto más relevantes del diseño e implementación de la capa de negocio de dichos módulos son: las transacciones.

3.6.1. Transacciones a nivel de negocio

Las transacciones se encargan de que los datos de una aplicación se mantengan, de acuerdo a las reglas del negocio, en un estado consistente. Para garantizar la consistencia de la información de la base de datos; cada método definido en la interfaz de la capa de negocio se debe ejecutar generalmente como una transacción. Con el objetivo de garantizar la ejecución transaccional de todas las funcionalidades implementadas en los objetos de negocio de los módulos Pertenencias y Medidas Disciplinarias : ***PertenenciaManagerImpl***, ***SancionDisciplinariaManagerImpl*** y ***IndisciplinaManagerImpl***, se utilizan las facilidades que el framework Spring brinda para ello. Para la gestión de transacciones de forma declarativa Spring provee un soporte. Las transacciones son aplicadas en forma de aspectos y se declaran en el fichero de configuración de su contexto.

A continuación se muestra un ejemplo de la aplicación de transacciones de forma de declarativa a un objeto de negocio, donde con la etiqueta `<aop:advisor>` se declara el `pointcut` (punto en que se aplicará el aspecto) y con la etiqueta `<tx:advice>` se declara el aspecto específico es decir, cómo se va a aplicar la transacción . La expresión *execution* significa cuando el método sea ejecutado. La expresión que se encuentra entre paréntesis que en este caso es: **..*SancionDisciplinariaManagerImpl.*(..)* representa los métodos a los que se aplicará la transacción. El primer * significa cualquier tipo de retorno; la expresión **..*SancionDisciplinariaManagerImpl* significa cualquier clase cuyo nombre termine en *SancionDisciplinariaManagerImpl* y la expresión *.*(..)* significa cualquier método con cualquier parámetro. En la etiqueta `<tx:method>` al no especificarse el nivel de aislamiento de la transacción mediante el método ***isolation***, se toma el nivel de aislamiento por defecto del medio almacenamiento, que en el caso de Oracle es lectura confirmada (*read committed*) que evita la

lectura de datos sucios, y con el atributo `rollback-for` se definen las excepciones que, en caso de ser lanzadas, se desea que hagan fallar (*rollback*) la transacción.

```
<aop:config>
   <aop:advisor
        pointcut="execution(**..*SancionDisciplinariaManagerImpl.*(..))"
         advice-ref="sancionDisciplinariaMgrAdvice" />
</aop:config>

<tx:advice id="sancionDisciplinariaMgrAdvice">
   <tx:attributes>
             <tx:method name="*" rollback-for="java.lang.Exception" />
   </tx:attributes>
</tx:advice>
```

En el ejemplo se aplican las transacciones a todos los métodos del objeto ***SancionDisciplinariaManagerImpl*** con esta configuración los objetos de estas clases se ejecutaran en contextos de transacciones sin siquiera ser conscientes de ello.

3.7. Diseño e implementación de la capa de acceso a datos

El diseño de la capa de acceso a datos de las funcionalidades de los módulos Pertenencias y Medidas Disciplinarias se encuentra en el Anexo 7 y Anexo 8 respectivamente.

La implementación de la capa de acceso a datos se basa fundamentalmente en la creación de los ficheros de mapeo de Hibernate (hbm.xml) y la implementación de los objetos de acceso a datos. Toda esta actividad se realiza con el uso del framework Hibernate, que permite asociar a un objeto java una tabla de la base de datos, relacionando así cada atributo del objeto con cada campo de la tabla.

A continuación se muestra un ejemplo de un fichero de mapeo del módulo Medidas Disciplinarias donde el atributo `package` representa el paquete en el cual se encuentra la clase que va a ser mapeada. En la etiqueta `<class>` se especifican los nombres de la clase y de la tabla de la base de datos. Dentro de la etiqueta `<id>` se encuentra el nombre del atributo de la clase que va a almacenar el identificador, el tipo de dato, la columna de la tabla con la cual va a ser mapeada y el generador del identificador, en este caso, es la clase *PrefixPersistentGenerator*. En la

etiqueta <property> se configura un atributo de la clase declarando el nombre del mismo, el tipo de dato y la columna de la tabla con la cual va a ser mapeado. Por cada atributo hay que generar una etiqueta <property>. La etiqueta <many-to-one> significa que hay una relación de uno a muchos, donde se especifica el nombre del atributo, la clase de este atributo (el tipo de dato) y la columna de la tabla con la cual se mapea.

```
<hibernate-mapping
package="vnz.sigep.seguridadcustodia.medidasdisciplinarias.domain">
<class name="Indisciplina" table="INDISCIPLINA">
  <id name="id" type="string">
  <column name="ID_INDISCIPLINA" length="20"/>
  <generator
  class="vnz.sigep.common.global.util.dataaccess.id.PrefixPersistentGenerator">
    <param name="sequence">seq_INDISCIPLINA</param>
    <param name="maxlo">1000</param>
    </generator>
  </id>
  <property name="nombre" type="string">
     <column name="NOMBRE_INDISCPLINA" length="100" not-null="true" />
  </property>

  <property name="descripcion" type="string">
      <column name="DESCRIPCION"/>
  </property>

  <property name="fecha" type="timestamp">
      <column name="FECHA" length="11" not-null="true"/>
  </property>

  <property name="lugar" type="string">
      <column name="LUGAR"/>
  </property>

  <many-to-one name="clasificacion"
       class="vnz.sigep.common.global.domain.NomClasificacionIndisciplina">
       <column name="ID_CLAS_INDISCIPLINA" length="20" not-null="true"/>
  </many-to-one>
  <many-to-one name="categoria"
      class="vnz.sigep.common.global.domain.NomCategoriaIndisciplina">
      <column name="ID_CATEGORIA" length="20" not-null="true"/>
  </many-to-one>
</class>
</hibernate-mapping>
```

3.8. Diseño e implementación de la capa de interfaz de usuario

En el proyecto SIGEP para realizar el diseño e implementación de la capa interfaz de usuario se deben dominar un conjunto de tecnologías como: JavaScript, JSP, Spring-MVC, Jasper Reports y HTML; debido a que son necesarias utilizarlas de acuerdo a los requisitos visuales del usuario del sistema. A continuación se muestra un ejemplo del diseño e implementación de esta capa para la funcionalidad: registrar - modificar indisciplina a un individuo; que tiene como objetivo, como su nombre lo indica registrar y modificar una indisciplina cometida por el interno. Los datos que se registran son: nombre de la indisciplina, fecha, hora y lugar donde ocurrió el hecho, la descripción del hecho, la categoría de la indisciplina, de acuerdo a la categoría se registra la clasificación y además, los individuos involucrados en la falta cometida.

Una vez que el usuario introduce los datos, mediante un botón se realiza la petición web. Dichos datos son validados en el lado del cliente mediante funciones JavaScript en el fichero *registroIndisciplina.js.* A continuación fragmentos de código de este fichero.

```
// Validar los campos del formulario.
validarForm : function() {
  var select = dojo.byId('selCategorias');
  var clasif = dojo.byId('clasificacion');
  var nombre = dojo.byId('nombre');
  var fecha = dojo.widget.byId('indisciplina_fecha_id');
  var hora = dojo.widget.byId('indisciplina_hora_id');
  var lugar = dojo.byId('lugar');
  var descrip = dojo.byId('descripcion');
  var table = dojo.widget.byId('individuos_involucrados_navTable');
   if (select.value == -1) {
       sigep.seguridadcustodia.sanciones.registroIndisciplina.campoInvalido(
       select,sigep.seguridadcustodia.sanciones.registroIndisciplina.locale.
            indisciplinas.camposRequeridos,"ERROR");
             return false;
    }
   if (clasif.value == -1) {
       sigep.seguridadcustodia.sanciones.registroIndisciplina.campoInvalido(
       clasif,sigep.seguridadcustodia.sanciones.registroIndisciplina.locale.
       indisciplinas.camposRequeridos,"ERROR");
 return false;
        }
      ...
  }
  if(fecha.isEmpty()) {
     sigep.seguridadcustodia.sanciones.registroIndisciplina.campoInvalido
     (fecha,sigep.seguridadcustodia.sanciones.registroIndisciplina.locale.
```

```
        indisciplinas.camposRequeridos,"MESSAGE");
      return false;

   } else if(!fecha.isValid()) {
           sigep.seguridadcustodia.sanciones.registroIndisciplina.campo
           Invalido(fecha,sigep.seguridadcustodia.sanciones.registro
           Indisciplina.locale.indisciplinas.datosInvalidos,"ERROR");
            return false;
        }
       ...
   }
```

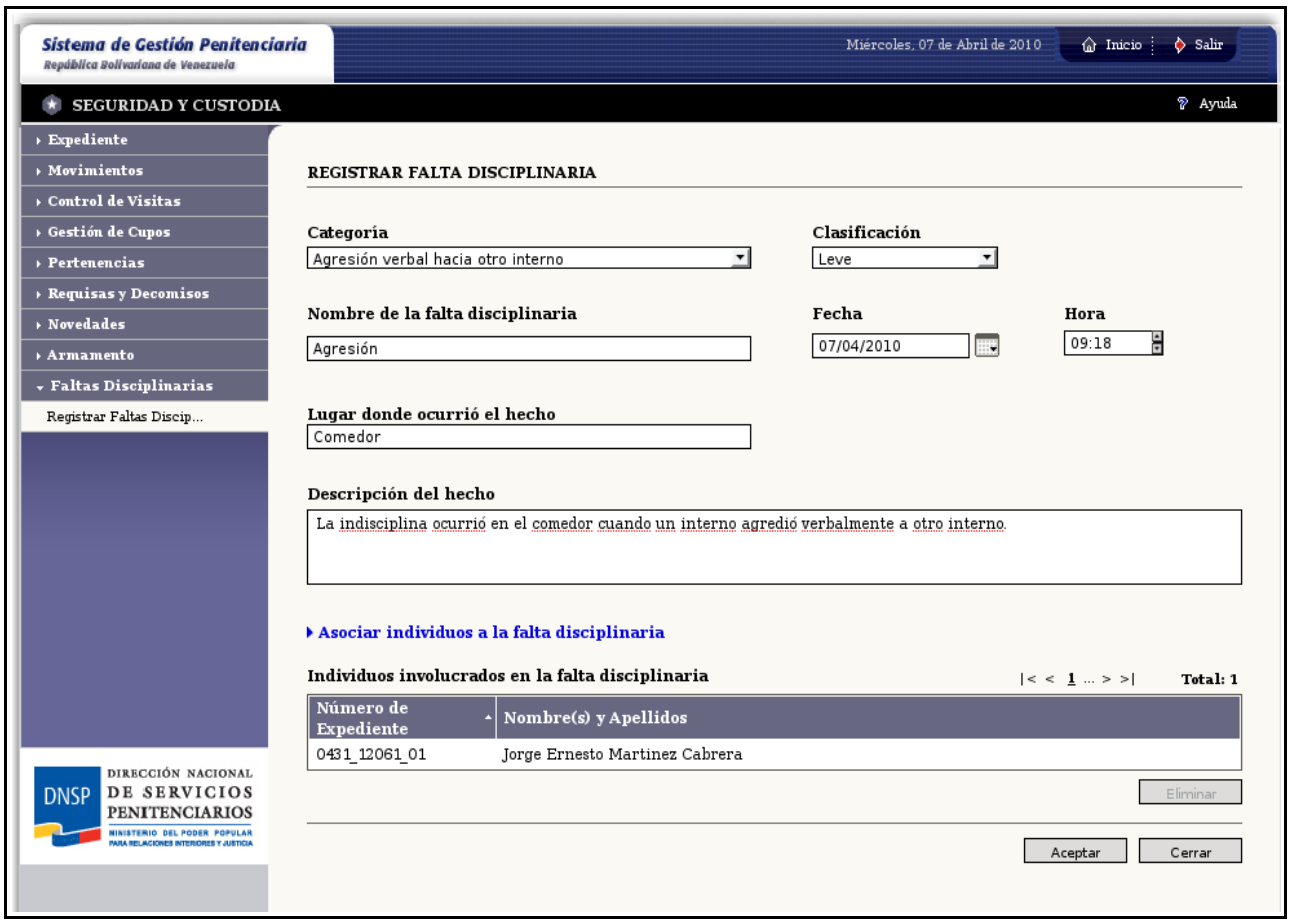
Esta petición es mapeada en el contexto web de Spring de la siguiente forma:

```
<bean id="indisciplinaURLAgrupation"
class="vnz.sigep.administracion.seguridad.util.urlagrupation.URLAgrupation">
      <property name="name" value="Faltas disciplinarias" />
      <property name="pattern" value="/indisciplina" />
       <property name="mappings">
           <props>
                    <prop key="/registrar.htm, Persistir
                              falta">registrarIndisciplinaController
                    </prop>
             </props>
      </property>
</bean>
```

El encargado de manejar esta petición es el controlador *registrarIndisciplinaController* (Anexo 6), el cual es mapeado también en el contexto web de Spring como se muestra a continuación:

```
<bean id="registrarIndisciplinaController"
 class="vnz.sigep.seguridadcustodia.medidasdisciplinarias.web.Registrar
       IndisciplinaController">
      <property name="medidasDisciplinariasFacade">
            <ref bean="medidasDisciplinariasFacade"/>
      </property>
      <property name="nomencladoresFacade">
            <ref bean="nomencladoresFacade"/>
      </property>
      <property name="formView" value="registrarIndisciplina"/>
</bean>
```

Con los datos que el controlador le envía; las páginas JSP construyen documentos HTML, quedando así de la siguiente forma:



3.9. Integración de los módulos Pertenencias y Medidas Disciplinarias al SIGEP

Terminada las actividades del diseño y la implementación de los módulos Pertenencias y Medidas Disciplinarias se realizan las pruebas por el equipo de calidad, luego se empaquetan en ficheros JAR y se declaran componentes estables del SIGEP v2.0 para ser sometidos a prueba de aceptación con el cliente, pasando así a formar parte de la línea base del software.

3.10. Análisis de resultados de la validación con el cliente

Los módulos Pertenencias y Medidas Disciplinarias fueron verificados en dos ocasiones: en una prueba de aceptación con usuarios finales y especialistas funcionales donde se detectaron (2) no

conformidades relacionadas con la interfaz gráfica del módulo Pertenencias y en una prueba piloto en un ambiente real en el establecimiento Mínima de Carabobo del estado de Valencia, en donde se detectó (1) no conformidad y (1) solicitud de cambio del módulo Medidas Disciplinarias. En la siguiente tabla se reflejan cada uno de estos datos.

Pruebas	Módulo	Descripción	Tipo de Petición
Prueba de Aceptación	Pertenencias	Eliminar 'Litros' de las unidades de Medidas.	No Conformidad
Prueba de Aceptación	Pertenencias	Cambiar 'Ubicación' por 'Depósito'.	No Conformidad
Prueba Piloto	Medidas Disciplinarias	Cuando se registra una sanción, el campo observación no está validado para que admita hasta una cantidad determinada de caracteres.	No Conformidad
Prueba Piloto	Medidas Disciplinarias	Adicionar al nomenclador de Medidas "Reclusión en su propia celda.	Solicitud de Cambio

En la última prueba se midieron los resultados de otros módulos del sistema; que pertenecen a la versión 2.0 del SIGEP y como resultado se obtuvo que la media de no conformidades es de 4.3 y la de solicitudes de cambios es de 1. Teniendo en cuenta estos datos el módulo de Medidas Disciplinarias se encuentran por debajo de la media en cuanto a no conformidades y en relación a la cantidad de solicitudes de cambios en la media. En la figura 3.3 se muestran los resultados del módulo, con respecto al resto de los módulos del subsistema Seguridad y Custodia que fueron verificados también en dicha prueba.

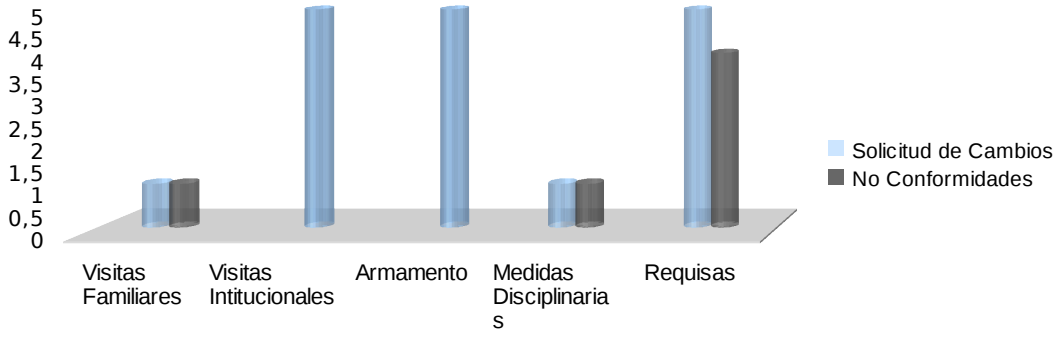


Figura 3.3 Comparación con otros módulos del subsistema Seguridad y Custodia

En sentido general los usuarios, especialistas funcionales e informáticos de la institución han quedado satisfechos con el trabajo realizado. Como prueba de lo anterior se destaca la estabilidad de los requisitos definidos y la robustez de la implementación, de tal forma que no se han generado reportes de soporte técnico en relación a los módulos y para la gestión de cambios solicitada por el cliente en proceso de ejecución desde diciembre 2009 no se tuvo en cuenta ninguno de los módulos.

3.11. Conclusiones

En este capítulo se realiza el diseño y la implementación de los módulos Pertenencias y Medidas Disciplinarias siguiendo el flujo de trabajo definido para el SIGEP. Se siguieron los lineamientos definidos para la arquitectura del sistema, con la utilización de los frameworks se agilizó el proceso de desarrollo de los módulos ahorrando código y tiempo del cronograma. Se realizó la integración de los módulos al SIGEP sin que módulos ya existentes sufrieran cambios o afectaciones. Se realizaron pruebas de validación con el cliente donde todas las no conformidades que se detectaron a los módulos fueron solucionadas inmediatamente. En sentido general los usuarios, especialistas funcionales e informáticos de la institución han mostrado su satisfacción con el trabajo realizado.

CONCLUSIONES

Como resultado del presente trabajo se desarrollaron los módulos Pertenencias y Medidas Disciplinarias del subsistema Seguridad y Custodia del SIGEP v2.0. Siguiendo la arquitectura y herramientas definidas para el sistema, así como los flujos de trabajo; se garantizó la correcta integración de los módulos al SIGEP, sin afectar a otros módulos ni a la solución en general. Durante el proceso diseño e implementación de las funcionalidades se generaron una serie de artefactos que contribuyeron a que la solución propuesta para este trabajo quedara documentada satisfactoriamente. Ambos módulos fueron probados en condiciones reales de trabajo en el Centro Penitenciario Mínima de Carabobo con resultados satisfactorios, evidenciando un correcto funcionamiento del software. El cliente quedó satisfecho con el trabajo realizado, pues contribuye al seguimiento y control de las Pertenencias y Medidas Disciplinarias de los reclusos en las sedes de la DNSP.

RECOMENDACIONES

Se recomienda la realización de un monitoreo constante del rendimiento de los módulos en la medida en que la cantidad de datos que el sistema maneja vaya creciendo, puesto que actualmente el sistema está probado para una cantidad limitada de datos que no ponen al límite la solución propuesta. Además valorar la utilización de las herramientas y arquitectura definida en este trabajo para desarrollar sistemas bajo la plataforma JEE, ya que está probada la eficiencia de las mismas.

REFERENCIA BIBLIOGRÁFICA

1. **Benavides Zayla, Yadira y Gómez Martinez, Juan Carlos**. Diseño e implementación de los módulos Decisiones y Egresos del Sistema de Gestión Penitenciaria de la República Bolivariana de Venezuela. Universidad de las Ciencias Informaticas. Ciudad de La Habana. 2008. p 115

2. **Ossorio, Manuel**. Diccionario de Ciencias Jurídicas, Políticas y Sociales. p 1007

3. **Jacaboson, Ivar y Booch, Grady y Rumbaugh, James**. El Proceso Unificado de Desarrollo de Software. 2000. Addison Wesley. p 438.

4. **Sitio Oficial de JDBC**. 2010. [En línea: http://java.sun.com/jdbc].

5. **Sitio Oficial de Spring**. 2010. [En línea: http://www.springframework.org].

6. **Hibernate Tools for Eclipse and Ant**. 2010. [En línea: https://www.hibernate.org/255.html].

7. **Spring IDE**. 2010. [En línea: http://www.springide.org].

8. **Visual Paradigm SDE for Eclipse**. 2010. [En línea: http://www.visual-paradigm.com /product/ sde/ec].

9. **Supclipse**. 2010. [En línea: http://subclipse.tigris.org].

BIBLIOGRAFÍA CONSULTADA

Cesar Arias, Arturo. Proyecto Técnico de Asesoría Especializada, Colaboración Médica Odontológica, Comunicación Institucional y Solución Tecnológica para apoyar la modernización del Sistema Penitenciario de la República Bolivariana de Venezuela. 2006. p 95. [Consultada: diciembre, 2009.]

República del Ecuador Ministerio de Justicia y Derechos Humanos. [En línea: http://www.minjusticia-ddhh.gov.ec:8000/mjdh/admin/login.jsp]. [Consultada: diciembre, 2009.]

República de Panamá Ministerio de Gobierno y Justicia. [En línea: http://www.sistemapenitenciario.gob.pa]. [Consultada: diciembre, 2009.]

Code Conventions for de Java(TM). 2010. [En línea: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html]. [Consultada: diciembre, 2009.]

Sitio Oficial de Hibernate. 2010. [En línea: http://www.hibernate.org]. [Consultada: diciembre, 2009.]

Sitio Oficial de Oracle. 2010. [En línea: http://www.oracle.com]. [Consultada: diciembre, 2009.]

Sitio Oficial de Eclipse. 2010. [En línea: http://www.eclipse.org]. [Consultada: diciembre, 2009.]

JasperReport. 2010. [En línea: http://www.jasperforge.org]. [Consultada: diciembre, 2009.]

Sitio Oficial Dojo Toolkit. 2010. [En línea: http://www.dojotoolkit.org]. [Consultada: diciembre, 2009.]

Sitio Oficial Visual Paradigm. 2010. [En línea: http://www.visual-paradigm.com]. [Consultada: diciembre, 2009.]

Sitio Oficial Apache Tomcat. 2010. [En línea: http://tomcat.apache.org]. [Consultada: diciembre, 2009.]

AspectJ Development Tools. 2010. [En línea: http://www.eclipse.org/ajdt/]. [Consultada: diciembre, 2009.]

Descripción del Producto. SIGEP v 2.0. [Consultada: diciembre, 2009.]

Pimentel, Luis Alberto y Pérez Rivero, Iósev . Arbaweb: Arquitectura Base sobre la Web. Ciudad de La Habana. 2007. p 95. [Consultada: enero, 2010.]

Pimentel, Luis Alberto. Descripción de la Arquitectura. Ciudad de La Habana. 2007. p 30 [Consultada: enero, 2010.]

Hendrick Kaisler, Stephen. Software Paradigms. 2005. [Consultada: enero, 2010.]

López del Castillo, Javier. Sanciones Disciplinarias: Descripción de Funcionalidades. Cuidad de La Habana. 2008. p 11[Consultada: marzo, 2009.]

Vega Miniet, Yanet. Control de Pertenencia: Descripción de Funcionalidades. Ciudad de La habana. 2008. p 9[Consultada: marzo, 2009.]

ANEXOS

Anexo 1. Implementación del método *registrarEntregaPertenencias* perteneciente al manager *PertenenciaManagerImpl* del módulo Pertenencias.

```
public void registrarEntregaPertenencias(EntregaPertenencia entrega)
       throws Exception {
       InputAssert.notNull(entrega,
                "La entrega de pertenencias no puede ser nula");
       InputAssert.notNull(entrega.getFechaEntrega(),
                "La fecha de entrega de pertenencias no puede ser nula");
       if (entrega.getPertenencias() != null) {

          for (Iterator iterator = entrega.getPertenencias().iterator();
               iterator.hasNext();) {
                Pertenencia pertenencia = (Pertenencia) iterator.next();

              // recupero la pertenencia dada de la BD
              Pertenencia pertenenciaBD = pertenenciaDAO.findById(pertenencia
                     .getId(), false);

     if (pertenenciaBD != null) {
         // la marco como entregada.
         pertenenciaBD.setFechaEntrega(entrega.getFechaEntrega());
         // registro los cambios en la pertenencia
         pertenenciaDAO.persist(pertenenciaBD);
         // busco la autorizacion de recogida de esa pertenencia para
         // ese vinculo
         AutorizoEntrega autorizoEntregaBD = autorizoEntregaDAO
              .findAutorizoByIndividuoVinculo(entrega
                     .getVinculo(), pertenenciaBD);
         // marco la entrega por parte de ese vinculo de la
         // pertenencia
         if (autorizoEntregaBD != null)
              autorizoEntregaBD.setFechaEntrega(entrega
                     .getFechaEntrega());
         // cambios sobre el autorizo
          autorizoEntregaDAO.persist(autorizoEntregaBD);

              }
          }
       }
}
```

Anexo 2. Implementación del método *registrarSancionDisciplinaria* perteneciente al manager SancionDisciplinaria*ManagerImpl* del módulo Medidas Disciplinarias.

```
public void registrarSancionDisciplinaria(
        SancionDisciplinaria sancionDisciplinaria,
        List<IndividuoInvolucradoIndisciplina> asociarIndividuoIndisciplina,
        List<IndividuoInvolucradoIndisciplina> desasociarIndividuoIndisciplina,
        List<MedidaDisciplinaria> registrarMedidaDisciplinaria,
        List<MedidaDisciplinaria> eliminarMedidaDisciplinaria)
        throws Exception {
      InputAssert.notNull(sancionDisciplinaria,"sancionDisciplinaria es
      requerido");
      InputAssert.notNull(sancionDisciplinaria.getFecha(),"sancionDisciplinaria.
      fecha es requerido");
      SancionDisciplinaria sancionDisciplinariaBD = null;
      if (!StringUtils.hasText(sancionDisciplinaria.getId()))
           sancionDisciplinariaBD = sancionDisciplinariaDAO.persist
          (sancionDisciplinaria);
      else {
          sancionDisciplinariaBD = sancionDisciplinariaDAO.findById(
          sancionDisciplinaria.getId(), false);
          // actualizar datos
         sancionDisciplinariaBD.setFecha(sancionDisciplinaria.getFecha());
          sancionDisciplinariaBD.setObservaciones(sancionDisciplinaria
          .getObservaciones());
          sancionDisciplinariaBD.setFuncionario(sancionDisciplinaria
          .getFuncionario());
          sancionDisciplinariaBD = sancionDisciplinariaDAO
          .persist(sancionDisciplinariaBD);}
        // registrar medidas
        if (registrarMedidaDisciplinaria != null) {
           for (MedidaDisciplinaria medidaDisciplinaria :
               registrarMedidaDisciplinaria) {
                medidaDisciplinaria.setSancion(sancionDisciplinariaBD);
                medidaDisciplinariaDAO.persist(medidaDisciplinaria);
            }
        }
        // eliminar medidas
        if (eliminarMedidaDisciplinaria != null) {
            for (MedidaDisciplinaria medidaDisciplinaria :
                eliminarMedidaDisciplinaria) {
                 InputAssert.hasText(medidaDisciplinaria.getId(),"medida
                disciplinaria a eliminar debe tener ID");
                 medidaDisciplinaria.setSancion(sancionDisciplinariaBD);
                 medidaDisciplinariaDAO.delete(medidaDisciplinariaDAO.findById(
                 medidaDisciplinaria.getId(), true));
}
        }
        // asociar sancion a los individuos(IndividuoInvolucradoIndisciplina)
        if (asociarIndividuoIndisciplina != null) {
```

```
            for (IndividuoInvolucradoIndisciplina
                individuoInvolucradoIndisciplina :
                asociarIndividuoIndisciplina) {
              InputAssert.notNull(individuoInvolucradoIndisciplina.
             getExpediente(),
              "individuoInvolucradoIndisciplina.expediente es requrido");
              InputAssert.notNull(individuoInvolucradoIndisciplina
              .getIndisciplina(),
              "individuoInvolucradoIndisciplina.indisciplina es requrido");
              individuoInvolucradoIndisciplina.setSancionImpuesta
             (sancionDisciplinariaBD);
              individuoInvolucradoIndisciplinaDAO.persist
             (individuoInvolucradoIndisciplina);
            }
      }
     // desasociar sancion a los individuos(IndividuoInvolucradoIndisciplina)
     if (desasociarIndividuoIndisciplina != null) {
         for (IndividuoInvolucradoIndisciplina
           individuoInvolucradoIndisciplina : desasociarIndividuoIndisciplina){
            InputAssert.notNull(individuoInvolucradoIndisciplina.getExpediente(),
            "individuoInvolucradoIndisciplina.expediente es requrido");
            InputAssert.notNull(individuoInvolucradoIndisciplina
           .getIndisciplina(),
          "individuoInvolucradoIndisciplina.indisciplina es requrido");
           individuoInvolucradoIndisciplina.setSancionImpuesta(null);
           individuoInvolucradoIndisciplinaDAO.persist
         (individuoInvolucradoIndisciplina);
      }
   }
}
```

Anexo 3. Implementación del método *registrarIndisciplina* perteneciente al manager *IndisciplinaManagerImpl* del módulo Medidas Disciplinarias.

```
public void registrarIndisciplina(Indisciplina indisciplina,
        List<Expediente> asociar, List<Expediente> desasociar)
        throws Exception {
    indisciplina = indisciplinaDAO.persist(indisciplina);
    if (asociar != null) {
        for (Expediente expediente : asociar) {
             IndividuoInvolucradoIndisciplina iii = new
            IndividuoInvolucradoIndisciplina(
             indisciplina, expediente, null);
             individuoInvolucradoIndisciplinaDAO.persist(iii);
        }
     }
    if (desasociar != null) {
        for (Expediente expediente : desasociar) {
             IndividuoInvolucradoIndisciplina iii = new
            IndividuoInvolucradoIndisciplina(
             indisciplina, expediente, null);
```

```
                individuoInvolucradoIndisciplinaDAO.delete
            (individuoInvolucradoIndisciplinaDAO.findById(iii.getId(), true));
        }
    }
}
```

Anexo 4. Implementación del controlador *RegistrarEntregaPertenenciasController*.

```
public class RegistrarEntregaPertenenciasController extends AbstractController {
       private PertenenciasFacade pertenenciasFacade;
       private SimpleDateFormat sdFF=new SimpleDateFormat("dd/MM/yyyy");
       public void setPertenenciasFacade(PertenenciasFacade pertenenciasFacade) {
              this.pertenenciasFacade = pertenenciasFacade;
       }
       protected ModelAndView handleRequestInternal(HttpServletRequest request,
                     HttpServletResponse response) throws Exception {
     EntregaPertenencia entregaPertenencia = new EntregaPertenencia();
     Date fechaEntrega = sdFF.parse(request.getParameter("fechaEntregaString"));
     String idInd = request.getParameter("idIndividuo");
     Individuo individuo = new Individuo(idInd);
     String idVinc = request.getParameter("idVinculo");
     Vinculo vinculo = new Vinculo(idVinc);
     JSONArray pertenencias =
     JSONArray.fromString(request.getParameter("pertenencias"));
     List<Pertenencia> pertenenciasEntregadas = new ArrayList<Pertenencia>();
     for (int i = 0; i < pertenencias.length(); i++) {
              Pertenencia pertenencia = new Pertenencia();
              pertenencia.setId((String) pertenencias.getJSONObject(i).get
            ("idPertenencia"));
              pertenenciasEntregadas.add(pertenencia);
       }
       entregaPertenencia.setFechaEntrega(fechaEntrega);
       entregaPertenencia.setPertenencias(pertenenciasEntregadas);
       entregaPertenencia.setIndividuo(individuo);
       entregaPertenencia.setVinculo(vinculo);
       //registrar la entrega de pertenencias
       pertenenciasFacade.registrarEntregaPertenencias(entregaPertenencia);
       return null;

   }
}
```

Anexo 5. Implementación del controlador *SancionDisciplinariaSimpleFormController*.

```
public class SancionDisciplinariaSimpleFormController extends SimpleFormController {
     private MedidasDisciplinariasFacade medidasDisciplinariasFacade;
```

```
	private NomencladoresFacade nomencladoresFacade;
	public MedidasDisciplinariasFacade getMedidasDisciplinariasFacade() {
		return medidasDisciplinariasFacade;
	}
	public void setMedidasDisciplinariasFacade(
			MedidasDisciplinariasFacade medidasDisciplinariasFacade) {
		this.medidasDisciplinariasFacade = medidasDisciplinariasFacade;
	}
	public NomencladoresFacade getNomencladoresFacade() {
		return nomencladoresFacade;
	}
	public void setNomencladoresFacade(NomencladoresFacade
	     nomencladoresFacade) {
		this.nomencladoresFacade = nomencladoresFacade;
	}
	public SancionDisciplinariaSimpleFormController() {
		setCommandName("sancionDisciplinaria");
		setCommandClass(SancionDisciplinariaCommand.class);
		setFormView("registrarMedidas");
	}
	protected ModelAndView processFormSubmission(HttpServletRequest request,
	   HttpServletResponse response, Object command, BindException errors)
			throws Exception {
	   SancionDisciplinariaCommand sancionDisciplinariaCommand =
		(SancionDisciplinariaCommand) command;
	   List<IndividuoInvolucradoIndisciplina> indisciplinasAdicionadas =
		new ArrayList<IndividuoInvolucradoIndisciplina>();
	   List<IndividuoInvolucradoIndisciplina> indisciplinasEliminadas = new
		ArrayList<IndividuoInvolucradoIndisciplina>();
	   JSONArray jsonArrayIndisciplinasAdicionadas =
	   JSONArray.fromString(request.getParameter("indisciplinasAdicionadas"));
	   JSONArray jsonArrayIndisciplinasEliminados =
	   JSONArray.fromString(request.getParameter("indisciplinasEliminadas"));
	   SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
	   for (int i = 0; i < jsonArrayIndisciplinasAdicionadas.length(); i++) {
		String id = (String)
		jsonArrayIndisciplinasAdicionadas.getJSONObject(i).get("id");
		 IndividuoInvolucradoIndisciplina indisciplina = new
		IndividuoInvolucradoIndisciplina(new Indisciplina(id), new
		Expediente(sancionDisciplinariaCommand.getIdExpediente()), null);
		 indisciplinasAdicionadas.add(indisciplina);
	    }
   for (int i = 0; i < jsonArrayIndisciplinasEliminados.length(); i++) {          String
id = (String)
	   jsonArrayIndisciplinasEliminados.getJSONObject(i).get("id");
	    IndividuoInvolucradoIndisciplina indisciplina = new
	   IndividuoInvolucradoIndisciplina(new Indisciplina(id), new
	   Expediente(sancionDisciplinariaCommand.getIdExpediente()), null);
	    indisciplinasEliminadas.add(indisciplina);
    }
   List<MedidaDisciplinaria> medidaDisciplinariaAdicionadas = new
   ArrayList<MedidaDisciplinaria>();
   List<MedidaDisciplinaria> medidaDisciplinariaEliminadas = new
   ArrayList<MedidaDisciplinaria>();
```

```
 List<MedidaDisciplinaria> medidaDisciplinariaModificadas = new
 ArrayList<MedidaDisciplinaria>();
 JSONArray jsonArrayMedidaDisciplinariaAdicionadas =
 JSONArray.fromString(request.getParameter("medidasDisciplinariasAdic
 ionadas"));
 JSONArray jsonArrayMedidaDisciplinariaEliminados =
 JSONArray.fromString(request.getParameter("medidasDisciplinarias
 Eliminadas"));
 JSONArray jsonArrayMedidaDisciplinariaModificados =
 JSONArray.fromString(request.getParameter("medidasDisciplinarias
 Modificadas"));
for (int i = 0; i < jsonArrayMedidaDisciplinariaAdicionadas.length(); i++) {
   MedidaDisciplinaria medidaDisciplinaria = new MedidaDisciplinaria();
   String idNombreMD = (String)
   jsonArrayMedidaDisciplinariaAdicionadas.getJSONObject(i).get("idNombreMD");
   String fechaM = (String)
   jsonArrayMedidaDisciplinariaAdicionadas.getJSONObject(i).get("fechaM");
   String fechaInicio = (String)
   jsonArrayMedidaDisciplinariaAdicionadas.getJSONObject(i).get("fecha
   Inicio");
   String fechaFin = (String)
   jsonArrayMedidaDisciplinariaAdicionadas.getJSONObject(i).get("fechaFin");
   String rango = (String)
   jsonArrayMedidaDisciplinariaAdicionadas.getJSONObject(i).get("rango");
   String descripcion = (String)
   jsonArrayMedidaDisciplinariaAdicionadas.getJSONObject(i).get
   ("descripcion");
     if (!idNombreMD.equals("-1"))
         medidaDisciplinaria.setNombre(new NomMedidaDisciplinaria(idNombreMD));
     if (rango.equals("true")) {

         if (fechaInicio != null || !fechaInicio.equals(""))
              medidaDisciplinaria.setFechaInicio(sdf.parse(fechaInicio));
         if (fechaFin != null || !fechaFin.equals(""))
              medidaDisciplinaria.setFechaFin(sdf.parse(fechaFin));

     } else {
           if (fechaM != null || !fechaM.equals(""))
               medidaDisciplinaria.setFecha(sdf.parse(fechaM));
     }
     medidaDisciplinaria.setDescripcion(descripcion);
     medidaDisciplinariaAdicionadas.add(medidaDisciplinaria);
}

for (int i = 0; i < jsonArrayMedidaDisciplinariaEliminados.length(); i++) {
     MedidaDisciplinaria medidaDisciplinaria = new MedidaDisciplinaria();
     String id = (String)
    jsonArrayMedidaDisciplinariaEliminados.getJSONObject(i).get("id");
     medidaDisciplinaria.setId(id);
     medidaDisciplinariaEliminadas.add(medidaDisciplinaria);
 }
for (int i = 0; i < jsonArrayMedidaDisciplinariaModificados.length(); i++) {
     MedidaDisciplinaria medidaDisciplinaria = new MedidaDisciplinaria();
  String id = (String)
```

```
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("id");
    String idNombreMD = (String)
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("idNombreMD");
    String fechaM = (String)
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("fechaM");
    String fechaInicio = (String)
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("fechaInicio");
    String fechaFin = (String)
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("fechaFin");
    String rango = (String)
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("rango");
    String descripcion = (String)
    jsonArrayMedidaDisciplinariaModificados.getJSONObject(i).get("descripcion");
       medidaDisciplinaria.setId(id);
       if (!idNombreMD.equals("-1"))
           medidaDisciplinaria.setNombre(new NomMedidaDisciplinaria(idNombreMD));
       if (rango.equals("true")) {
              if (fechaInicio != null || !fechaInicio.equals(""))
                  medidaDisciplinaria.setFechaInicio(sdf.parse(fechaInicio));
              if (fechaFin != null || !fechaFin.equals(""))
                  medidaDisciplinaria.setFechaFin(sdf.parse(fechaFin));
       } else {
              if (fechaM != null || !fechaM.equals(""))
                  medidaDisciplinaria.setFecha(sdf.parse(fechaM));
       }
       medidaDisciplinaria.setDescripcion(descripcion);
       medidaDisciplinariaModificadas.add(medidaDisciplinaria);
   }
   List<MedidaDisciplinaria> medidasDesciplinariasAddAndEdit = new
   ArrayList<MedidaDisciplinaria>();
   medidasDesciplinariasAddAndEdit.addAll(medidaDisciplinariaAdicionadas);
   medidasDesciplinariasAddAndEdit.addAll(medidaDisciplinariaModificadas);
    SancionDisciplinaria sancionDisciplinaria =
    sancionDisciplinariaCommand.getSancionDisciplinaria();
    medidasDisciplinariasFacade.registrarSancionDisciplinaria
    (sancionDisciplinaria, indisciplinasAdicionadas, indisciplinasEliminadas,
    medidasDesciplinariasAddAndEdit, medidaDisciplinariaEliminadas);
    return null;
}
protected Map<String, Object> referenceData(HttpServletRequest request) throws
        Exception {
         Map<String, Object> model = new HashMap<String, Object>();
         model.put("nombres",
        nomencladoresFacade.listarNomMedidaDisciplinaria());
         String idSancion = request.getParameter("idSancion");
         Personal personal = this.obtenerFuncionario(idSancion);
         model.put("nombreFuncionario", personal != null ?
        personal.getNombreCompleto() : "");
        return model;
}
protected Object formBackingObject(HttpServletRequest request)
       throws Exception {
   if (!isFormSubmission(request)) {
       SancionDisciplinariaCommand command = new SancionDisciplinariaCommand();
```

```
        String idSancion = request.getParameter("idSancion");
        String idExpediente = request.getParameter("idExpediente");
        command.setIdExpediente(idExpediente);
        Personal personal = this.obtenerFuncionario(idSancion);
        command.setIdFuncionario(personal != null ? personal.getId() : "");
       if (idSancion != null && !idSancion.equals("")) {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            SancionDisciplinaria sancionDisciplinaria =
           medidasDisciplinariasFacade.buscarSancionDisciplinaria(new
           SancionDisciplinaria(idSancion));
            command.setId(sancionDisciplinaria.getId());
            command.setFecha(sdf.format(sancionDisciplinaria.getFecha()));
            if (sancionDisciplinaria.getObservaciones() != null)
                command.setObservaciones(sancionDisciplinaria.getObservaciones());
             }
             return command;

        } else
             return super.formBackingObject(request);
}
public Personal obtenerFuncionario(String idSancion) throws Exception {
        if (idSancion != null && !idSancion.equals(""))
            return medidasDisciplinariasFacade.buscarSancionDisciplinaria(new
           SancionDisciplinaria(idSancion)).getFuncionario();
            return medidasDisciplinariasFacade.obtenerFuncionario();
        }
}
```

Anexo 6. Implementación del controlador *RegistrarIndisciplinaController*.

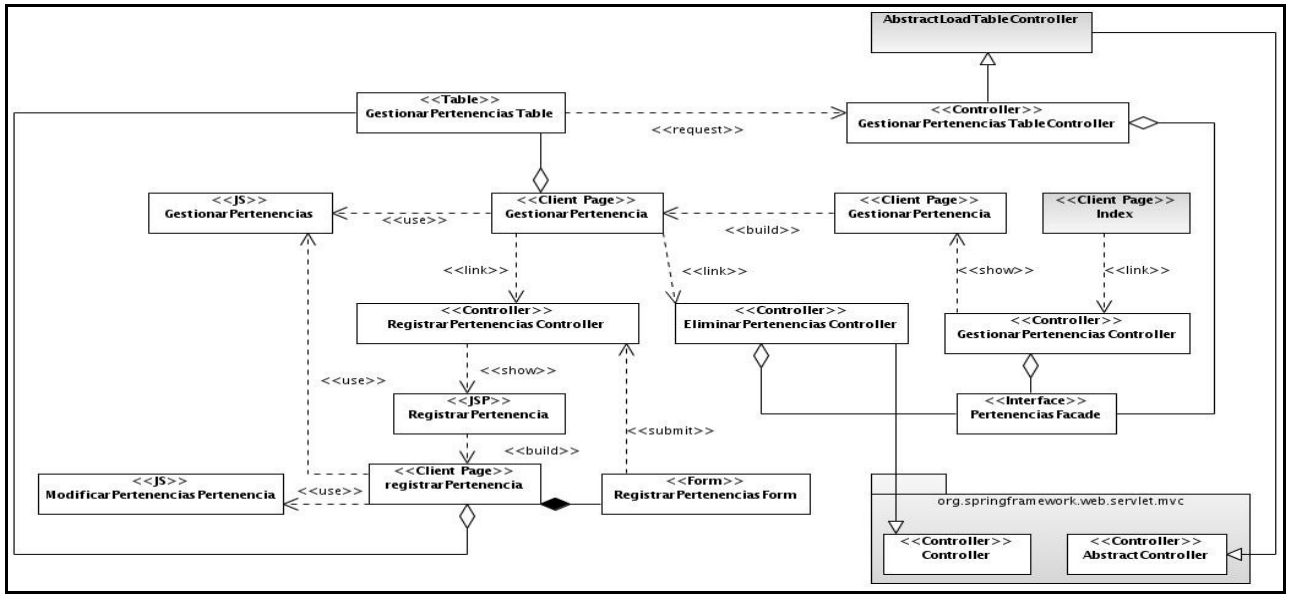
```
public class RegistrarIndisciplinaController extends SimpleFormController {
        private MedidasDisciplinariasFacade medidasDisciplinariasFacade;
        private NomencladoresFacade nomencladoresFacade;
        private SimpleDateFormat horaFormat = new SimpleDateFormat("HH:mm");
        private SimpleDateFormat fechaFormat = new SimpleDateFormat("dd/MM/yyyy");
        public RegistrarIndisciplinaController() {
               super();
               setCommandClass(RegistroIndisciplinaCommand.class);
        }
    public void setMedidasDisciplinariasFacade(
    MedidasDisciplinariasFacade medidasDisciplinariasFacade) {
               this.medidasDisciplinariasFacade = medidasDisciplinariasFacade;
        }
    public void setNomencladoresFacade(NomencladoresFacadenomencladoresFacade) {
               this.nomencladoresFacade = nomencladoresFacade;
        }
    protected ModelAndView onSubmit(HttpServletRequest request,
    HttpServletResponse response, Object commandRef,
    BindException errors) throws Exception {
   RegistroIndisciplinaCommand command = (RegistroIndisciplinaCommand)
   commandRef;
               try {
```

```
                Indisciplina indisciplina = command.getIndisciplina();
                if (!StringUtils.hasText(indisciplina.getId()))
                        indisciplina.setId(null);
              else
                 if (medidasDisciplinariasFacade.estaSancionada(indisciplina)) {
                      response.getWriter().write("{error:1}");
                      return null;
                   }
            medidasDisciplinariasFacade.registrarIndisciplina(indisciplina,
            command.getIndividuosAsociar(), command.getIndividuosDesasociar());
      } catch (Exception e) {
             response.getWriter().write("{error:1}");
             e.printStackTrace();
             return null;}
       response.getWriter().write("{error:0}");
       return null;
}
protected Map<String, Object> referenceData(HttpServletRequest request,
             Object command, Errors errors) throws Exception {
             Map<String, Object> data = new HashMap<String, Object>();
             data.put("categoriasIndisciplina", nomencladoresFacade
             .listarNomCategoriaIndisciplina());
             data.put("clasificacionesIndisciplina", nomencladoresFacade
             .listarNomClasificacionIndisciplina());
             Date ahora = new Date();
             data.put("fechaActual", fechaFormat.format(ahora));
             data.put("horaActual", horaFormat.format(ahora));
             return data;
}
protected Object formBackingObject(HttpServletRequest request)
throws Exception {
   if (!isFormSubmission(request)) {
        String id = request.getParameter("id");
        RegistroIndisciplinaCommand command;
       if (!StringUtils.hasText(id)) {
       // registro de una nueva indisciplina
          command = new RegistroIndisciplinaCommand(null);
          command.getIndisciplina().setFecha(new Date());
       } else {
             Indisciplina indisciplina = medidasDisciplinariasFacade
            .buscarIndisciplina(new Indisciplina(id));
              command = new RegistroIndisciplinaCommand(indisciplina);
             }
             return command;

             } else
                    return super.formBackingObject(request);}
protected void initBinder(HttpServletRequest request,
ServletRequestDataBinder binder) throws Exception {

       binder.registerCustomEditor(Date.class, "fecha", new
      CustomDateEditor(fechaFormat, true));
       binder.registerCustomEditor(Date.class, "hora", new
      CustomDateEditor(horaFormat, true));
```

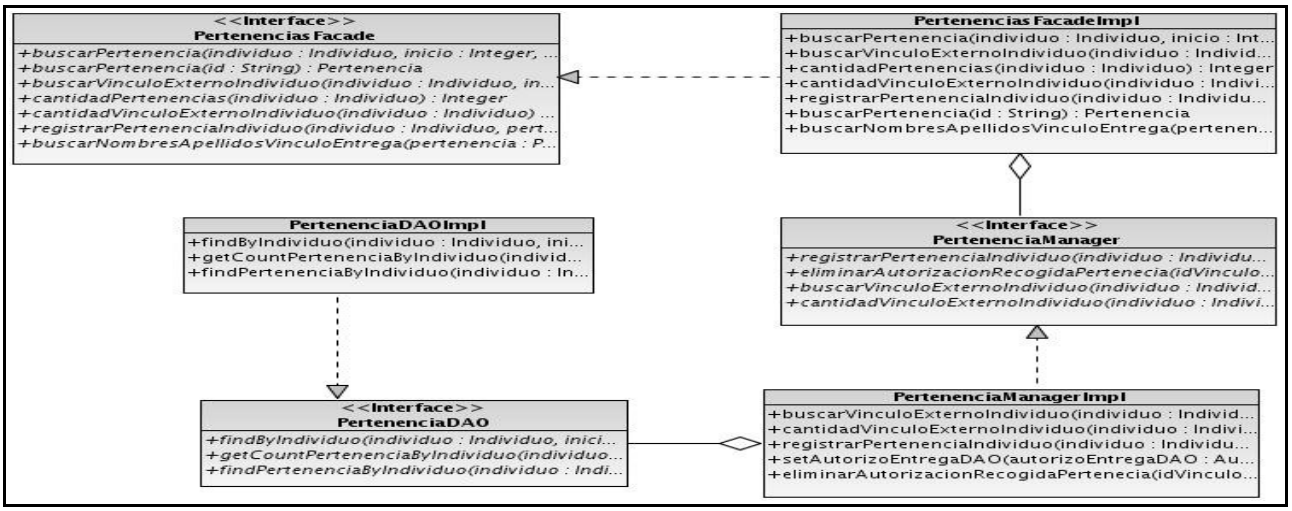
```
        binder.registerCustomEditor(List.class, new ExpedientesEditor());
}
private class ExpedientesEditor extends PropertyEditorSupport {

  public void setAsText(String text) throws IllegalArgumentException {
        if (JSONUtils.mayBeJSON(text) && text.startsWith("[")) {
           JSONArray array = JSONArray.fromString(text);
           List<Expediente> lista = new ArrayList<Expediente>();
        for (Integer i = 0; i < array.length(); i++) {
            JSONObject object = (JSONObject) array.get(i);
            lista.add(new Expediente(object.getString("idExpediente")));
               }
               setValue(lista);
           }
       }
    }
}
```

Anexo 7. Diseño de las funcionalidades del módulo Pertenencias.

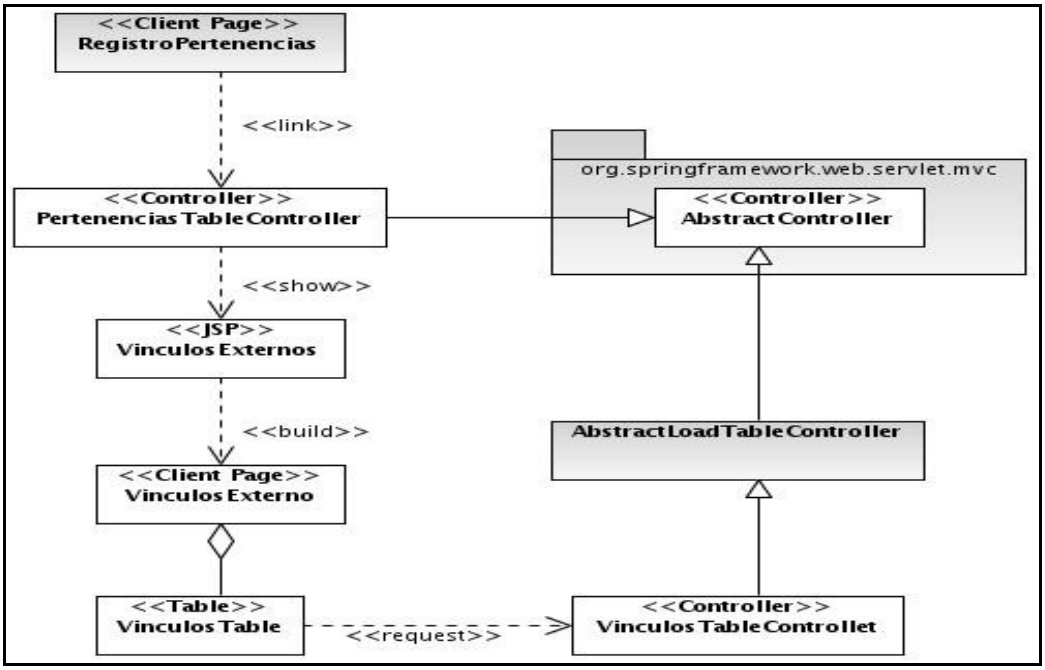
Capa de presentación de las funcionalidades Registrar, Modificar y Eliminar pertenencias a un individuo.



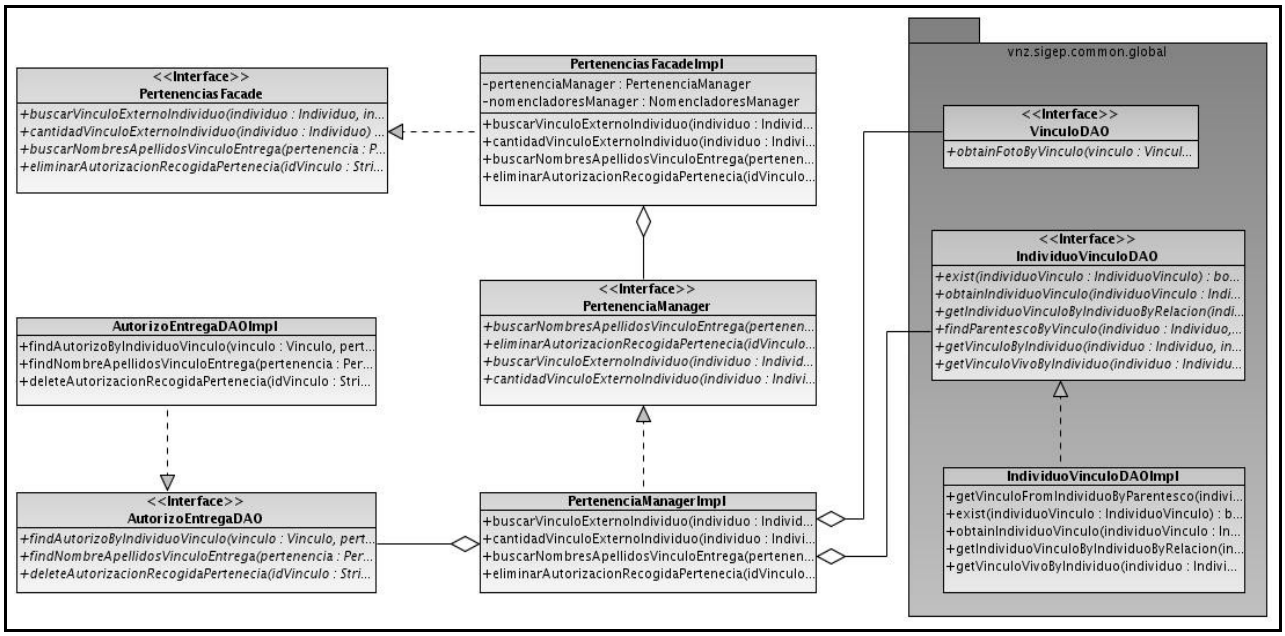
Capa de negocio y acceso a datos de las funcionalidades Registrar, Modificar y Eliminar pertenencias a un individuo.



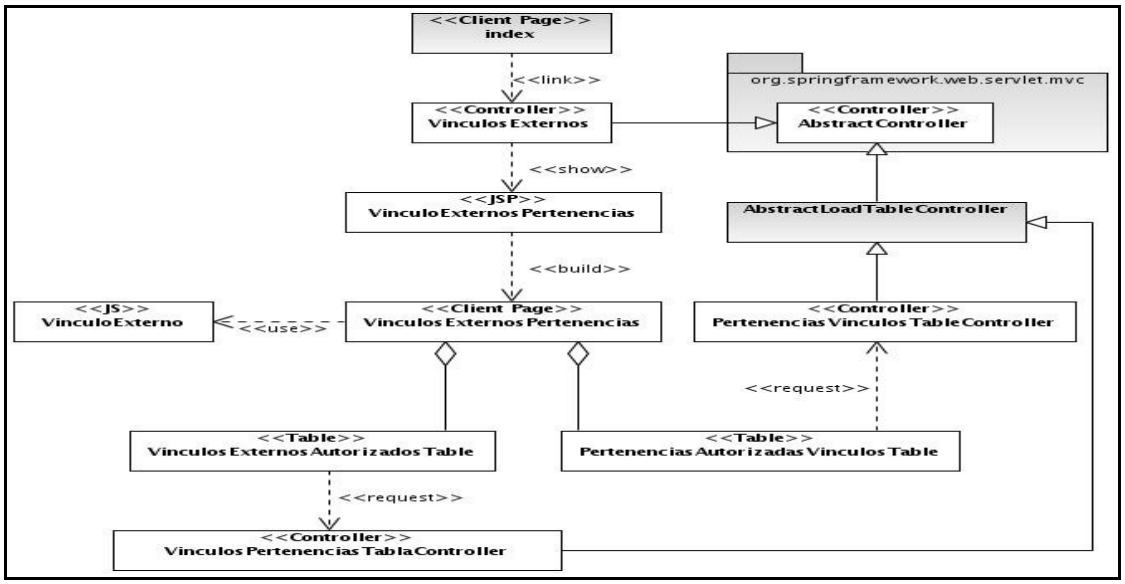
Capa de presentación de la funcionalidad Autorizar entrega de pertenencias a un vinculo externo.



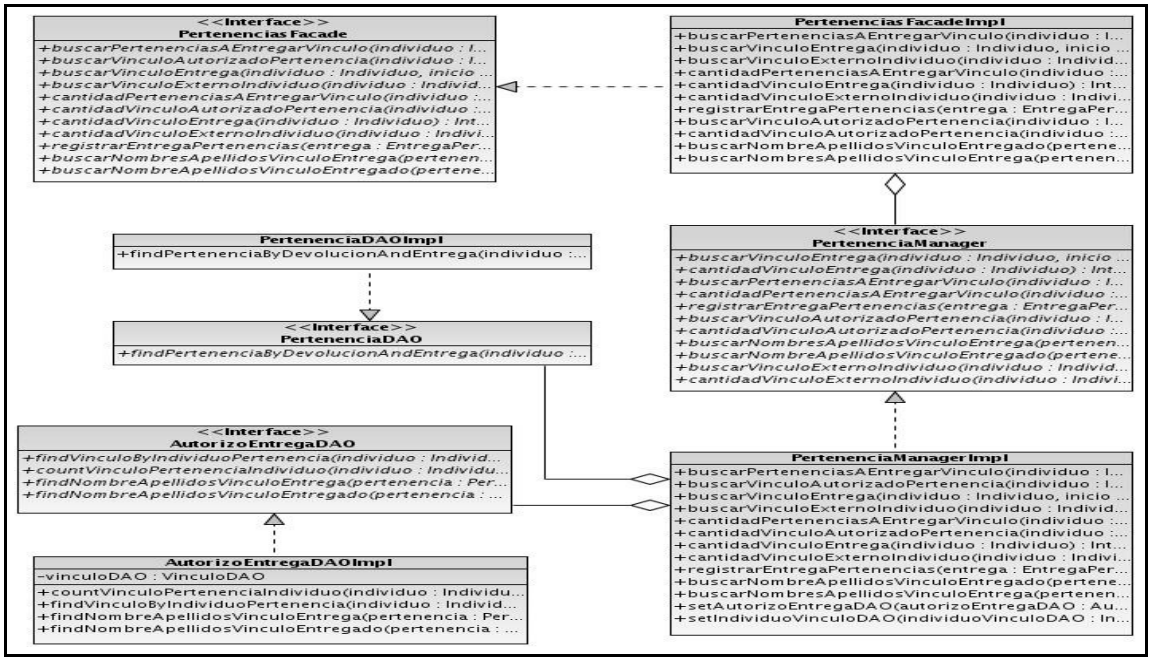
Capa de negocio y acceso a datos de la funcionalidad Autorizar entrega de pertenencias a un vinculo externo.



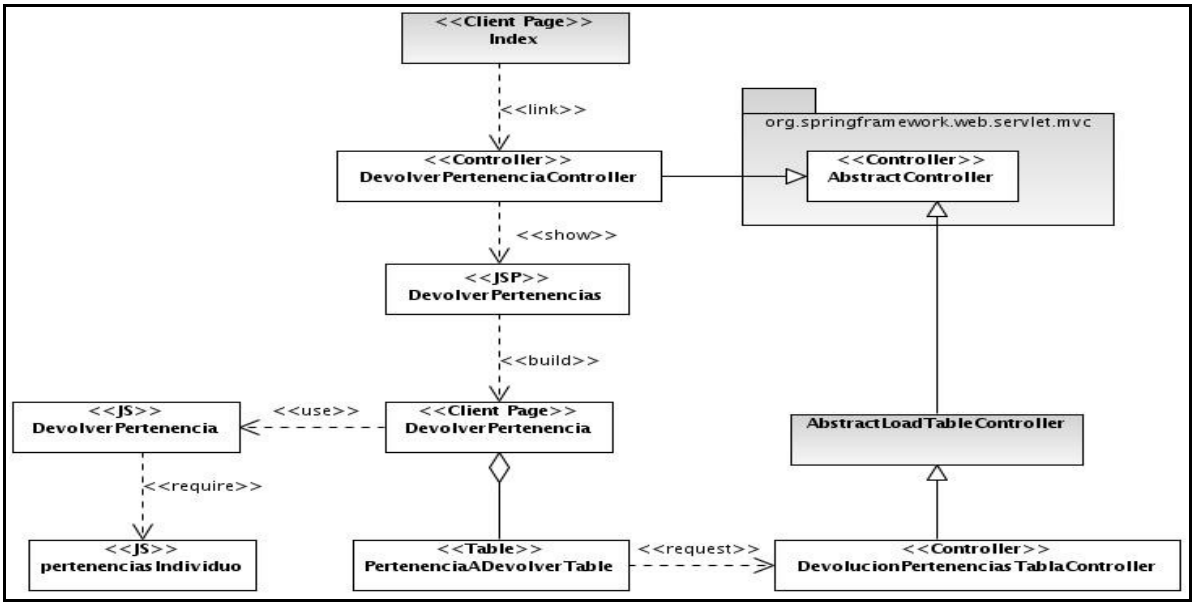
Capa de presentación de la funcionalidad Entregar pertenencias a un vinculo externo.



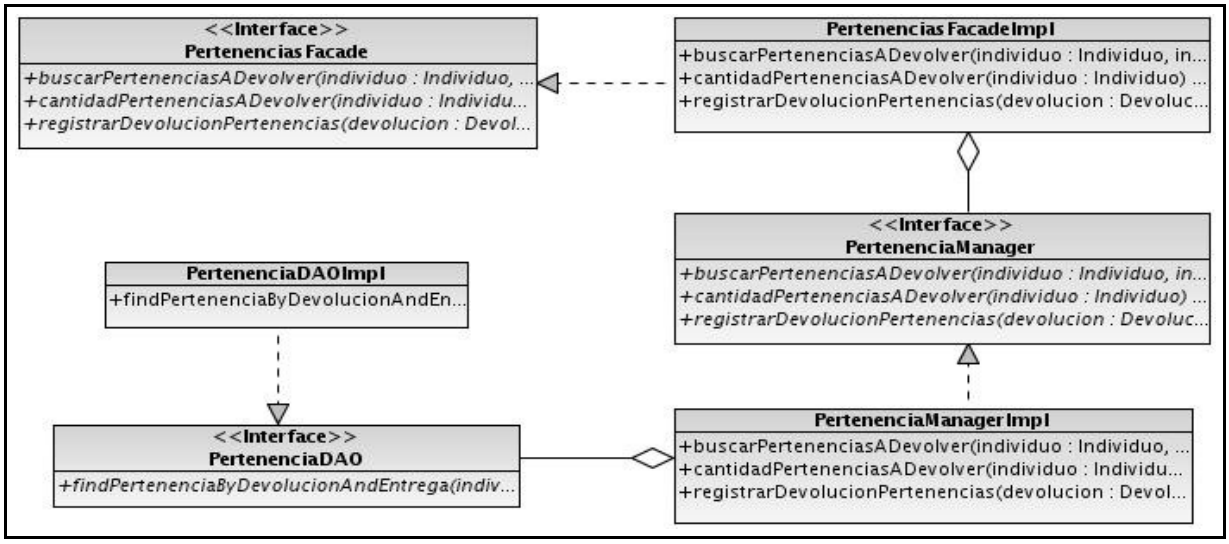
Capa de negocio y acceso a datos de la funcionalidad Entregar pertenencias a un vinculo externo.



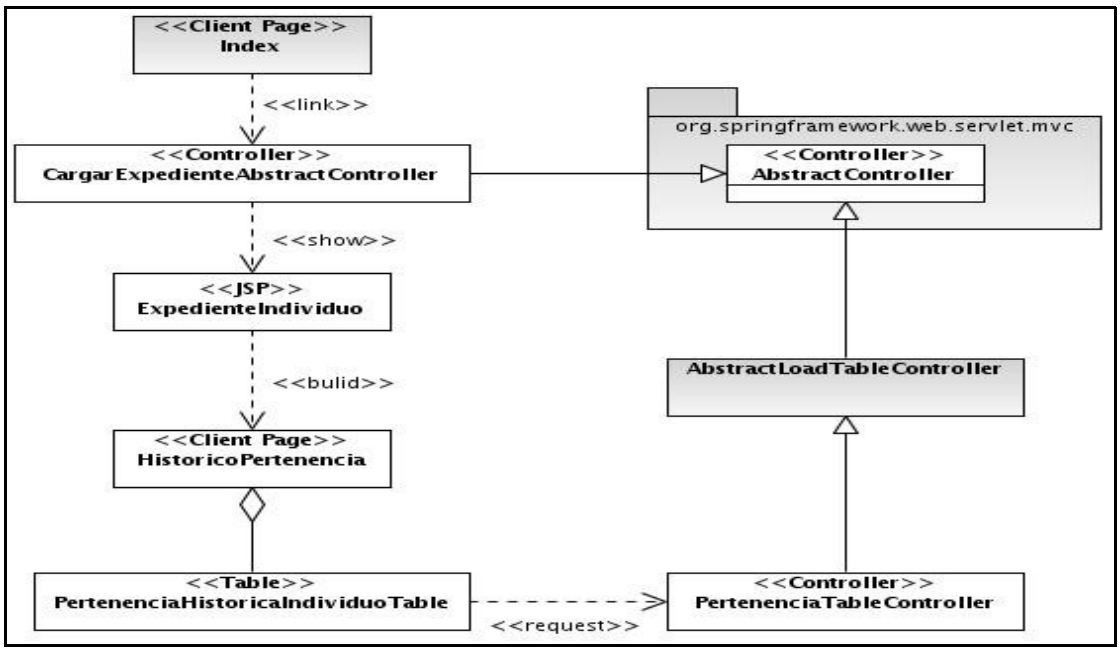
Capa de presentación de la funcionalidad Devolver pertenencias a un individuo.



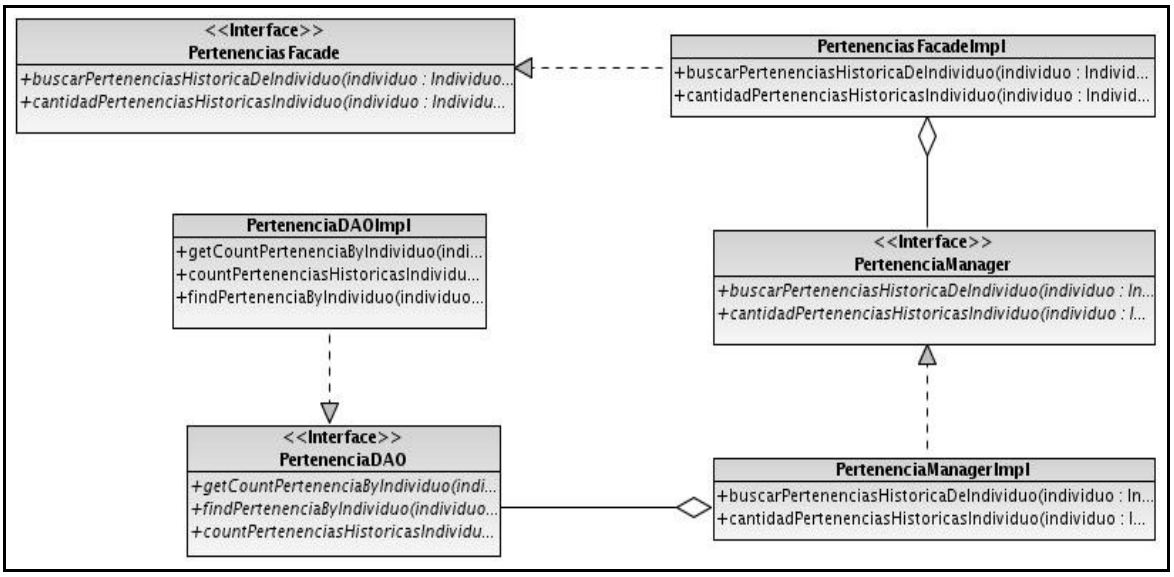
Capa de negocio y acceso a datos de la funcionalidad Devolver pertenencias a un individuo.



Capa de presentación de la funcionalidad Listar pertenencias de un individuo (en el expediente).

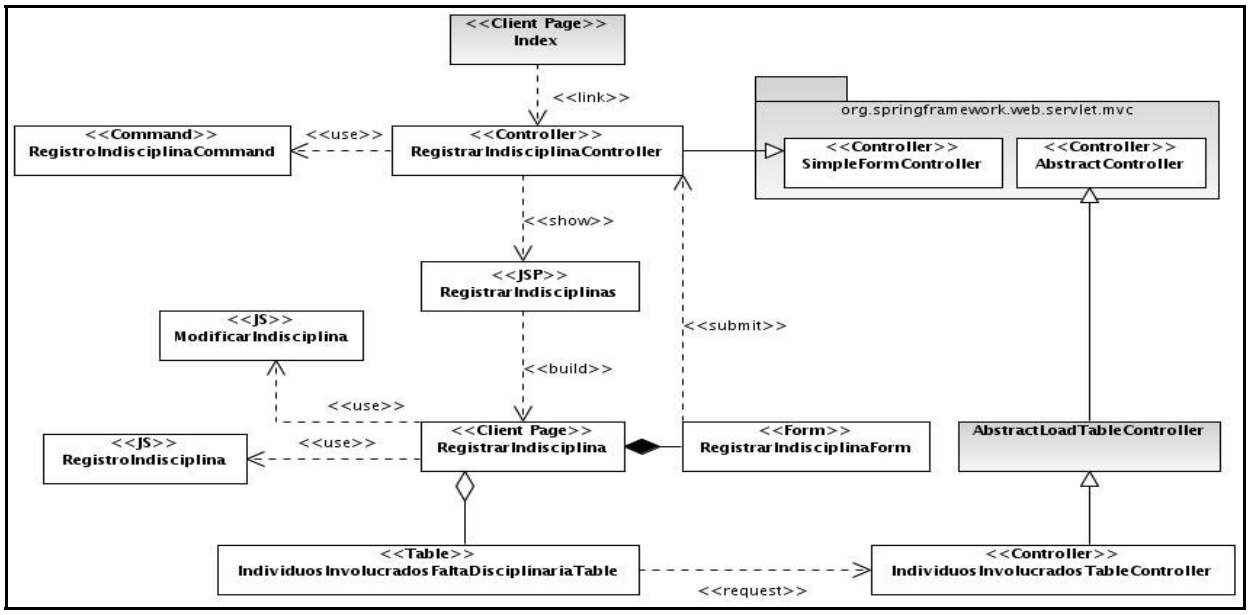


Capa de negocio y acceso a datos de la funcionalidad Listar pertenencias de un individuo (en el expediente).

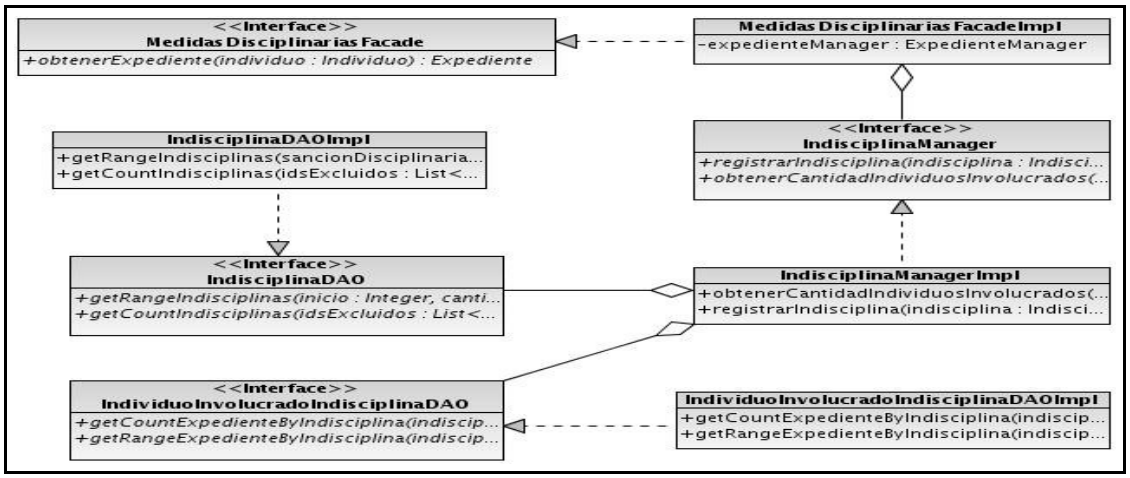


Anexo 8. Diseño de las funcionalidades del módulo Medidas Disciplinarias.

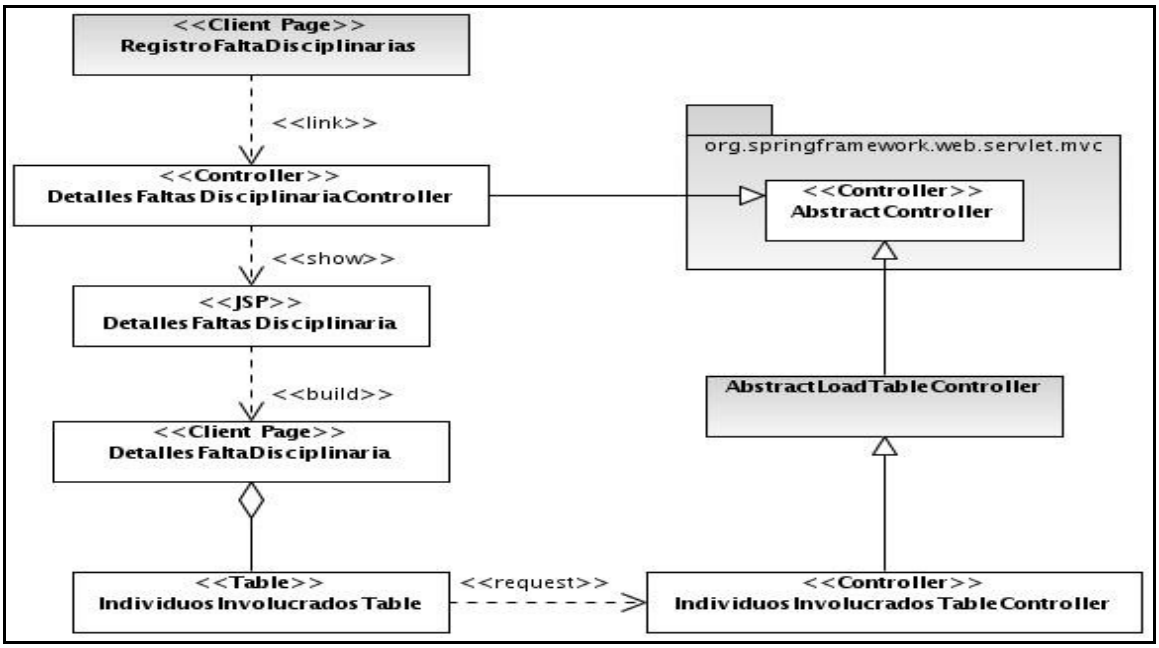
Capa de presentación de la funcionalidad Registrar indisciplina.



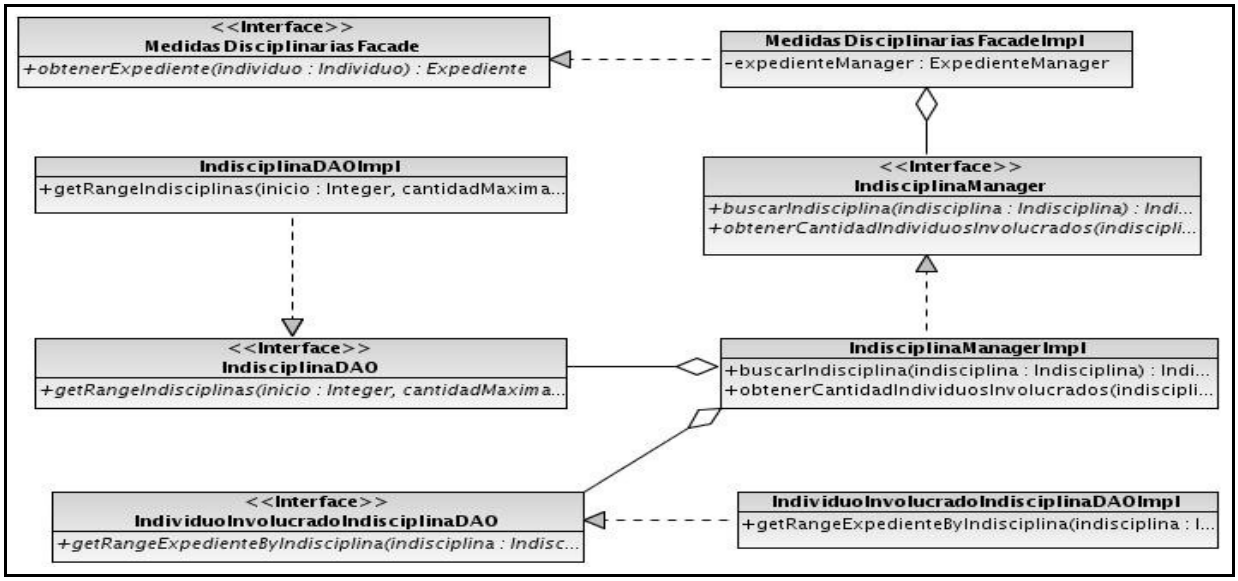
Capa de negocio y acceso a datos de la funcionalidad Registrar indisciplina.



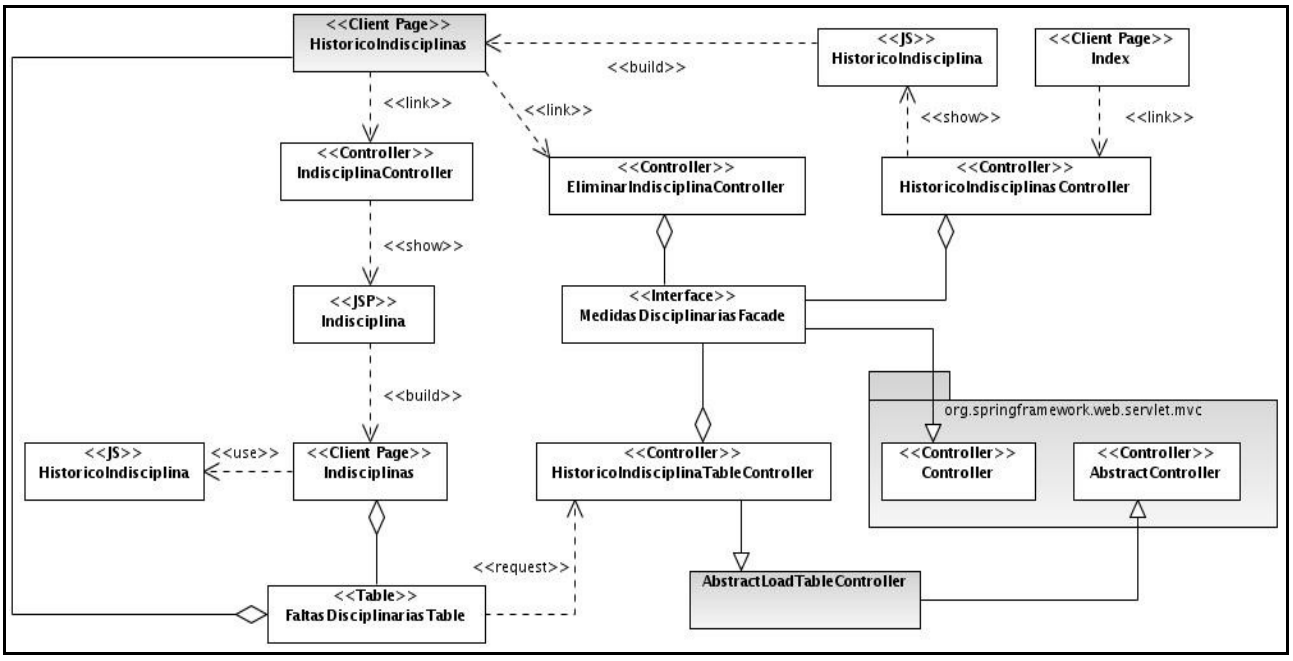
Capa de presentación de la funcionalidad Consultar detalles de la indisciplina.



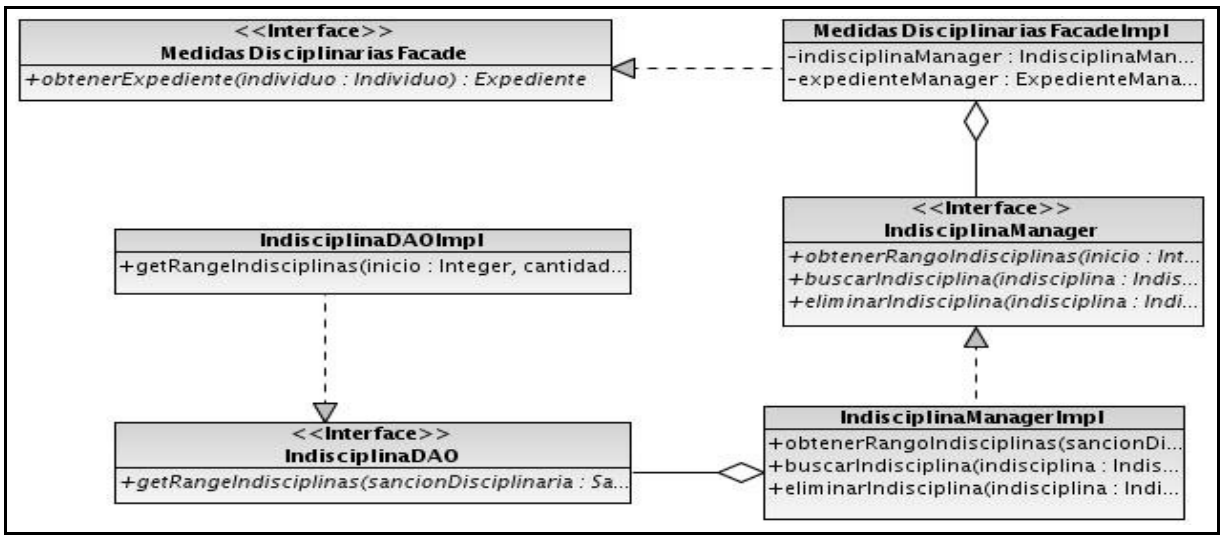
Capa de negocio y acceso a datos de la funcionalidad Consultar detalles de la indisciplina.



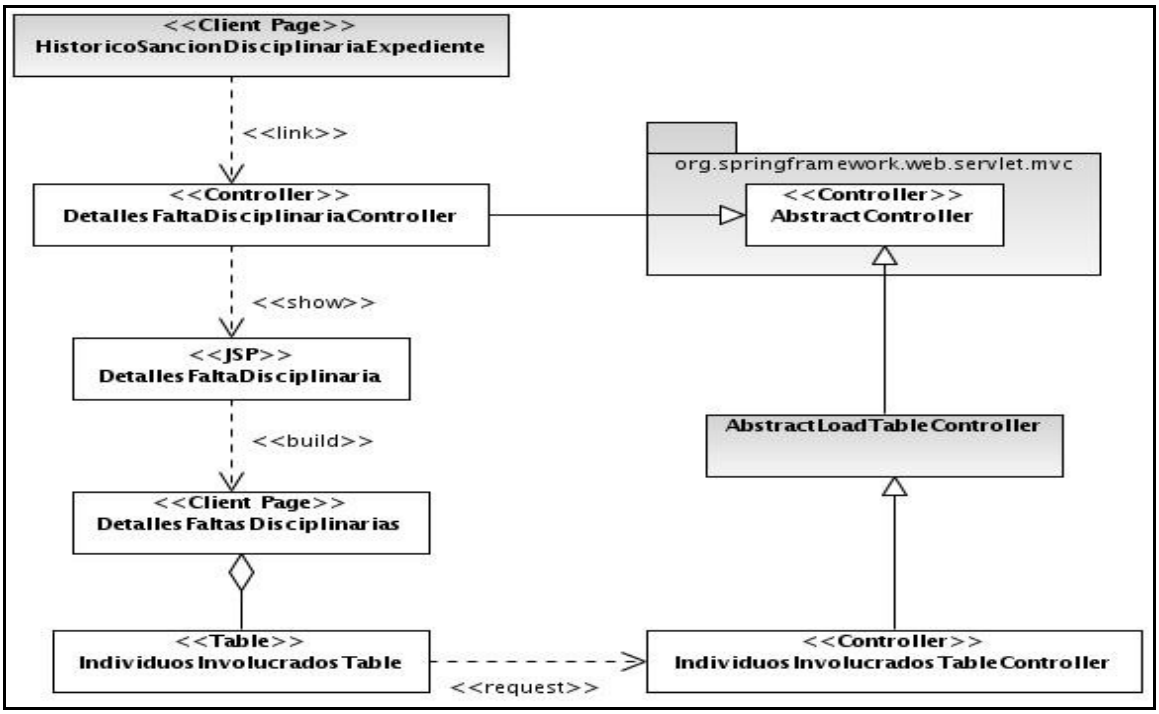
Capa de presentación de la funcionalidad Histórico de indisciplinas cometidas en el penal.



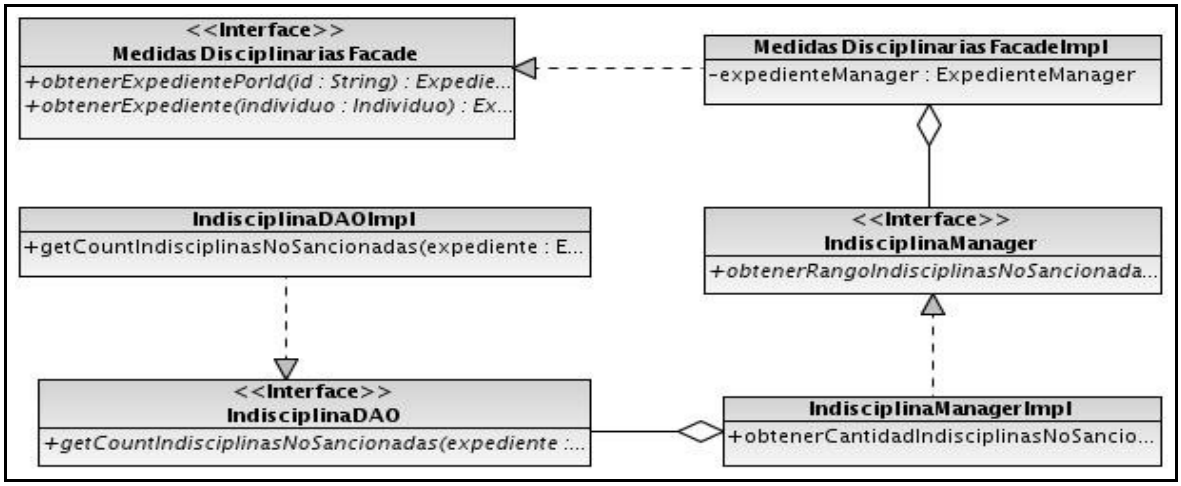
Capa de negocio y acceso a datos de la funcionalidad Histórico de indisciplinas cometidas en el penal.



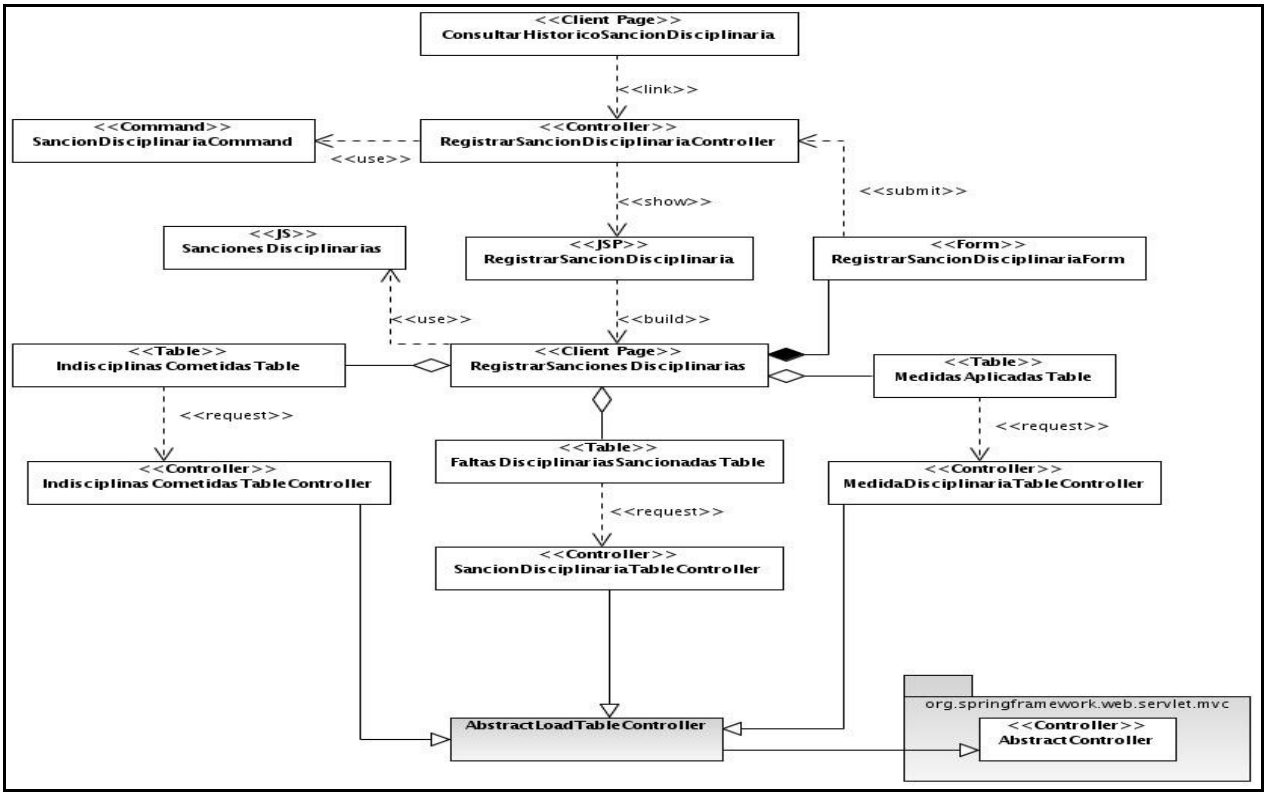
Capa de presentación de la funcionalidad Mostrar indisciplinas no sancionadas (en el expediente).



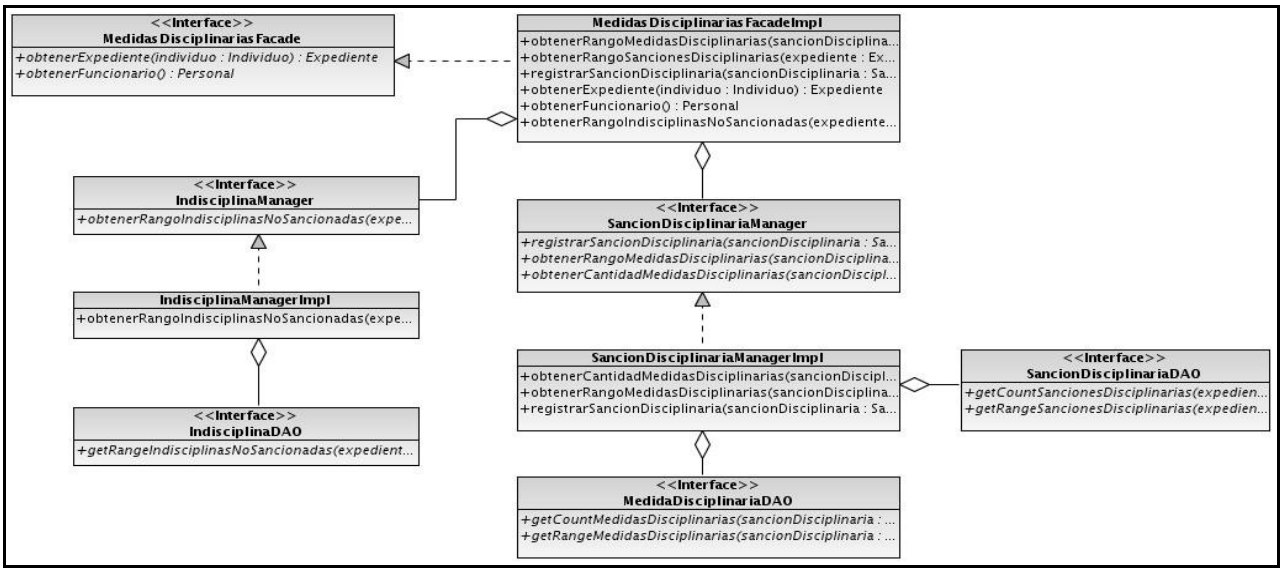
Capa de negocio y acceso a datos de la funcionalidad Mostrar indisciplinas no sancionadas (en el expediente).



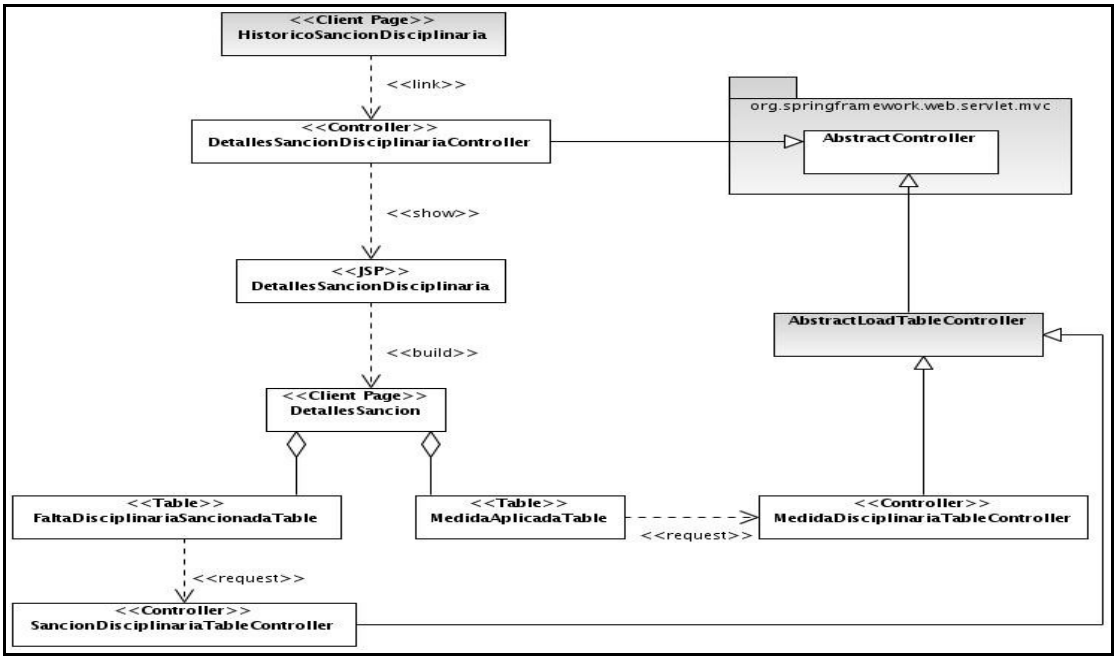
Capa de presentación de la funcionalidad Registrar sanción disciplinaria.



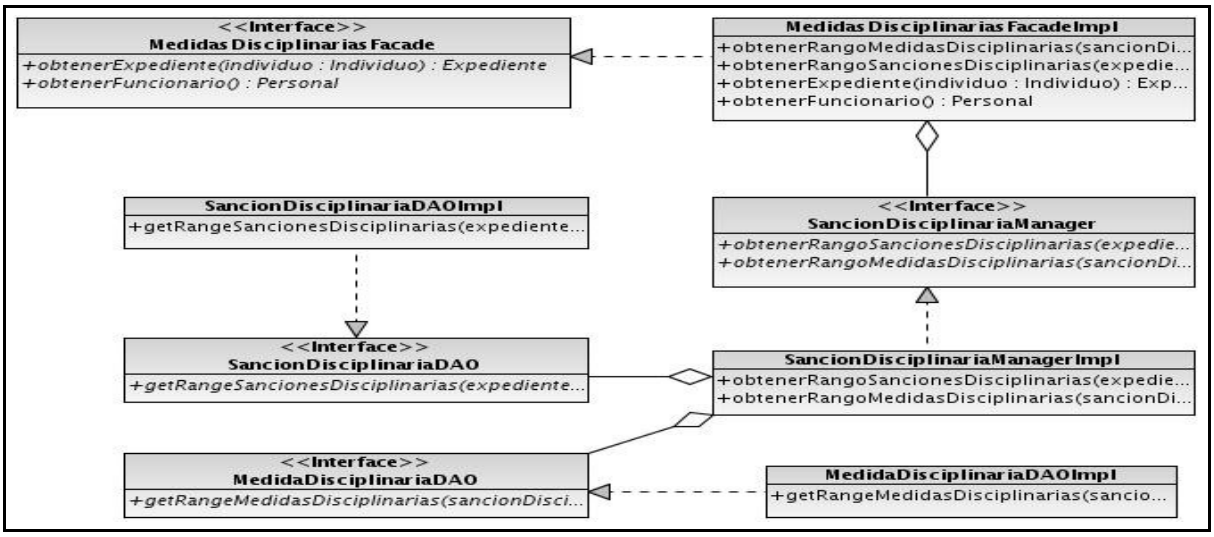
Capa de negocio y acceso a datos de la funcionalidad Registrar sanción disciplinaria.



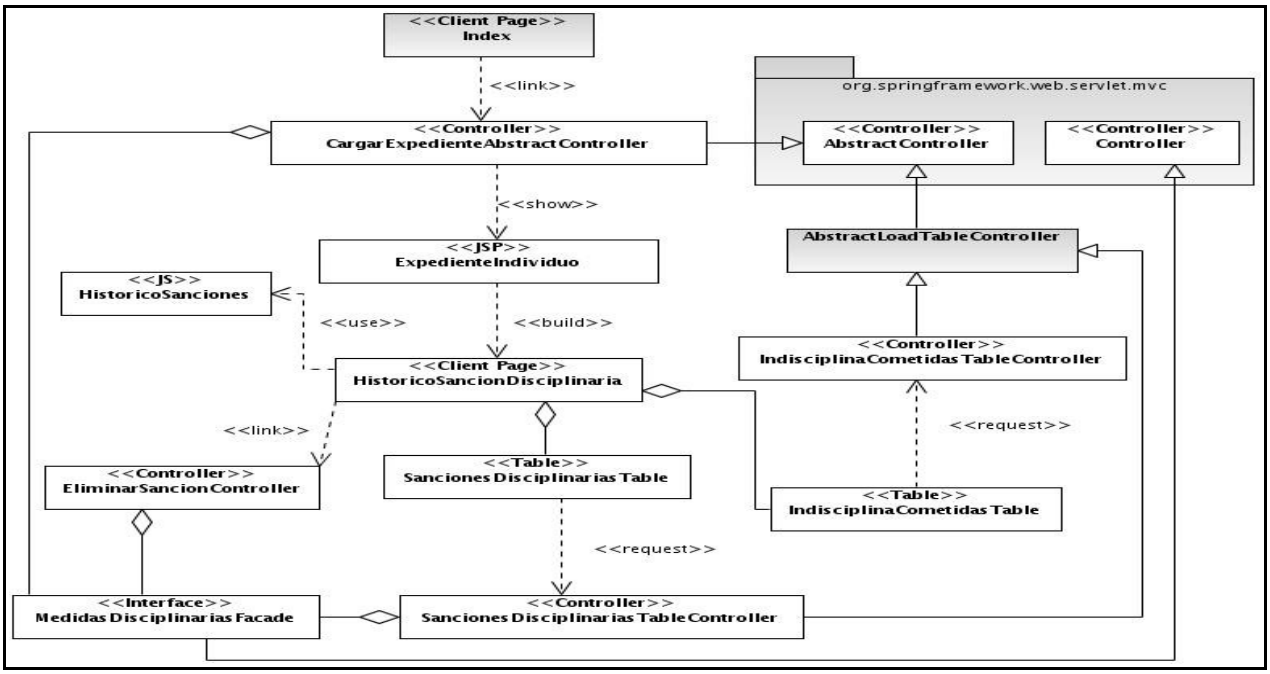
Capa de presentación de la funcionalidad Consultar detalles de sanción disciplinaria.



Capa de negocio y acceso a datos de la funcionalidad Consultar detalles de sanción disciplinaria.



Capa de presentación de la funcionalidad Consultar histórico de sanciones disciplinarias del individuo (en el expediente).



GLOSARIO

A

API (Application Programming Interface): Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

B

Beans: Se definen como "componentes de software reutilizables que se puedan manipular visualmente en una herramienta de construcción". Se usan para encapsular varios objetos en un único objeto para hacer uso de un sólo objeto en lugar de varios más simples.

I

Inversión de control (IoC): Conjunto de técnicas y patrones de diseño de software que invierte el control del flujo de un sistema. El control es invertido en comparación con el modelo tradicional de interacción expresado en una serie de llamadas consecutivas a procedimientos.

J

Javadoc: Utilidad de Sun Microsystems para generar APIs en formato HTML de un documento de código fuente Java. Es el estándar para documentar clases Java.

JAR: (Java Archive Tipo de archivo): Permite ejecutar aplicaciones escritas en lenguaje Java.

N

Nomenclador: Tabla de la base de datos que contiene datos constantes, ej. nacionalidades, nombre de colores, días de la semana.

O

ORM (Object Relacional Mapping): Persistencia automática y transparente de objetos de una aplicación en una base de datos relacional utilizando meta datos que describen la correspondencia entre el objeto y las tablas de la base de datos.