Universidad de las Ciencias Informáticas

Facultad 2

Título: Desarrollo del módulo Actividades Educativas perteneciente al Sistema Informativo de la Dirección de Establecimientos Penitenciarios.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Autor: René Díaz Roché

Tutor: Ing. Roberto Granda Ruiz

Co-tutor: Ing. Anabel Betancourt de los Santos

La Habana, 2012

"Año 53 de la Revolución"

“En la tierra hace falta personas que trabajen más y critiquen menos, que construyan más y destruyan menos, que prometan menos y resuelvan más, que digan mejor hoy que mañana."

Declaración de Autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2012.

René Díaz Roché	Ing. Roberto Granda Ruiz
_____	_____
Firma del Autor	Firma del Tutor

Tutor: Ing. Roberto Granda Ruiz
Correo Electrónico: rgranda@uci.cu

Co-tutora: Ing Anabel Betancourt de los Santos

Correo Electrónico: adelossantos@uci.cu

Autor: René Díaz Roché

Correo Electrónico: rroche@estudiantes.uci.cu

Agradecimientos

A mis padres, los máximos responsables de ser la persona que soy. Por guiarme por el camino correcto. Por su sacrificio y entrega a sus hijos, por quererme tanto a pesar de los dolores de cabeza que les di. Los quiero.

A mi novia (Lin), por aguantar durante estos años que no pudimos estar tan cerca, por ser mi apoyo en los momentos difíciles y la causa de alegría en innumerables ocasiones. Mi confesora, amiga y amante. Razón fundamental de la culminación de estos cinco años de carrera. Te amo.

A mi familia, por haberme brindado su apoyo en todo momento.

Al Robe, mi tutor, gran responsable de la realización de este trabajo, por luchar junto a nosotros y brindarnos su ayuda incondicional en todo momento.

A Ernesto, Dariel, Bernardo y el chino. Hermanos de estudio, fiestas, alegrías y tristezas, por haberme permitido ser amigo de personas tan especiales. En mí siempre tendrán un hermano.

A Zulay, Yasel y Elian. Compañeros y amigos durante estos cinco años de luchas y alegrías.

A Teresa, la madre del chino, por ser nuestra segunda madre en estos años alejados de nuestra familia. Por sus consejos, su preocupación y cariño, parte importante durante estos cinco años.

A Juanky, Apa, el Lea, Leonel, Damián, Yudi, Yudith y los demás integrantes del grupo. Creo que lo único reprochable de estos cinco años fue el no habernos conocido antes.

A las personas del proyecto que de alguna forma u otra colaboraron con la realización de este trabajo.

A mis padres, mi hermano y mi futura esposa, las personas más importantes en mi vida.

Aprovechando el desarrollo de las tecnologías de la información y las comunicaciones que se está llevando a cabo en el país, la Dirección de Establecimientos Penitenciarios (DEP), en conjunto con la Universidad de las Ciencias Informáticas, deciden crear el proyecto Prisiones Cuba.

Este proyecto tiene como objetivo la creación del Sistema Informativo de la Dirección de Establecimientos Penitenciario (SIDEP), el cual pretende integrar los tres sistemas que existen en la DEP: SACORE; SACDEP[1] y SAIDEP[2], y adicionar un conjunto de funcionalidades necesarias para el correcto funcionamiento de los procesos en los centros penitenciarios.

El presente trabajo tiene como objetivo el desarrollo del módulo Actividades Educativas perteneciente al SIDEP. Teniendo como punto de partida el análisis de los requerimientos de software establecidos de acuerdo con el cliente y haciendo uso de las tecnologías y herramientas definidas en la arquitectura del proyecto.

La solución final de este módulo tiene como propósito registrar la información referente al desarrollo de las actividades educativas en el sistema penitenciario. Permitirá realizar la planificación de estas actividades, así como el registro de su ejecución. Además del registro de otros procesos vinculados a estas, como: el registro de los convenios educativos y la gestión de los grupos de las unidades y los colectivos. Esto permitirá a los funcionarios de los centros penitenciarios contar con información fidedigna para el control de las actividades educativas.

Palabras clave: Actividades Educativas, herramientas, software, tecnologías.

[1] Sistema Automatizado de Capacidades de la Dirección de Establecimientos Penitenciarios

[2] Sistema de Automatización de Incidencias de la Dirección de Establecimientos Penitenciarios

Índice de Figuras

INTRODUCCIÓN

Al triunfar la Revolución en el año 1959 se hereda un sistema penitenciario caracterizado por la corrupción, la discriminación racial, la promiscuidad, los abusos y los maltratos a los que eran sometidos los reclusos. El nuevo gobierno revolucionario toma una serie de medidas dirigidas a crear nuevas formas de enfrentar los delitos de acuerdo a los perjuicios que estos producían y construir un Sistema Penitenciario humano, sustentado en el respeto y el control riguroso de la aplicación de leyes, reglamentos y políticas que se inspiran en la máxima de reeducar y rehabilitar a cada persona recluida para su reinserción social. Durante los 53 años de Revolución el Sistema Penitenciario Cubano se ha consolidado como uno de los más humanitarios y justos, debido al perfeccionamiento de la legislación penitenciaria y su base reglamentaria, mejorando la calidad de vida y conducta social de los reclusos; erigiéndose el trabajo educativo como uno de los principales cimientos de las transformaciones experimentadas al interior de las instituciones carcelarias en la isla. (1)

En el año 1989 comienza en Cuba la informatización del Sistema Penitenciario, dando los primeros pasos con la automatización de los principales datos del recluso, así como algunos aspectos del control penal. Sin embargo, no es hasta años más tarde que mediante la orden 43/99 del Vice Ministro Primero se crea el Sistema Automatizado para el Control del Recluso (SACORE), tarea que es culminada en el año 2002, aunque su explotación no comienza hasta el año 2003. Este Sistema cuenta con tres módulos principales: Control Penal, Reeducación Penal y el Orden Interior, los cuales aumentan las especificaciones de la automatización de los datos principales del recluso y los aspectos de control penal existentes. En los 8 años de explotación del sistema, a pesar de sus facilidades y ayuda brindada a dichos centros aún cuenta con requisitos incompletos o pendientes. A la par con este sistema en los centros penitenciarios se utilizan otros dos sistemas como son: el Sistema Automatizado de Capacidades de la Dirección de Establecimientos Penitenciarios (SACDEP) y el Sistema de Automatización de Incidencias de la Dirección de Establecimientos Penitenciarios (SAIDEP), estos dos últimos también insuficientes para lograr una gestión eficiente de la información que en dicha institución se maneja. (2)

En el año 2008 la dirección del MININT[3] decide crear, en conjunto con la Universidad de las Ciencias Informáticas, el proyecto Prisiones Cuba, con el objetivo central de unificar los tres sistemas ya existentes

[3] Ministerio del Interior

en la Dirección de Establecimientos Penitenciarios y adicionarle un conjunto de funcionalidades necesarias para mejorar el funcionamiento de los procesos en los centros penitenciarios, aprovechando el desarrollo de las tecnologías de la información y las comunicaciones que se está llevando a cabo en el país. (2)

En el perfeccionamiento del sistema penitenciario y en correspondencia con las transformaciones desarrolladas en el campo educacional a nivel nacional, se ha instrumentado un conjunto de programas, proyectos y acciones que posibilita alcanzar con los internos en prisión mayores niveles de desarrollo educativo, y con ello resultados aún más efectivos en su rehabilitación y ulterior reinserción social. (1)

Uno de los problemas que presentan en la actualidad los sistemas utilizados en la Dirección de Establecimientos Penitenciarios está relacionado directamente con la planificación de las actividades educativas que se realizan en los centros penitenciarios, pues no existe un registro informático que sirva de apoyo para la realización de estos procesos. Registrándose de forma manual toda la información referente a la planificación y realización de las actividades educativas, situación que provoca que se genere un gran volumen de información. Pudiendo todo esto traer como consecuencia la pérdida de información.

Debido a lo anterior se plantea como **problema a resolver:** ¿Cómo gestionar de forma correcta la información de las actividades educativas en el sistema penitenciario cubano?

Definiendo a su vez como **objeto de estudio:** Los procesos de gestión de las actividades educativas.

Quedando definido como **campo de acción:** Informatización de los procesos de gestión de las actividades educativas del SIDEP.

Teniendo en cuenta lo anterior se plantea como **objetivo general:** Desarrollar el módulo Actividades Educativas del SIDEP a partir del análisis de los requisitos de software establecidos con el cliente y haciendo uso de la arquitectura y las tecnologías definidas por el proyecto.

Por lo que se plantea como **idea a defender:** El desarrollo del módulo Actividades Educativas del SIDEP permitirá la gestión de la planificación y ejecución de las actividades educativas.

Planteándose los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación.

- Realizar el modelo de diseño de los requisitos planteados para el módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.
- Implementar el módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.
- Validar las funcionalidades del módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.

Como apoyo a los objetivos específicos se plantearon las siguientes **tareas de investigación**:

- Investigación y análisis de otros sistemas similares.
- Descripción de las herramientas y tecnologías para dar solución al problema.
- Realización del diseño de los diagramas de clases para el módulo Actividades Educativas.
- Realización del diseño de base de datos para el módulo Actividades Educativas.
- Realización del diagrama de componentes para el módulo Actividades Educativas.
- Implementación del módulo Actividades Educativas.
- Diseño de los casos de prueba para el módulo Actividades Educativas.
- Realizar las pruebas de caja negra para el módulo Actividades Educativas.

Siendo sustentadas estas tareas por un conjunto de métodos de investigación:

Analítico-sintético: La utilización de este método se basa en el análisis de los sistemas penitenciarios: Sistema de Gestión Penitenciaria (SIGEP) y el Sistema Automatizado para el Control del Recluso (SACORE).

Modelación: Este método ayuda a comprender y analizar el sistema, mediante la realización de diferentes diagramas como los diagramas de Componentes y los Diagramas de Clases del Diseño.

Análisis documental: Basado en la exploración de toda la documentación generada por los analistas para el SIDEP, dígase, especificación de casos de usos, glosario de términos, diagramas de entidades así como también las especificaciones de la Arquitectura establecidas para la realización de este sistema.

Entrevista: Entrevista con la jefa de proyecto con el objetivo de obtener a través de una conversación planificada, información referente al sistema actualmente en uso en la DEP: SACORE.

Con el desarrollo del presente trabajo se esperan los siguientes aportes prácticos:

- El Modelo de Diseño del módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.
- El Modelo de Implementación del módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.
- Los componentes ejecutables del módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.
- La validación de las funcionalidades del módulo Actividades Educativas del Sistema Educativo de la Dirección de Establecimientos Penitenciarios.

El presente documento cuenta de 3 capítulos. En el primer capítulo, titulado Fundamentación Teórica, se realiza un estudio sobre los Sistemas Informáticos implantados en Centros Penitenciarios de varios países. Se analizan además las principales herramientas y tecnologías a utilizar para el desarrollo de los procesos relacionados con el diseño e implementación del módulo Actividades Educativas del Sistema Penitenciario Cubano.

En el capítulo 2, que lleva por título: Propuesta del módulo Actividades Educativas, se Aborda todo lo referente al diseño propuesto para la solución del software así como la descripción de la arquitectura a utilizar. Se especifican además el conjunto de funcionalidades a desarrollar dando un breve resumen de las mismas.

En el tercer capítulo, nombrado Implementación y Prueba, se genera el modelo de despliegue y los diagramas de componentes del módulo Actividades Educativas. Se abordan particularidades relacionadas con la implementación de la solución y se diseñan los casos de prueba para validar y verificar el funcionamiento de la solución propuesta.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1 Introducción

En el presente capítulo se realiza un estudio del uso de las aplicaciones informáticas para la automatización de los Sistemas Penitenciarios en el mundo y sus particularidades en Cuba, teniendo en cuenta la gestión de las actividades educativas. Se analizan además las principales herramientas y tecnologías a utilizar para el desarrollo de los procesos relacionados con el desarrollo del módulo Actividades Educativas del proyecto SIDEP.

1.1 Sistemas Informáticos en Sistemas Penitenciarios

Como parte del desarrollo tecnológico que vive el mundo en los últimos años los productos informáticos evolucionan dinámicamente para poder adaptarse a las necesidades tecnológicas actuales y los Sistemas Penitenciarios no han quedado exentos ante esta tendencia y al igual que disimiles organizaciones se han debido modernizar. Los viejos y rigurosos mecanismos de trabajo, así como los arduos procesos de gestión existentes durante décadas en las prisiones han sido sustituidos por eficientes Sistemas Informáticos capaces de gestionar dichos procesos. Son varios los países que cuentan con sistemas informáticos para la gestión penitenciaria. Estos sistemas tienen como objetivo facilitar el control de los reclusos y gestionar los procesos esenciales en los establecimientos penitenciarios relacionados con la población penal.

1.1.1 Sistema de Gestión Penitenciario (SIGEP)

El Sistema de Gestión Penitenciaria (SIGEP), desplegado en Venezuela y desarrollado en la Universidad de las Ciencias Informáticas tiene como objetivo general “desarrollar e implantar un sistema informático que soporte las decisiones estratégicas del Ministerio del Interior y Justicia y de la Dirección General de Custodia y Rehabilitación del Recluso”.

En este sistema, el módulo correspondiente al tratamiento de las actividades educativas lleva por nombre Cultura y Deporte. Este módulo registra la creación de los grupos pertenecientes al establecimiento penitenciario, dividiéndolos en dos tipos: grupos culturales y deportivos, los cuales se dividen en diferentes manifestaciones, para los grupos culturales: danza, pintura, teatro, etc., para los grupos deportivos: baloncesto, ajedrez, futbol, etc. La gestión de los grupos se realiza de forma diferente a las necesidades que presenta la Dirección de Establecimientos Penitenciarios de Cuba. El mismo, permite el registro de la ejecución de actividades deportivas y culturales. Siendo un inconveniente en este sistema que no permite

el registro de la planificación de las actividades. Durante el registro de la ejecución de una actividad se incorporan los internos que pertenecen a un grupo de la misma manifestación de la actividad que se realiza, sin permitir la incorporación de un interno que no esté asociado a ese grupo. Se registra solamente la participación activa del interno, sin permitir la participación como espectador, siendo esta una funcionalidad que pretende incorporar la DEP. (3)

El SIGEP está orientado según las características que presenta la región y de acorde a las leyes y regulaciones jurídicas vigentes en Venezuela.

El proyecto Humanización Penitenciaria tiene como objetivo la realización de un software semejante a este pero acorde a la organización del Sistema Penitenciario Cubano y basado en las leyes y regulaciones jurídicas vigentes en Cuba.

1.1.2 Sistema Automatizado para el Control del Recluso (SACORE)

Fue puesto en práctica a partir del 2003 garantizando respuestas inmediatas a las solicitudes de información de los diferentes órganos e instituciones, es un sistema que recoge prácticamente la totalidad de la información de los reclusos en todas las especialidades. Cuenta con más de 200 reportes impresos y permite el traslado automático de todos los datos del recluso al nivel nacional. A pesar de que este sistema resolvió una gran problemática en el momento de su creación, en la actualidad no resuelve todas las necesidades de información necesarias al departamento de dirección de la Dirección de Establecimientos Penitenciarios (DEP). Con la puesta en prácticas de este sistema, el Ministerio del Interior abre paso al desarrollo tecnológico e informático extendiendo su radio de acción a nuevas herramientas automatizadas que complementan la información del SACORE: El sistema de incidencias y el de capacidades (SAIDEP y SACDEP).

Aún con la existencia del SACORE muchos datos son manejados y conservados en formato duro, o a través de herramientas informáticas que no ofrecen grandes facilidades para su gestión. Una de las deficiencias que presenta este sistema es que no existe un área dedicada a la informatización de los procesos vinculados a la gestión de actividades educativas. Por consecuencia, no existe forma alguna de planificar las actividades, así como el registro de su ejecución y el registro de otras acciones de apoyo que se realizan como parte de estas actividades. Este sistema, tampoco tiene en cuenta la gestión de la información referente a los grupos formados en las unidades y colectivos, o el rol que cumplen los internos

asociados a estos. Tampoco se registran los convenios que se llevan a cabo con otras instituciones para realizar actividades de este orden. (2)

Por todas estas razones expuestas anteriormente, es interés de la dirección de la DEP, la realización del módulo Actividades Educativas; que permitirá el manejo de la información generada a partir de los procesos de gestión de las actividades educativas.

1.2 Tecnologías y herramientas utilizadas para el desarrollo de la aplicación

A continuación se realiza una breve descripción de la metodología, herramientas y tecnologías propuestas por el equipo de arquitectos del proyecto Prisiones Cuba.

1.2.1 Metodología de desarrollo

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares. Está formada por fases, cada una de las cuales se puede dividir en sub-fases que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo. (4)

Para el desarrollo del SIDEP el equipo de arquitectura optó por utilizar el Proceso Unificado de Desarrollo (RUP), que constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Permite que el trabajo sea más ágil y organizado, ya que divide en 4 fases el desarrollo del software (Inicio, Elaboración, Construcción, Transición), como se muestra en la Figura 1.

La justificación de la selección de RUP como metodología de desarrollo está sustentada en las características de esta metodología y en que para el desarrollo del SIDEP, que es un sistema de gran envergadura, es necesario atravesar por todas las fases que RUP ofrece.

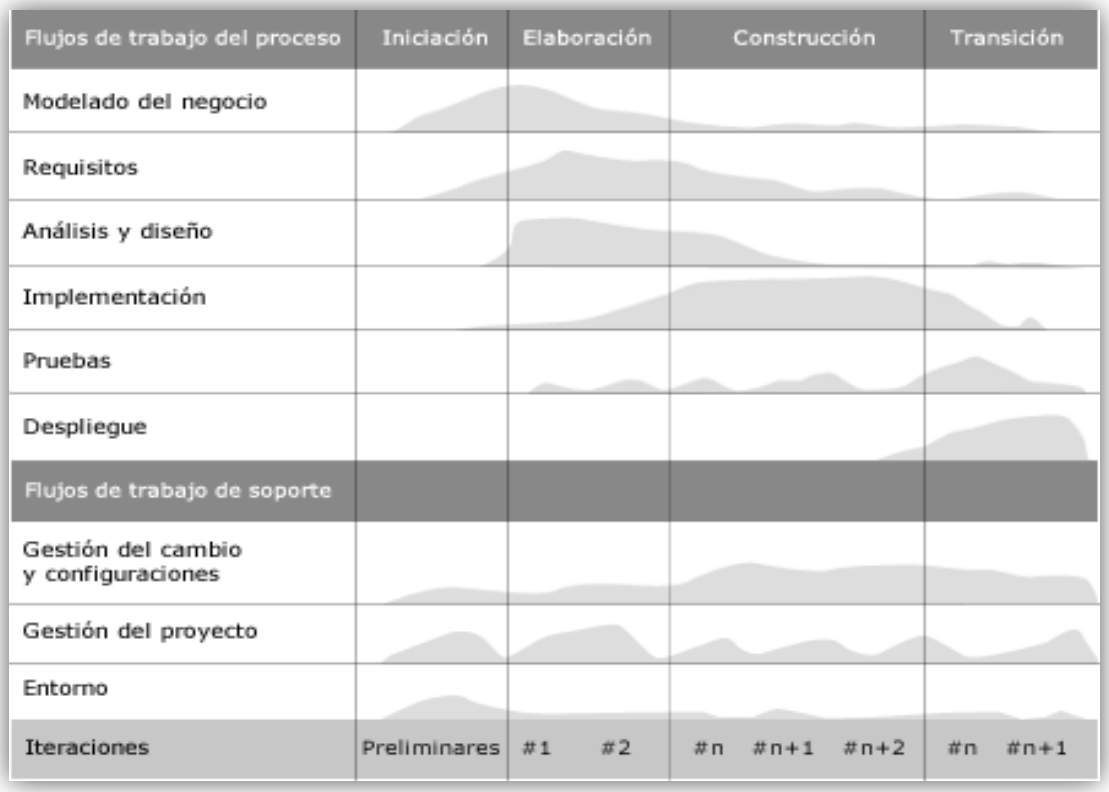

Figura 1: Ciclo de vida de RUP.

RUP define los trabajadores (roles) y sus respectivas responsabilidades (actividades), los artefactos (productos) que se deben generar y los flujos de trabajo de las disciplinas que se definen como secuencia de actividades realizadas por trabajadores y que producen un resultado de valor observable. (4)

De forma general está caracterizada por los siguientes aspectos:

- **Guiado por casos de uso:** los casos de uso son el instrumento para validar la arquitectura del software y extraer los casos de prueba.
- **Centrado en la arquitectura:** los modelos son proyecciones del análisis y el diseño, constituyen la arquitectura del producto que se desea desarrollar.
- **Iterativo e incremental:** durante todo el proceso de desarrollo se producen versiones incrementales (que se acercan al producto terminado) del producto en desarrollo.

1.2.2 Tecnologías

- **JavaScript**

En el SIDEP este lenguaje es utilizado para las validaciones de entrada de datos en el lado del cliente. Es un lenguaje de programación "ligero" y orientado a objetos. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model[4] (DOM).

Tradicionalmente se venía utilizando en páginas web HTML[5] para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor; implementándose también de esta forma en la actualidad. JavaScript se interpreta en el **agente de usuario**[6], al mismo tiempo que las sentencias van descargándose junto con el código HTML. (5)

- **UML[7] 2.0**

Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas. Permite la visualización, especificación, construcción y documentación de los artefactos de sistemas. Básicamente facilita a los desarrolladores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados. UML ofrece un estándar para escribir un "plano" del sistema, incluyendo aspectos conceptuales tales como: procesos de negocios y funciones del sistema, y aspectos concretos como: expresiones de lenguajes de programación, esquemas de bases

[4] Document Object Model (Modelo de Objeto del Documento)

[5] Hypertext Markup Language (Lenguaje de Marcas de Hipertextos)

[6] Aplicación informática que funciona como cliente en un protocolo de red

[7] Unified Modeling Language (Lenguaje Unificado de Modelado)

de datos y componentes de software reutilizables. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP). (6)

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. Algunos de ellos son:

- Diagrama de clases.
- Diagrama de componentes.
- Diagrama de despliegue.
- Diagrama de casos de uso.
- Diagrama de secuencia.

El uso de UML permitió la realización de los diagramas de clases, de componentes, de despliegue y secuencia; como parte del proceso de diseño del módulo Actividades Educativas.

- **Dojo 1.5**

Es una librería de clases JavaScript utilizada como framework de presentación, seleccionado por el equipo de arquitectura del proyecto. Contiene Apis[8] y widgets (controles) para facilitar el desarrollo de aplicaciones Web. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop Apis, widget Apis, abstracción de eventos, almacenamiento de Apis en el cliente, e interacción de Apis con AJAX.

Resuelve asuntos de usabilidad comunes como pueden ser la navegación y detección del navegador, soportar cambios de URL[9] en la barra de URLs para luego regresar a ellas, y la habilidad de degradar cuando AJAX/JavaScript no es completamente soportado en el cliente. Proporciona una gama más amplia de opciones en una sola biblioteca JavaScript y es compatible con navegadores antiguos. (7)

- **Java EE**

Java Enterprise Edition es un conjunto de componentes que forman una plataforma de programación para el desarrollo y ejecución de aplicaciones desarrolladas en lenguaje de programación Java con arquitectura

[8] *Application Programming Interface* (Interfaz de programación de aplicaciones)

[9] Universal Resource Location (Localizador Universal de Recursos)

de N capas distribuidas, que se apoya ampliamente en componentes de software modulares ejecutándose en un servidor de aplicaciones. (8)

- **Groovy 1.7.8**

Es el lenguaje utilizado para el desarrollo de aplicaciones Grails .Es un lenguaje dinámico basado en la máquina virtual de Java que tiene como ventaja que su sintaxis es compatible con Java, pero añade características dinámicas y sintácticas inspiradas en lenguajes como Python, Ruby y Smalltalk que ahorran muchas líneas de código. Precisamente su similitud con Java permite que la curva de aprendizaje para los desarrolladores familiarizados con Java sea casi nula. El código fuente Groovy se compila a bytecodes igual que Java, y es posible instanciar objetos java desde Groovy. Groovy incluye mejoras sintácticas en relación con la facilidad de manejos de objetos mediante nuevas expresiones y sintaxis, distintas formas de declaración de literales. Control de flujo avanzado mediante nuevas estructuras, y nuevos tipos de datos, con operaciones y expresiones específicas. En resumen: el código Groovy es mucho más breve que el de Java. (9)

- **Oracle 11g R2**

Es un Sistema Gestor de Bases de Datos con características objeto-relacionales, que utiliza tecnología cliente/servidor. Permite la gestión de grandes bases de datos, un alto rendimiento en transacciones, disponibilidad controlada de los datos de las aplicaciones, la gestión de la seguridad y la autogestión de la integridad de los datos. Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando: soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma. (10)

- **Grails 1.3.7**

Grails es un framework para aplicaciones web libre desarrollado sobre el lenguaje de programación Groovy (el cual a su vez se basa en la plataforma Java). Grails pretende ser un marco de trabajo altamente productivo siguiendo paradigmas tales como convención sobre configuración o no te repitas (DRY o Don't Repeat Yourself), proporcionando un entorno de desarrollo estandarizado y ocultando gran parte de los detalles de configuración al programador.

Debido a que aprovecha características dinámicas de Groovy es ágil y dinámico permitiendo hacer cosas que son difíciles en otros frameworks basados en Java.

Grails se puede ampliar a través de plugins. Por su dinamismo es capaz de acortar el ciclo de desarrollo, ahorrando tiempo de trabajo y agilizándolo. Incluye un contenedor web, base de datos, sistemas de construcción y pruebas. Esta combinación puede reducir el tiempo inicial del proyecto y el tiempo de instalación del desarrollador a minutos en lugar de horas.

Sus principales características son:

- **Alta productividad:** Grails tiene tres características que intentan incrementar su productividad comparándolo con los Framework Java tradicionales:
    1. Inexistencia de configuración XML
    2. Entorno de desarrollo preparado para funcionar desde el primer momento
    3. Funcionalidad disponible mediante métodos dinámicos.
- **Integración con la plataforma Java:** Al estar construido sobre la plataforma Java, con lo que es muy fácil integrarlo con librerías Java, Framework y código existente. La mejor característica que Grails ofrece en este ámbito es una integración transparente con clases mapeada mediante el Framework Hibernate ORM[10] . Esto significa que aplicaciones existentes que utilicen Hibernate pueden utilizar Grails sin recompilar el código o reconfigurar las clases Hibernate, aprovechando los métodos de persistencia que se mencionan anteriormente. Una consecuencia es que se puede utilizar scaffolding[11] con las clases Java mapeadas con Hibernate. Otra consecuencia es que las capacidades de Grails están totalmente disponibles para estas clases y las aplicaciones que las usan.
- **Persistencia:** El modelo de datos en Grails se graba en la base de datos utilizando GORM (Grails Object Relational Mapping). Las clases de dominio se guardan en el directorio *grails-app/domain.*

Grails está construido sobre 5 fuertes pilares:

- **Groovy:** para la creación de propiedades y métodos dinámicos en los objetos de la aplicación.
- **Hibernate:** para la persistencia.

[10] Mapeo de Objeto Relacional, Object Relational Mapping

[11] Generación automática de código que implementa las cuatro funcionalidades básicas :crear, leer, modificar y eliminar

- **Spring:** para los flujos de trabajo e inyección de dependencias.
- **SiteMesh:** un framework robusto y estable para mostrar las páginas de presentación.
- **Ant:** para la gestión del proceso de desarrollo.

1.2.3 Herramientas

- **NetBeans 7.0**

Es utilizado como ambiente de desarrollo pues trae incluido entre sus novedades más destacadas el uso de Groovy, además de permitir el empleo de módulos (en inglés plug-in) para ampliar sus funcionalidades: se utilizó el módulo Groovy, que provee soporte para el lenguaje Groovy y el framework Grails. El IDE[12] NetBeans es un reconocido entorno de desarrollo integrado disponible para Windows, Mac, Linux y Solaris, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. El proyecto NetBeans está formado por un IDE de código abierto y una plataforma de aplicación que permite a los desarrolladores crear con rapidez aplicaciones web, empresariales, de escritorio y móviles.

Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactorización. Cuenta con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos planos, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente. Puede extenderse usando otros lenguajes de programación como son C/C++, Phyton. Es un producto libre y gratuito sin restricciones de uso. (11)

- **Embarcadero ER Studio 8.0**

Es una herramienta de modelado de datos fácil de usar y multinivel, para el diseño y construcción de bases de datos a nivel físico y lógico. Direcciona las necesidades diarias de los administradores de bases de datos, desarrolladores y arquitectos de datos que construyen y mantienen aplicaciones de bases de datos grandes y complejas. ER/Studio está equipado para crear y manejar diseños de bases de datos

[12] Integrated Development Environment (Entorno de Desarrollo Integrado)

funcionales y confiables. Ofrece fuertes capacidades de diseño lógico, sincronización bidireccional de los diseños físicos y lógicos, construcción automática de bases de datos, documentación y fácil creación de reportes. (12)

ER/Studio ofrece las siguientes funcionalidades:

- Capacidad fuerte en el diseño lógico.
- Sincronización bidireccional de los diseños lógico y físico.
- Construcción automática de Base de Datos.
- Reingeniería inversa de Base de Datos.
- Documentación basada en HTML.
- Un Repositorio para el modelado.

La utilización de esta herramienta permitió la realización del diseño físico de la base de datos.

- **Apache Tomcat 6.0**

Tomcat es el servidor web más utilizado a la hora de trabajar con Java en entornos web, es una implementación completamente funcional de los estándares de JSP y Servlets, desarrollado en código abierto en lenguaje Java por lo que funciona en cualquier sistema operativo que disponga de una máquina virtual Java. Es desarrollado en un entorno abierto y participativo. Publicado bajo la licencia Apache versión 2, siendo una marca registrada de la Fundación de Software Apache. Teniendo además una aplicación de administración intuitiva basada en web. (13)

En el caso del SIDEP se selecciona como servidor de aplicaciones debido precisamente a que la plataforma de desarrollo que se utiliza es Java.

- **Visual Paradigm 8.0**

Visual Paradigm para UML (VP-UML) es una herramienta CASE[13] de diseño UML. Su mayor éxito radica en la capacidad de ejecutarse sobre diferentes sistemas operativos lo que le confiere la característica de ser multiplataforma. Constituye un software privativo y sus distintas ediciones son compatibles.

Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Utiliza UML como lenguaje de modelado, ofreciendo soluciones de

[13] Computer Aided Software Engineering (Ingeniería de Software Asistida por Computadora)

software que permiten a las organizaciones desarrollar las aplicaciones con mayor calidad de forma rápida, satisfactoria y barata. Es fácil de usar y presenta un entorno gráfico agradable para el usuario. Facilita una excelente interoperabilidad con otras herramientas CASE y principales IDE de desarrollo. (14)

Permite:

- Realizar el modelado de base de datos, el modelado de requerimientos, la interoperabilidad, la generación de documentación y la generación de código base para diferentes lenguajes de programación como Java, C# y PHP además de permitir la integración con herramientas de desarrollo (IDE).
- Dibujar todos los tipos de diagramas de clases, código inverso, y generar código desde diagramas.

Durante la fase de diseño de la aplicación sirvió para la realización de los distintos diagramas de clase: diagrama de componentes, de interacción y diagrama de clases del diseño.

1.3 Conclusiones Parciales

Luego del análisis de algunos sistemas informáticos para centros penitenciarios utilizados en otros países, se llegó a la conclusión de que estos presentan algunas similitudes en cuanto al desarrollo de las actividades educativas. Estos sistemas se rigen por las regulaciones y leyes vigentes del país donde son implementados, por lo que muchos de estos procedimientos no serian aplicables a nuestro país. Se realizó un estudio del sistema en uso actualmente en la DEP demostrándose sus deficiencias en cuanto a la gestión de las actividades educativas. Todo esto demuestra la necesidad de desarrollar una aplicación que cumpla las necesidades de la DEP de Cuba.

Se abordó RUP como metodología de desarrollo utilizada; herramientas tales como: el NetBeans 7.0, Embarcadero ER Studio 8.0, Apache Tomcat 7.0 y Visual Paradigm 7.0; además de las tecnologías utilizadas: JavaScript, UML, Dojo 1.5, la plataforma Java EE, el lenguaje Groovy 1.7.8, Oracle 11g R2 como gestor de base de datos, y el framework de desarrollo Grails 1.3.7 .

CAPÍTULO 2: PROPUESTA DEL MÓDULO ACTIVIDADES EDUCATIVAS

2 Introducción

En el presente capítulo se abordan los principales elementos presentes en el desarrollo del producto. Se describen las funcionalidades del módulo Actividades Educativas y se abordan temas fundamentales como la composición de la arquitectura del sistema, los diagramas de clases e interacción, así como el diseño de la base de datos para la persistencia de la información.

2.1 Descripción de los casos de uso del módulo Actividades Educativas

En la siguiente tabla se realiza una breve descripción de los casos de uso que modelan el negocio a realizar en el módulo Actividades Educativas. (15)

Nombre del Caso de Uso	Descripción
Gestionar Plan de Actividades de Tratamiento	Permite registrar, consultar o actualizar un Plan de Actividades de Tratamiento.
Gestionar Plan Político Educativo del Colectivo	Permite registrar, consultar o actualizar un Plan Político Educativo del Colectivo.
Registrar actividad al Plan de Tratamiento	Permite registrar una nueva actividad en el Plan de Actividades de Tratamiento
Registrar actividad al Plan Político Educativo	Permite registrar una nueva actividad en el Plan Político Educativo de un colectivo.
Registrar ejecución de actividad	El caso de uso permite registrar la ejecución de una actividad.
Gestionar Convenio Educativo	Brinda la posibilidad de registrar, consultar o actualizar un convenio educativo.
Consultar planes pasados	Brinda la posibilidad de consultar los detalles de planes pasados.
Registrar incidencia con los medios	Registra una incidencia con los medios de la actividad educativa.
Consultar listado de incidencias con los medios	Permite consultar un listado de las incidencias ocurridas con los medios en una actividad educativa.
Gestionar Plan de Aseguramiento de Actividad	Permite registrar el Plan de Aseguramiento a una actividad creada en el Plan de Actividades de Tratamiento.
Gestionar grupo del colectivo	Brinda la posibilidad de registrar, modificar, consultar o eliminar un grupo artístico o deportivo del colectivo.
Gestionar grupo de la unidad	Permite registrar, modificar, consultar o eliminar un grupo artístico o deportivo de la unidad.

Tabla 1: Descripción de casos de uso

2.2 Arquitectura del sistema

La arquitectura del sistema está basada en la propuesta por Grails, el framework utilizado para el desarrollo de la aplicación. Este propone la utilización de un patrón muy popular en el desarrollo de aplicaciones web, denominado Modelo-Vista-Controlador (MVC). Este patrón establece que los componentes de un sistema de software debe organizarse en 3 capas lógicas principales: capa web, capa de control, y la capa de datos, en el caso de Grails se propone la utilización de una cuarta capa: la capa de servicios (16). La capa de presentación interactúa con la capa de servicios mediante los controladores y desde la capa de servicios se accede a la capa de datos mediante los servicios que esta brinda. Cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Esta arquitectura en capas por su diseño proporciona la facilidad de modificar cada capa todo lo posible sin infligir daños o alteraciones a la capa inmediata, a continuación se brinda una descripción de las mismas:

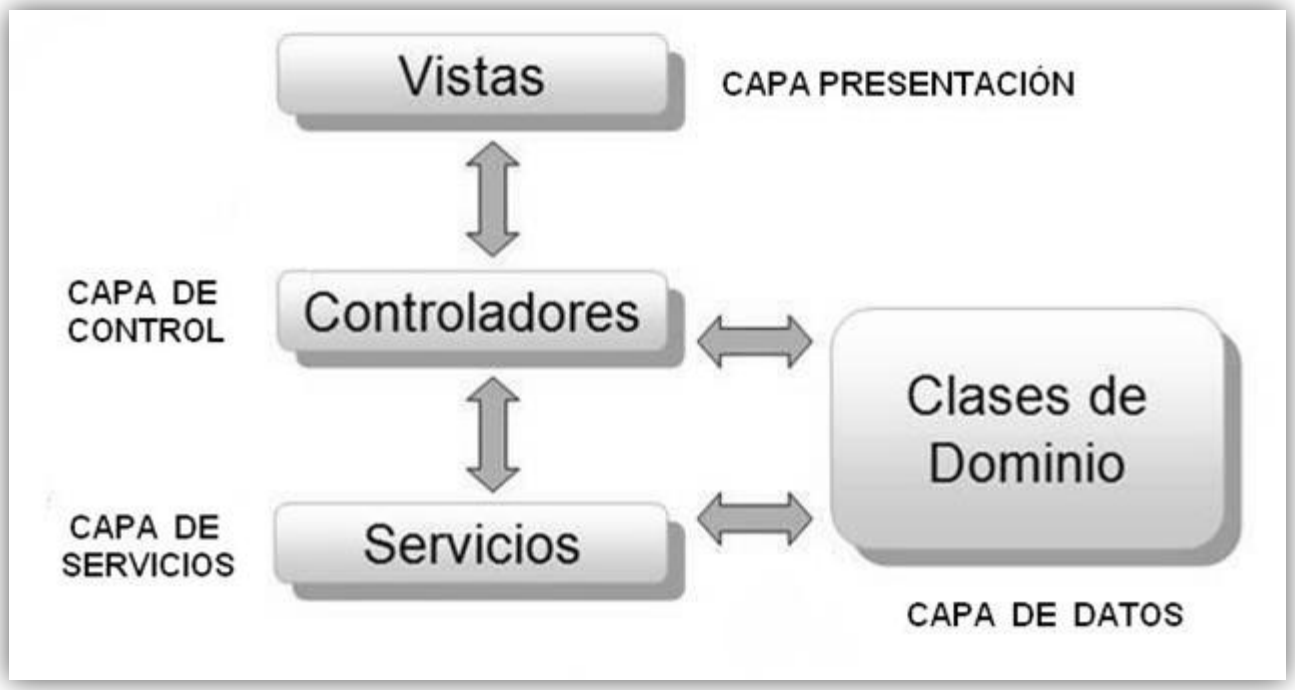

Figura 2: Arquitectura en Capas.

- **Capa de Presentación:** Grails utiliza para la interacción con el usuario la tecnología GSP basada en una implementación mediante JSP[14]. El mismo permite a los desarrolladores mezclar etiquetas de lenguajes de marcas tradicionales como HTML con código Groovy para producir vistas dinámicas. Las vistas son los recursos que junto al modelo generado por los controladores le permiten al cliente visualizar la información, estos pueden ser páginas HTML, documentos en formato PDF, hojas de cálculo, entre otras. En esta capa se utilizan diversos componentes como:
  - CSS
  - JavaScript
  - Clases de presentación
  - TagLib

La ubicación de estos componentes se puede observar en la siguiente figura.

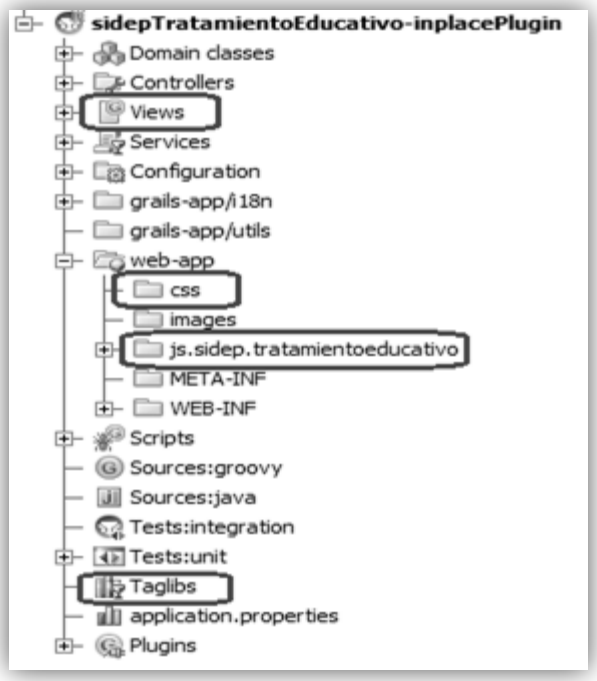

Figura 3: Componentes de la Capa de Presentación.

[14] ***P**ágina **S**ervidora de **J**ava*, **J**ava **S**erver **P**age.

- **Capa de control:** Esta será la capa responsable de implementar la lógica de presentación, que se realizara en las clases controladoras ubicadas en el paquete *Controllers*. Un controlador es una clase utilizada para el manejo de los pedidos provenientes de la aplicación. El controlador recibe la petición, realiza algún trabajo potencial con la misma y finalmente decide que sucederá a continuación, lo cual puede incluir algunos de los siguientes flujos.
    - Ejecutar otra función de controlador.
    - Mostrar una vista.
    - Mostrar información directamente con la respuesta de la petición.

Un controlador es prototipado, lo cual significa que una nueva instancia es creada por cada petición. Proveen la entrada principal para cualquier aplicación de Grails, coordinando los pedidos entrantes, delegando hacia los servicios o clases de dominio para la lógica de negocios y mostrando las vistas.

Figura 4: Paquete de controladores.

- **Capa de servicio:** En esta capa se encapsula toda la lógica de la aplicación en fachadas de negocio que son utilizadas por los controladores y se exponen algunos procesos de negocio a través de interfaces de servicios. A estas fachadas de negocio se le puede aplicar la seguridad a

nivel de métodos y de objetos de negocio, auditorías, cache, política de transacciones, entre otros. Sus clases radicarán según la arquitectura propuesta por Grails en el paquete *Services*.

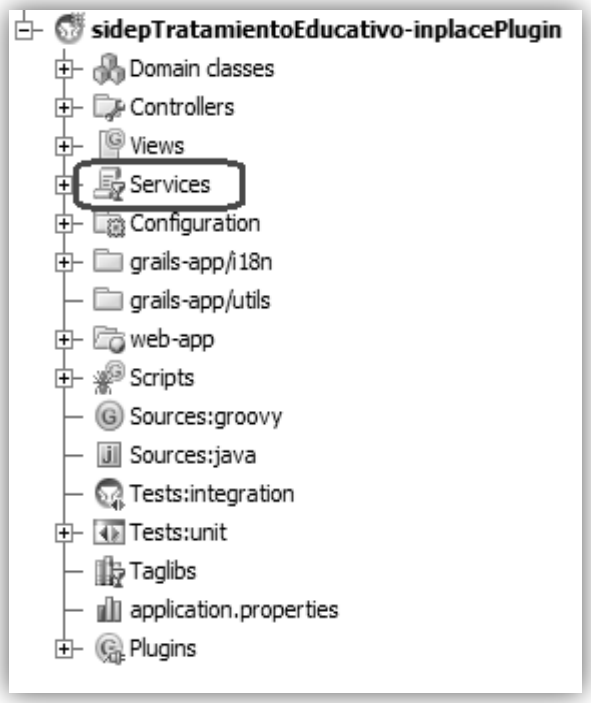

Figura 5: Paquete de Servicios.

- **Capa de datos:** Maneja los objetos de acceso a datos abstrayéndolos del mecanismo de persistencia usado; a través de interfaces que exponen las operaciones de persistencia. Para evitar trabajar directamente con un gestor de base de datos y sus tablas y permitir trabajar con objetos en su lugar, Grails utiliza Hibernate como una herramienta ORM. Pero esta vez, dada la naturaleza dinámica de Grails y la adopción del convenio sobre la configuración; crea una versión superior de una nueva implementación de Hibernate llamado Grails Objeto Mapeo Relacional (GORM) que simplifica el trabajo con Hibernate y elimina cualquier configuración externa (16). El modelo de datos en una aplicación Grails está compuesto por las clases del dominio, que se ubican en el paquete *Domain classes.*

Figura 6: Paquete de clases del dominio.

2.3 Patrones de diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Estos se caracterizan por estar conformados por un conjunto de elementos, como: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios). (17)

2.4.1 Inversión de Control (IoC)

La inversión de control es un patrón utilizado en grails según el cual las dependencias de un componente no deben gestionarse desde el propio componente para que este solo contenga la lógica necesaria para hacer su trabajo. Cuando se crea un componente en la aplicación, Grails configura a Spring para controlar su ciclo de vida (cuándo se crea, cuántas instancias mantiene vivas a la vez, cómo se destruyen, etc.) y sus dependencias (que otros componentes necesita para desarrollar su trabajo y como seguirlos). El objetivo de esta técnica es mantener los componentes lo más sencillo posible, incluyendo únicamente

código que tenga relación con la lógica de negocio. Así la aplicación será más fácil de comprender y mantener. (16)

2.4.2 Patrones Grasp

De los patrones Grasp[15] se utilizarán de forma general los patrones Alta Cohesión, Bajo Acoplamiento, Experto, Controlador y Creador. Los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor de acoplamiento.

- **Alta Cohesión**: La alta cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme (17). Este patrón fue utilizado en el tratamiento de las actividades educativas, pues existen varios tipos de actividades que requieren ser tratadas de forma diferente, por lo cual se hizo necesaria la utilización de un servicio asociado a un controlador para el tratamiento de las actividades. Dividiendo de esta forma el proceso de planificación de las diferentes actividades.
- **Bajo Acoplamiento**: Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases (17). Este patrón se pone de manifiesto en el proceso de desarrollo del software al relacionar un controlador con un servicio, pudiendo en algunas ocasiones llegar a relacionarse un controlador con varios servicios.
- **Experto**: Asigna una responsabilidad al más competente en información, donde la clase cuenta con la información necesaria para cumplir la responsabilidad. Es el principio básico de asignación de responsabilidades que suele utilizarse en el diseño Orientado a Objetos. Es el patrón que más se usa para asignar responsabilidades (17). Grails implementa este patrón al utilizar GORM para el mapeo de objetos de la base de datos. Las tablas de la base de datos son mapeadas según lo definido en la clase del dominio, siendo esta clase la experta en información de la entidad que representa.

[15] *Patrones generales de software para asignación de responsabilidades,* General Responsibility Assignment Software Patterns

- **Controlador**: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado (17). Los controladores son un ejemplo del uso de este patrón, siendo los encargados de manejar los eventos y mensajes entre dos capas, por ejemplo, entre las vistas y los servicios.
- **Creador:** Este patrón plantea la asignarle a una clase B la responsabilidad de crear una instancia de la clase A. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador se da soporte al bajo acoplamiento. (17)

2.4.3 Patrones de diseño GOF

Los patrones GOF [16] son patrones de diseño que definen una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. De los patrones que propone el grupo de los patrones GOF se utilizará el Singleton. Dicho patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Su uso se pone de manifiesto en la utilización de los servicios, pues estos por defecto usan Singleton, pues existe una sola instancia de la clase que se usará en todos los artefactos que declaren la variable asociada a esta. Tiene como inconveniente que no se puede guardar de forma privada la información de una petición realizada por un usuario, pues otro usuario que estuviera trabajando en la misma sección del software vería instantáneamente cualquier modificación. Este inconveniente puede eliminarse haciendo uso de la variable “scope”, asignándole el valor “session”, creando de esta forma una instancia diferente del servicio para cada sesión.

2.4 Diagramas de clases

2.5.1 Diagramas de clases del diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la

[16] Gang of Four, Banda de los Cuatro

información que se manejará en el sistema y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

A continuación se muestra el diseño del diagrama de clases del CU Gestionar Plan de Actividades de Tratamiento.

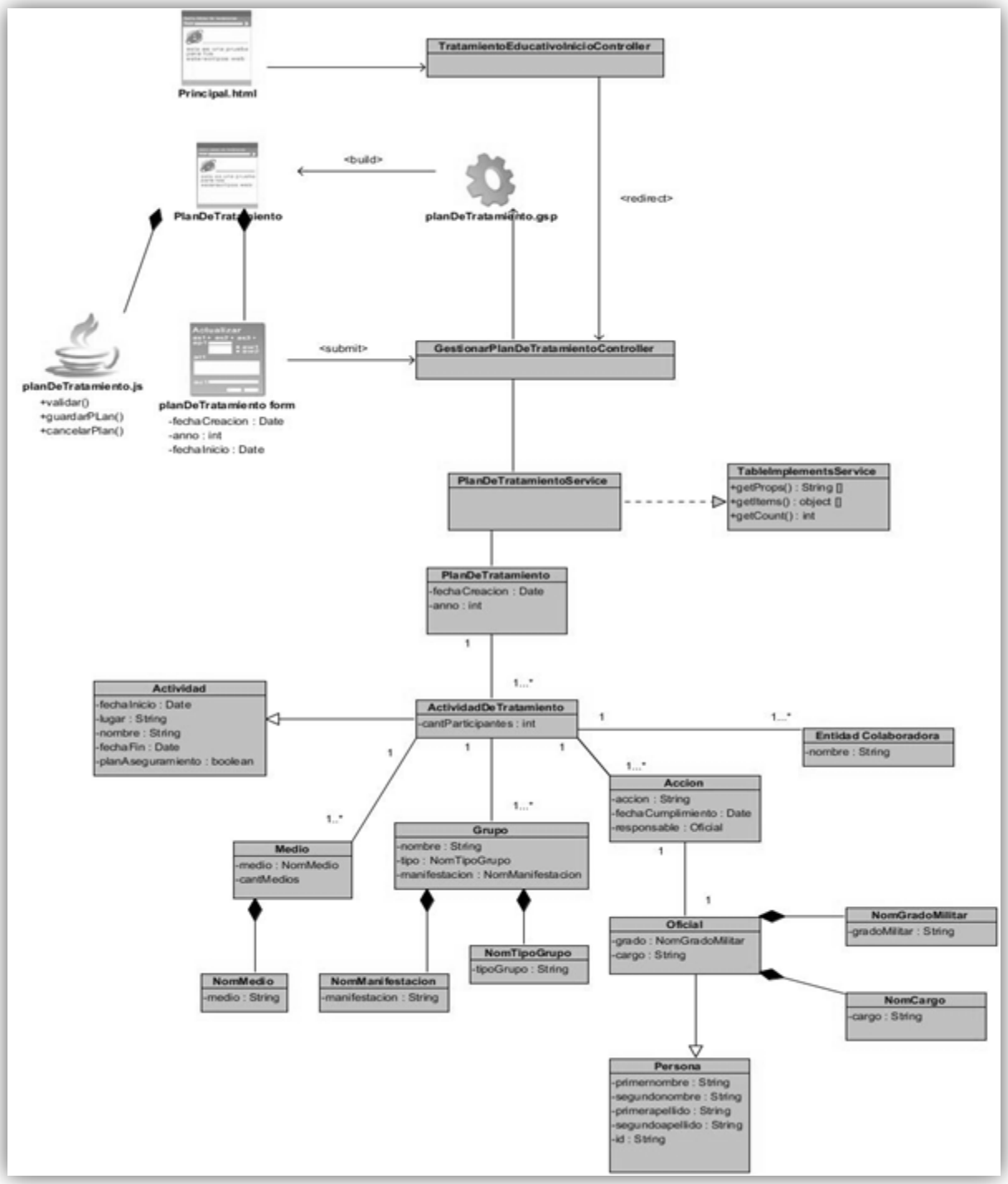

Figura 7: CU Gestionar Plan de Actividades de Tratamiento.

En el diagrama anterior se pude observar la relación que existe entre las entidades que intervienen durante la creación de un Plan de Actividades de Tratamiento.

El Plan de Actividades de Tratamiento contiene un conjunto de actividades de tratamiento, estas son las actividades que se planifican de forma anual para una unidad y afectan a los colectivos en dependencia de los grupos que participan. Presenta una relación de herencia con la clase Actividad, al compartir datos en común con esta y añadir otros nuevos. Existen dos tipos de actividades: Actividad de Tratamiento y Actividad educativa. Al crearse una actividad de tratamiento es necesario se asocian a esta las entidades colaboradoras (**Entidad Colaboradora**) que se verán implicadas en el desarrollo de la actividad. Es necesario también especificar los grupos (**Grupo**) que participarán, pues el colectivo al que pertenece el grupo se verá afectado con la realización de la actividad. Se relacionan también los medios (**Medio**) que se utilizarán y un conjunto de acciones (**Accion**) que puedan ser necesarias para el desarrollo de la actividad .Teniendo cada acción un oficial (**Oficial**) responsable de su realización.

2.5.2 Descripción de las clases

A continuación se realiza una descripción de las clases fundamentales presentes en el CU Gestionar Plan de Actividades de Tratamiento.

Nombre de la vista: planDeTratamiento. html		
Tipo de clase: Interfaz		
Descripción: En ella se registran los datos pertenecientes a un plan de actividades de tratamiento. Cuenta con un formulario en el cual se muestran los datos necesarios para crear el plan y con un archivo con extensión JavaScript para la validación de los errores en el momento de introducir los datos.		
Atributo	**Tipo**	**Descripción**
fechaCreacion	fecha	Guarda la fecha en que se crea el plan.
anno	número	Guarda el año en que se confecciona el plan.

Tabla 2: Descripción de la vista Plan de Tratamiento.

Nombre de la vista: actividadDeTratamiento. html		
Tipo de clase: Interfaz		
Descripción: Esta vista permite registrar los datos pertenecientes a las actividades de un Plan de Tratemiento. Cuenta con un formulario en el cual se muestran los datos necesarios para crear las actividades y con un archivo con extensión JavaScript para la validación de los errores en el momento de introducir los datos.		
Atributo	**Tipo**	**Descripción**

fecha	date	Guarda la fecha en la que se realizará la actividad
lugar	string	Guarda donde se realizará una actividad perteneciente al plan.
nombre	string	Guarda el nombre de una actividad perteneciente al plan.
tipoActividad	NomTipoActEd	Guarda los tipos de actividades que pueden realizarse.
fechaFin	date	Guarda la fecha en que terminará de realizarse una actividad perteneciente al plan.
aseguramiento	boolean	Este campo especifica si la actividad perteneciente al plan lleva incluida un plan de aseguramiento.
cantParticipantes	int	Guarda la cantidad de participantes a una actividad del plan.
nombreAccion	string	Guarda la acción que se desarrollará en una actividad perteneciente al plan.
fechaCumplimiento	date	Guarda la fecha en que debe dársele cumplimiento a la acción.
responsable	Oficial	Representa la relación entre la clase Accion y la clase Oficial. Siendo el oficial el responsable de dar cumplimiento a la acción.

Tabla 3: Descripción de la vista Plan de Tratamiento.

Nombre del controlador: PlanDeTratamientoController		
Tipo de clase: Controladora		
Descripción: Se encarga de gestionar los datos pertenecientes a un Plan de Tratamiento		
Atributo	**Tipo**	**Descripción**
planDeTratamientoService	PlanDeTratamientoService	Registra el Plan de Tratamiento.
Acciones		
Nombre	**Descripción**	
index()	Redirecciona a la interfaz planDeTratamiento.	
finalizarPlan ()	Guarda un Plan de Tratamiento utilizando el servicio planDeTratamientoService.	
updatePlan()	Permite actualizar un Plan de Tratamiento.	

Tabla 4: Descripción del controlador Plan de Tratamiento.

Nombre del fichero: actividadDeTratamiento. js

Tipo de clase: complemento de interfaz de usuario	
Descripción: Es la encargada de validar los datos pertenecientes a un Actividad de Tratamiento y mandar las peticiones al servidor.	
Funciones	**Descripción**
addActividad()	Corresponde al botón adicionarActividad que realiza la acción de guardar una actividad.
addAccion()	Corresponde al botón adicionarAccion que tiene la función de guardar una acción.
addMedio()	Corresponde al botón adicionarMedio que tiene la función de guardar un medio.
validarActividad()	Valida los datos de una actividad introducidos por el usuario.
validarAccion()	Valida los datos de una acción introducidos por el usuario.
validarMedio()	Valida los datos de un medio introducido por el usuario.

Tabla 5: Descripción del JavaScript Actividad de Tratamiento.

Nombre del fichero: PlanDeTratamiento.js	
Tipo de clase: complemento de interfaz de usuario	
Descripción: Es la encargada de validar los datos pertenecientes a un Plan de Tratamiento y mandar las peticiones al servidor.	
Funciones	**Descripción**
validarPlan()	Valida los datos correspondientes a un plan de tratamiento.

Tabla 6: Descripción del JavaScript Plan de Tratamiento.

Nombre de la clase: PlanDeTratamientoService		
Tipo de clase: clase del servicio		
Descripción: Es la encargada de salvar un Plan de Tratamiento		
Atributo	**Tipo**	**Descripción**
planT	PlanTratamiento	Almacena un plan de tratamiento.
Acciones	**Descripción**	
salvarPlan(actividad: actividad)	Recoge los datos del Plan de Tratamiento para guardarlo.	

Tabla 7: Descripción del servicio Plan de Tratamiento.

Nombre de la clase: Plan de Tratamiento	
Tipo de clase: Entidad	
Atributo	**Tipo**
fecha de creación	date

año	número

Tabla 8: Descripción de la clase Plan de Tratamiento

Nombre de la clase: Actividad	
Tipo de clase: Entidad	
Atributo	**Tipo**
fecha	date
lugar	string
nombre	string
tipoActividad	NomTipoActEd
fechaFin	date
aseguramiento	boolean

Tabla 9: Descripción de la clase Actividad.

Nombre de la clase: Actividad de Tratamiento	
Tipo de clase: Entidad	
Atributo	**Tipo**
cantParticipantes	int
existPAseg	boolean

Tabla 10: Descripción de la clase Actividad de Tratamiento.

Nombre de la clase: Accion	
Tipo de clase: Entidad	
Atributo	**Tipo**
accion	string
fechaCumplimiento	date
responsable	Oficial

Tabla 11: Descripción de la clase Acción.

Nombre de la clase: Grupo	
Tipo de clase: Entidad	
Atributo	**Tipo**
nombreGrupo	string
tipoGrupo	NomTipoGrupo
manifestacion	NomManifestacion

Tabla 12: Descripción de la clase Grupo.

Nombre de la clase: Medio

Tipo de clase: Entidad	
Atributo	**Tipo**
medio	string
cantMedios	int

Tabla 13: Descripción de la clase Medio.

Nombre de la clase: Entidad Colaboradora	
Tipo de clase: Entidad	
Atributo	**Tipo**
nombreEntidad	string

Tabla 14: Descripción de la clase Entidad Colaboradora.

2.5.3 Diagramas de Colaboración

En los diagramas de colaboración se muestran las interacciones entre objetos creando enlaces entre ellos y añadiendo mensajes entre ellos. El nombre de un mensaje denota el propósito del objeto invocante en la interacción con el objeto invocado. (4)

A continuación se muestra el diagrama de colaboración del CU Gestionar Plan de Actividades de Tratamiento.

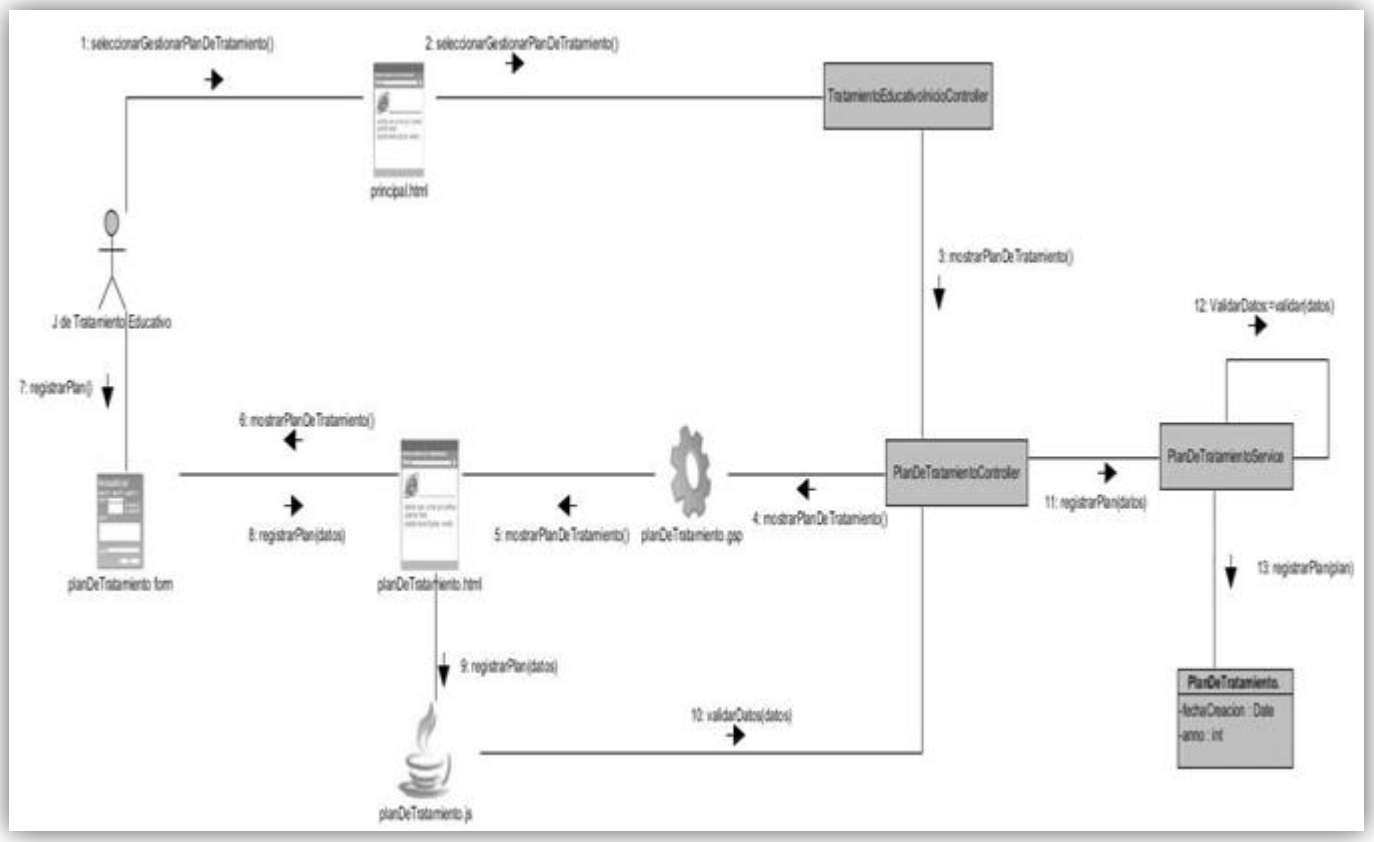

Figura 8: CU Gestionar Plan de Actividades de Tratamiento.

En esta figura se observa la secuencia de acciones que debe seguir un usuario para crear un Plan de Tratamiento.

Inicialmente el usuario selecciona en el menú Actividades Educativas la opción Plan de Tratamiento, en caso de que exista un plan de tratamiento para el año en curso, se mostrará en la interfaz una tabla con los datos del plan. Si el plan del año no ha sido creado se mostrarán los campos para introducir los datos del plan, una vez validados los datos del lado del cliente, mediante la utilización de JavaScript, estos se envían al controlador encargado de su manejo, que a su vez los envía hacia el servicio encargado de implementar la lógica concerniente a un plan de tratamiento, donde son validados nuevamente y se crea una instancia de la entidad **PlanDeTratamiento**, registrándose a continuación un nuevo Plan de Tratamiento.

2.6 Diseño de la Base de Datos

Un diagrama o modelo entidad-relación es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información así como sus interrelaciones y propiedades.

A continuación se muestra el modelo entidad-relación perteneciente al módulo Actividades Educativas del SIDEP.

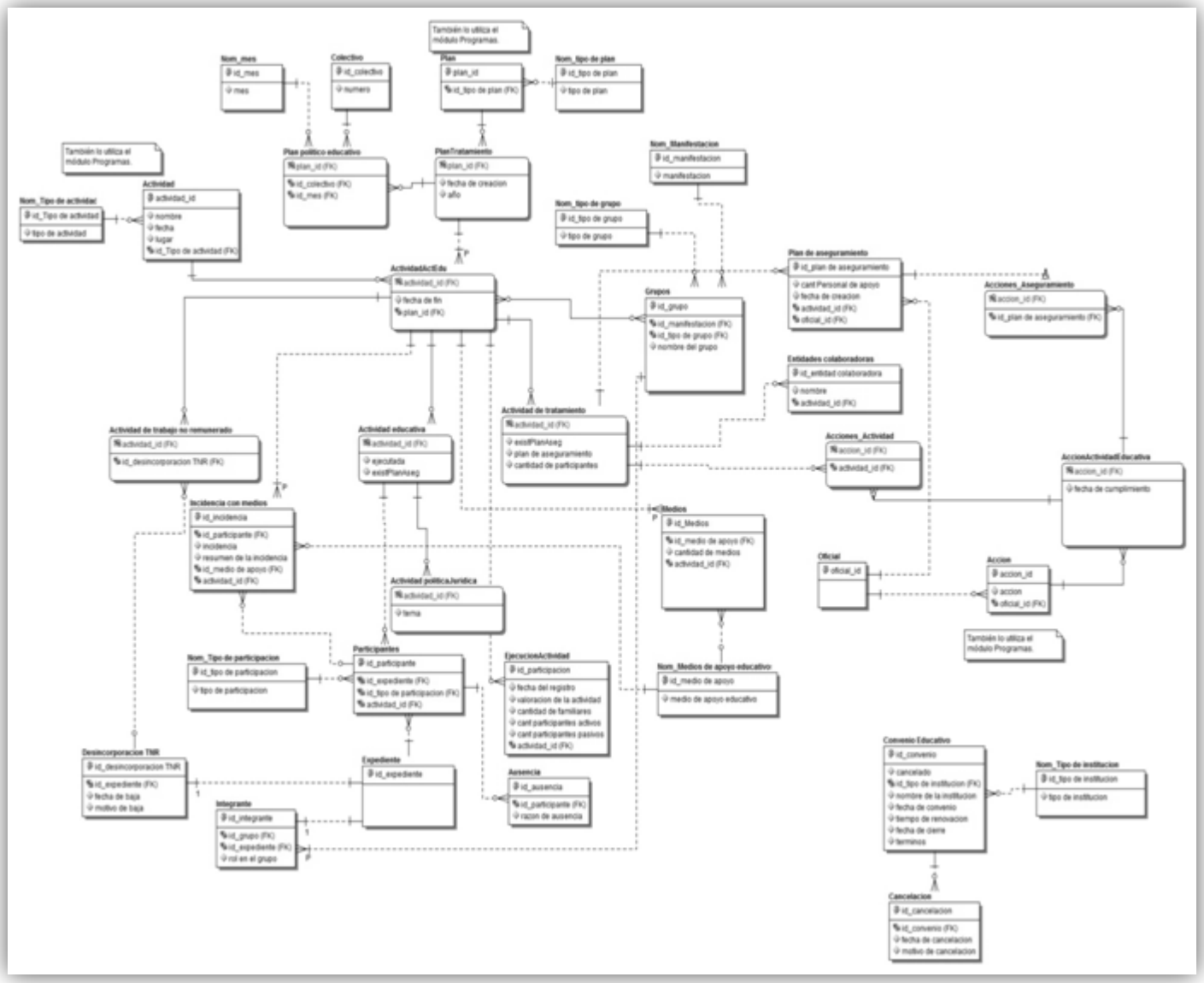

Figura 9: Modelo Entidad-Relación del módulo Actividades Educativas.

Como resultado de la realización del modelo entidad-relación se obtuvieron un total de 35 tablas. Un aspecto a resaltar en el modelo de datos es la herencia relacionada a las actividades, ya sea una actividad de tratamiento o educativa. Estas contienen la información referente a las distintas clasificaciones de las actividades. La entidad Plan de Tratamiento contiene los datos referentes al plan anual que realiza el Jefe de Tratamiento Educativo, de la cual hereda la entidad Plan Político Educativo que planifica el Educador Guía de forma mensual para el colectivo al cual atiende. Existe también la entidad AccionActividadEducativa que hereda de la entidad Acción, siendo esta última una entidad común con otro módulo. De la entidad AccionActividadEducativa heredan a su vez las entidades, AccionActividad y AccionAseguramiento, separando estas dos últimas para diferenciar cuando una acción corresponde a la planificación de un Plan de Aseguramiento ó a una actividad.

2.7 Descripción de las tablas

A continuación se describen las tablas fundamentales para el desarrollo del módulo Actividades Educativas.

Nombre de la tabla: Plan de Tratamiento		
Descripción: Almacena los datos del Plan de Tratamiento correspondientes a la unidad.		
Atributo	**Tipo**	**Descripción**
PLANTRAT_ID	NUMBER(19)	Identificador de un Plan de Tratamiento.
FECHA_CREACION	DATE	Guarda la fecha en que se crea el plan.
ANNO	NUMBER(10)	Guarda el año en que se confecciona el plan.
TIPO_PLAN_ID	NUMBER(19)	Identificador del tipo de plan.

Tabla 15: Descripción de la tabla Plan de Tratamiento.

Nombre de la tabla: Plan Político Educativo		
Descripción: Almacena los datos del Plan de Tratamiento correspondientes a la unidad.		
Atributo	**Tipo**	**Descripción**
PLANPOL_ID	NUMBER(19)	Identificador de un Plan Político Educativo.
COLECTIVO_ID	NUMBER(19)	Identificador del colectivo al que pertenece el plan.
MES	NUMBER(10)	Guarda el mes en que se confecciona el plan.

Tabla 16: Descripción de la tabla Plan Político Educativo.

Nombre de la tabla: Acciones

Descripción: Almacena los datos específicos de las acciones a desarrollar como parte de la actividad.		
Atributo	**Tipo**	**Descripción**
ACCION_ID	NUMBER(19)	Identificador de una acción.
NOMBRE_ACCION	VARCHAR2(255)	Guarda el nombre de la acción que se desarrollará en la actividad.
FECHA_CUMPLIMIENTO	DATE	Guarda la fecha en que debe dársele cumplimiento a la acción.
RESPONSABLE_ID	NUMBER(19)	Identificador del responsable de la acción.

Tabla 17: Descripción de la tabla Acción.

Nombre de la tabla: Actividad Educativa		
Descripción: Almacena los datos de una actividad educativa.		
Atributo	**Tipo**	**Descripción**
ACTEDUC_ID	NUMBER(19)	Identificador de una Actividad Educativa.
EJECUTADA	NUMBER(1)	Especifica si la actividad ha sido ejecutada o no.
PLAN_ID	NUMBER(19)	Identificador del plan al cual pertenece la actividad.
EXISTPASEG	NUMBER(1)	Especifica si la actividad lleva incluida un plan de aseguramiento.

Tabla 18: Descripción de la tabla Actividad Educativo.

Nombre de la tabla: Actividad de Tratamiento		
Descripción: Almacena los datos específicos de un actividad perteneciente al Plan de Tratamiento.		
Atributo	**Tipo**	**Descripción**
AT_ID	NUMBER(19)	Identificador de una Actividad de Tratamiento.
CANT_PARTICIPANTES	NUMBER(10)	Almacena la cantidad de participantes en la actividad.
EXISTPASEG	NUMBER(1)	Especifica si la actividad lleva incluida un plan de aseguramiento.
PLAN_ID	NUMBER(19)	Identificador del plan al cual pertenece la actividad.

Tabla 19: Descripción de la tabla Actividad de Tratamiento.

Nombre de la tabla: Grupo
Descripción: Almacena los datos específicos de los grupos artísticos o deportivos que participarán en una actividad.

Atributo	Tipo	Descripción
GRUPO_ID	NUMBER(19)	Identificador del grupo.
NOMBRE_GRUPO	VARCHAR2(255)	Guarda el nombre del grupo que participará en la actividad.
TIPO_GRUPOS_ID	NUMBER(19)	Especifica el tipo del grupo que participará en la actividad.
MANIFESTACION_ID	NUMBER(19)	Especifica la manifestación en la que se desarrolla el grupo que participará en la actividad.
UNIDAD	NUMBER(1)	Especifica si el grupo pertenece a una unidad.

Tabla 20: Descripción de la tabla Grupo.

2.8 Conclusiones Parciales

En este capítulo se realizó la descripción de la arquitectura n capas, basada en el patrón MVC, que se utilizará para el desarrollo de la aplicación. Se describieron también los diversos patrones de diseño utilizados: el patrón IoC, los patrones GRASP: Alta Cohesión, Bajo Acoplamiento, Experto, Controlador y Creador; así como el patrón Singleton, perteneciente a los patrones de diseño GOF. Se obtuvo el diseño del sistema para el módulo Actividades Educativas a través de los diagramas de clases del diseño y el modelo entidad relación perteneciente a la base de datos, describiendo las clases involucradas. El modelo de diseño realizado permitirá dar paso a la implementación del módulo.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

3 Introducción

En este último capítulo se presentan los diagramas correspondientes a la implementación del módulo Actividades Educativas, específicamente los diagramas de despliegue y de componentes. Además se realizará el diseño de los casos de prueba y se ejecutarán las pruebas de función para validar las funcionalidades del sistema.

3.1 Diagrama de Despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. (4)

A continuación se muestra el diagrama de despliegue.

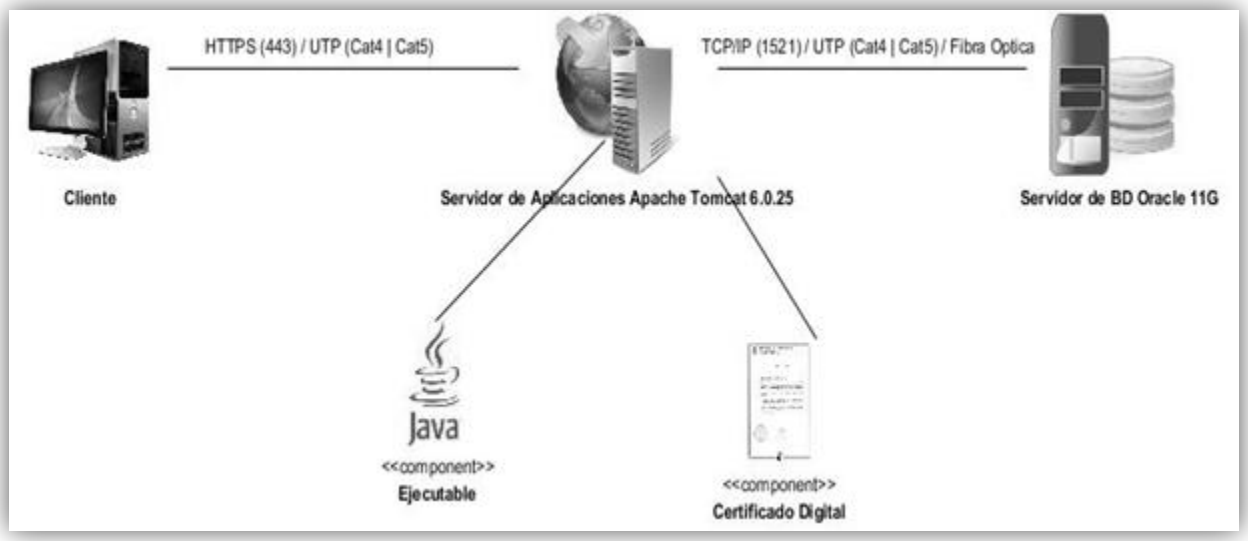

Figura 10: Diagrama de despliegue.

Para el despliegue del módulo el cliente utilizará las siguientes tecnologías:

Puestos de trabajo (PC usuario)

- RAM: 512 MB
- Navegador web Mozilla Firefox 3.6.*

Servidor de la aplicación

- Microprocesador: 4 núcleos, 3 GHz
- RAM: 4 GB
- Espacio necesario para la instalación: 250 MB
- Espacio libre: 250 GB
- JDK 1.6
- Apache Tomcat 6.0

Servidor de Bases de datos

- Microprocesador: 4 núcleos, 3 GHz
- RAM: 4 GB
- Espacio necesario para la instalación: 2 GB
- Espacio libre: 1 TB
- JDK 1.6
- Oracle 11g

3.2 Implementación

La implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros. (4)

3.3 Diagrama de componentes

Un diagrama de componentes muestra la estructura de los componentes, incluyendo clasificadores que especifican componentes, y artefactos que los implementan. También se pueden utilizar para mostrar la estructura de alto nivel del modelo de implementación en términos de subsistemas de implementación, y las relaciones entre elementos de implementación.

Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño. De ahí que los componentes tengan relaciones de trazas con los elementos del modelo que implementan. (4)

En la siguiente figura se muestra el diagrama de componentes que ilustra la relación del módulo Actividades Educativas con los restantes subsistemas y módulos.

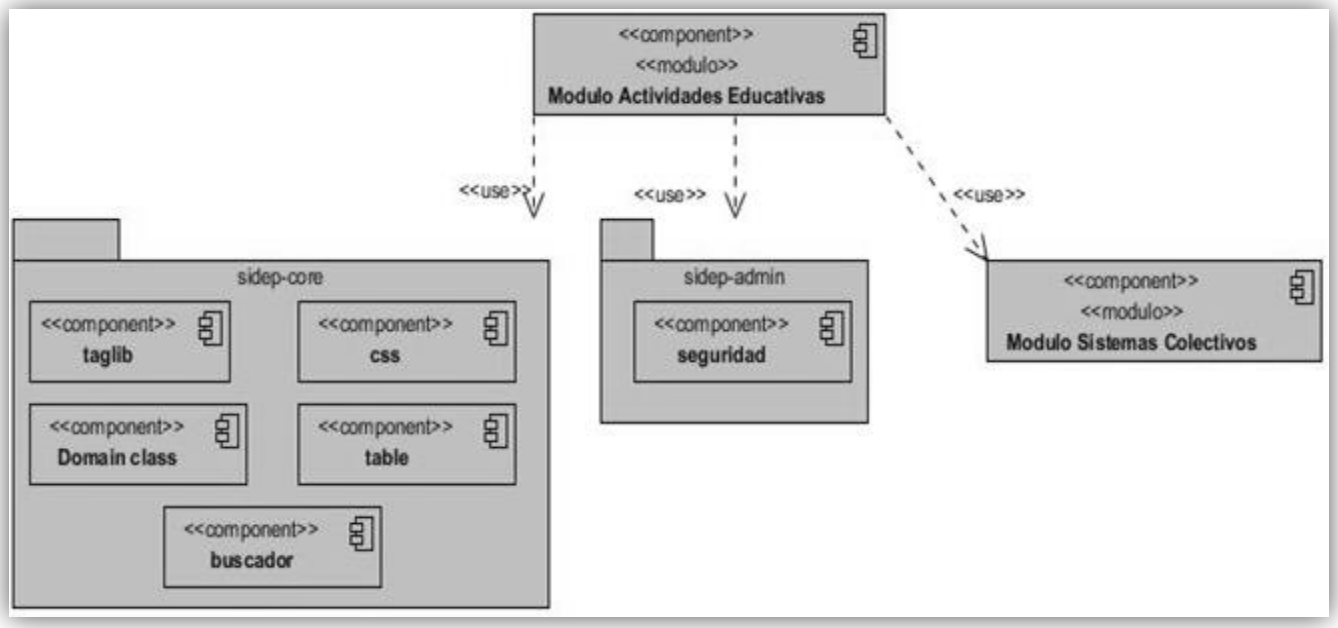

Figura 11: Diagrama de interacción del módulo con otros subsistemas.

Como se puede observar en la figura anterior el módulo Actividades Educativas está relacionado con el subsistemas **sidep-core**, pues en este se encuentran clases del dominio comunes para otros subsistemas que fueron necesarias utilizar para la realización del módulo. Se utiliza también un taglib[17] definido en el proyecto, y los archivos css, que se utilizan para la construcción de las vistas, así como un componente brindado para la construcción de las tablas y otro para los buscadores. Este subsistema brinda también la información referente al usuario que inicia sesión en el sistema, siendo necesaria esta información para el funcionamiento del módulo. La relación con el subsistema **sidep-admin** está dada por la importancia que este presenta al ser el encargado de manejar los temas de seguridad de la aplicación. La utilización del

[17] Librería de etiquetas

módulo **Sistemas Colectivos** fue necesaria para la obtención de los internos relacionados al colectivo de un determinado Educador Guía logueado en el sistema, siendo necesario para la conformación de los grupos de las distintas manifestaciones de los colectivos.

A continuación se detallan los componentes utilizados en el módulo Actividades Educativas.

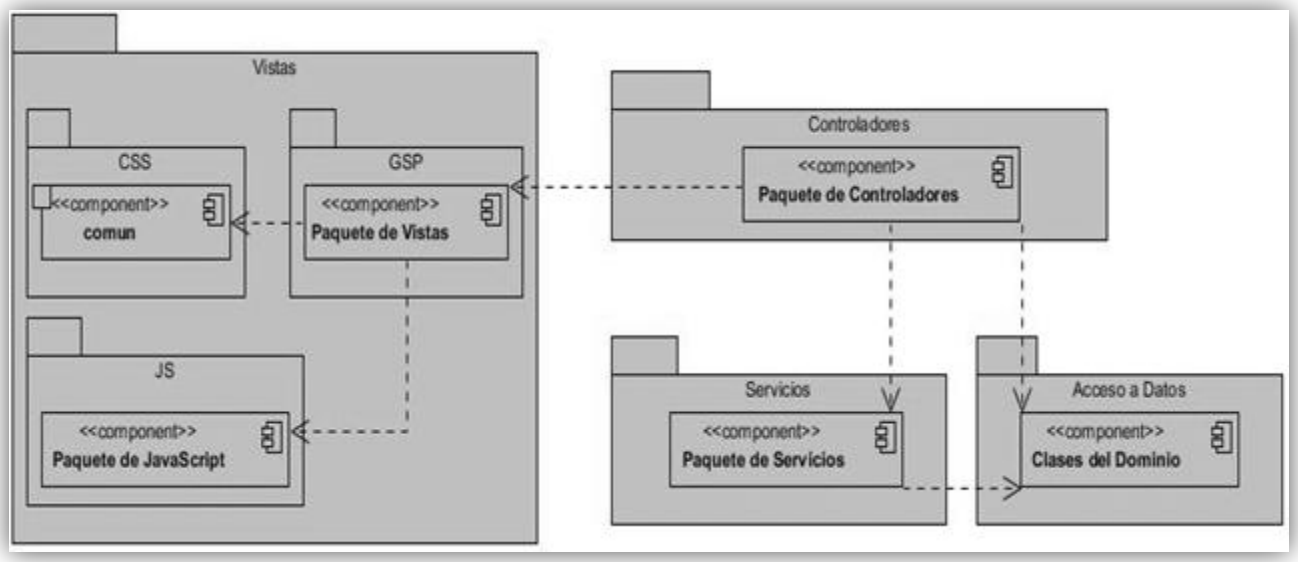

Figura 12: Diagrama de componentes del módulo.

La distribución de los componentes se realizó teniendo en cuenta la estructura de las capas presentes en la arquitectura del sistema:

- Capa de Presentación.
- Capa Controladora.
- Capa de Datos.
- Capa de Servicios.

3.4 Implementación de la capa de Acceso a Datos

El modelo de datos en una aplicación Grails está compuesto por las entidades del dominio. Grails utiliza GORM, un gestor de persistencia escrito en Groovy para controlar el ciclo de vida de las entidades.

3.4.1 Implementación de las entidades del dominio

En la mayoría de las aplicaciones las entidades del dominio solo poseen información que describe dichas entidades, sin relación con la implementación de la lógica del negocio, Grails sin embargo, fortalece su estructura haciendo mayor la dependencia de controladores y vistas. Las clases de dominio contendrán la validación de sus atributos mediante expresiones y restricciones propias de Grails, las que facilitan en gran medida el comportamiento de dichos datos tanto en las vistas como en los métodos a los que puedan estar vinculados en los controladores. Las clases de dominio, las cuales estarán contenidas en el paquete *Domain class*, contendrán a su vez las relaciones existentes entre ellas siguiendo la estructura del modelo de relaciones que propone Grails.

```
package sidep.tratamientoeducativo.domain.actividadeseducativas

class ActividadEducativa extends ActividadActEdu {
    boolean existPAseg
    boolean ejecutada
    static hasMany = [participantes: Participante, grupos: Grupo]
    static belongsTo = [plan: PlanPoliticoEducativo]
    static mapping = {
        table 'ACT_EDU'
        tablePerHierarchy false
        id column: 'actEduc_id'
    }

    static constraints = {
        participantes(nullable: true)
        grupos(nullable: true)
        plan(nullable: false)
    }
}
```

Figura 13: Clase del dominio que describe una Actividad educativa.

En la figura anterior se pueden apreciar diferentes aspectos que puede poseer una entidad del domino, en este caso la entidad Actividad Educativa, que hereda de la clase ActividadActEdu:

- La propiedad estática *hasMany* garantiza que una actividad educativa pueda tener varios grupos.
- La propiedad estática *belongsTo* establece por convenio que existe una relación de subordinación entre una actividad educativa y un plan político educativo. Esta propiedad permite además el

borrado en cascada, lo cual garantiza que al borrar un plan político educativo se borren las actividades educativas subordinadas a este.

- La propiedad estática *constraints* nos permite restringir los valores que pueden asignarse a cada entidad. Se utiliza también a la hora de crear el esquema de datos y para la generación de vistas mediante scaffolding.
- La propiedad estática *mapping* permite personalizar el mapeo de la entidad, definiendo estáticamente las propiedades que se mapearán a la base de datos. En este caso definiendo el nombre de la tabla mediante la propiedad *table* y el valor que tendrá el nombre de la columna id mediante la propiedad *column.*

3.4.2 Operaciones sobre el modelo de datos

Grails nos brinda a través de GORM un conjunto de funcionalidades y métodos dinámicos que facilitan la manipulación de las entidades del dominio.

Actualizaciones: todas las entidades poseen métodos de instancias que permiten su manipulación.

- save(): Este método permite insertar el registro en la base de datos o actualizarlo si ya existía. Es importante tener en cuenta que si queremos que los datos se envíen a la base de datos en el mismo momento en que se invoca el método save(), es necesario la utilización del argumento *flush: true.*

```
boolean addIncidencia(String nombre, String resumen, def listadoMedios, def listadoInternos) {
    def incidencia = new IncidenciaMedio()
    boolean respuesta = false
    incidencia.incidencia = nombre
    incidencia.resumenIncidencia = resumen
    incidencia.internos = buscarInternos(listadoInternos)
    incidencia.medios = buscarMedios(listadoMedios)
    incidencia.actividad = actividad
    if (incidencia.validate()) {
        respuesta = incidencia.save(flush: true)
        incidenciasAdd.add(incidencia)
    }
    return respuesta
}
```

Figura 14: Método para guardar la entidad Incidencia.

- delete(): Este método se utiliza para eliminar un registro. Al igual que save() podemos pasar el argumento flush a delete() para que la petición se ejecute de forma inmediata.

```
def eliminar(long id) {
    boolean respuesta = false
    respuesta = Grupo.get(id).delete()
    return respuesta
}
```

Figura 15: Método para eliminar una instancia de la entidad Grupo.

Bloqueo de datos

El bloqueo de datos es una de las funcionalidades que brinda el uso de GORM, este facilita el trabajo con los datos en caso de grandes volúmenes de tráfico. GORM brinda dos tipos de bloqueo de datos:

- Bloqueo optimista: GORM aplica por defecto este comportamiento. Esto significa que cada instancia tendrá un campo *version* que se incrementará cada vez que se modifiquen sus propiedades. Cada vez que hagamos una llamada al método save() sobre un objeto, se comparará el valor de la versión del objeto en memoria con el almacenado en la base de datos, y si no coinciden significará que algún otro proceso ha modificado el objeto después de que este fuera cargado. Esto hará un rollback de la transacción y lanzará una excepción. (16)
- Bloqueo pesimista: cuando se carga un objeto desde la base de datos, se solicitará a la base de datos el bloqueo de la fila correspondiente en la tabla, de forma que otras operaciones (incluso de lectura) sobre ella, tendrán que esperar a que termine de ejecutarse la operación actual. Para utilizar el bloqueo pesimista debemos invocar el método lock() que tienen todas las clases de entidad. (16)

Para la realización del módulo fue importante la utilización del bloqueo optimista al ser necesario permitir la modificación de gran cantidad de datos. Este tipo de bloqueo protege la corrupción de datos por modificaciones concurrentes.

Consultas

Entre las posibilidades que brinda GORM se encuentra la facilidad de realizar consultas a las tablas existentes en la base de datos de diferentes maneras:

- GORM inyecta una serie de métodos estáticos (de clase) en nuestras clases persistentes para hacer consultas sobre los datos almacenados en la tabla o tablas correspondientes. A continuación se muestran algunas formas de obtener instancias de nuestras clases de entidad.

```
def obtenerGrupo(long id) {
    Grupo.get(id)
}
```

Figura 16: Método estático utilizado para obtener un Grupo de la base de datos.

```
List ListadoGrupoColectivo(def params) {
    def items = []
    def grupos = GrupoDelColectivo.list(params)
    grupos.each {
        def map = [:]
        map.nombre_grupo = it.nombreGrupo
        map.tipo_grupo = it.tipoGrupos
        map.manifestacion = it.manifestacion
        map.colectivo = it.colectivo
        map.id = it.id
        items << map
    }
    return items
}
```

Figura 17: Método estático utilizado para obtener los objetos de la tabla GrupoDelColectivo.

Cuando no se conoce el identificador de la instancia o instancias que buscamos, necesitamos hacer consultas, que en GORM puede enfocarse de varias maneras.

- Dinamic fynders (localizadores dinámicos): los localizadores dinámicos son una de las ventajas que nos brinda GORM. Durante el desarrollo de la aplicación se utilizaron los métodos findBy o findAllBy, así como combinaciones de operadores y propiedades. La diferencia entre findBy y findAllBy es que el primero solo devolverá la primera instancia que cumpla con la condición de búsqueda, mientras que el segundo devolverá una lista con todos los resultados que correspondan. A pesar de las posibilidades que brindan estos localizadores dinámicos, presentan como limitante que solo se pueden realizar las consultas teniendo en cuenta 2 criterios de búsqueda.
- Criteria: se utilizan cuando se hace necesario hacer búsquedas avanzadas que incluyen muchos campos y comparaciones complejas como se muestra a continuación.

```
List buscarPlanPolitico(def params) {
    tratamiento = false
    planes = PlanTratamiento.createCriteria() list(params) {
        and {
            if (params.plan.tipoPlan) {
                tipoPlan {
                    eq("id", params.plan.tipoPlan.toString()?.toLong())
                }
            }
            if (params.plan.mes) {
                eq("mes", params.plan.mes.toInteger())
            }
            if (params.plan.anno) {
                eq("anno", params.plan.anno.toInteger())
            }
        }
    }
    return planes
}
```

Figura 18: Búsqueda de Plan Político utilizando varios criterios.

3.5 Implementación de la Capa de Presentación

3.5.1 Implementación de las vistas

Las vistas son responsables de hacer la interfaz de usuario y se implementan mediante archivos GSP, de los cuales se ha dicho anteriormente que es la tecnología de vistas que utiliza Grails y es una extensión de JSP que puede incluir código Groovy. Las GSP tienen extensión .gsp y están dentro del directorio *grails-app/Views* (Figura 3). Para introducir el código Groovy dentro de una página GSP se utilizan las etiquetas <% %>, así como <%=%> para evaluar las expresiones y los valores de salida. No es necesario importar bibliotecas de etiquetas para construirlas pues vienen incluidas en Grails.

```
<body>
<s:errors/>
<input type="hidden" name="appName" id="appName.id" value="${meta(name: 'app.name')}"/>
<form name="planpolitico.form" id="planpolitico.form.id"
      action="${createLink(controller: 'planPoliticoEducativo', action: 'siguiente')}" method="post" class="contorno">
    <input type="hidden" name="plan_seleccionado" id="plan_seleccionado" value="-1"/>
    <s:dateBox name="planpolitico.fecha" id="planpolitico.fecha"/>
    <s:input name="planpolitico.anno" id="planpolitico.anno" on="true" max="4"/>
    <s:months name="planpolitico.mes" id="planpolitico.mes"/>
    <s:clear/>
    <s:buttonBar>
        <s:button name="planpolitico.siguiente" id="planpolitico.siguiente" label="Siguiente"/>
        <s:button name="cancelarplanpolitico" id="cancelarplanpolitico" label="Cancelar"/>
    </s:buttonBar>
    <s:clear/>
    <input type="hidden" name="actPlan" id="actPlan" value="-1"/>
    <s:table id="planpolitico.tabla" action="listFromService" bean="planPoliticoDeAnnoService" update="true"
             delete="false" label="Plan Politico"/>
    <s:clear/>
    <s:clear/>
</form>
```

Figura 19: Código de la vista Plan Político Educativo.

En nuestro caso se ha utilizado una librería de etiquetas definida por el equipo de arquitectura del proyecto, que ha permitido simplificar la lógica de presentación. Por ejemplo:

- La etiqueta *s:dateBox* muestra un campo que permite seleccionar una fecha cualquiera.
- La etiqueta *s:months* muestra select desplegable que permite seleccionar un mes del año.
- La etiqueta *s:input* muestra un campo de texto que permite mediante algunas propiedades definidas especificar si el usuario puede entrar una cadena de caracteres ó un número, siendo la propiedad *on* para el número y *ol* para la cadena de caracteres.

Estas etiquetas también dan soporte a la internacionalización del lenguaje.

Las vistas son las encargadas a su vez de invocar las llamadas a las funciones de JavaScript las cuales van a estar contenidas en los archivos JS.

```
<script type="text/javascript"
src="${resource(dir: 'js/sidep/tratamientoeducativo/actividadeseducativas/planespasados', file: 'PlanesPasados.js')}">
</script>
<script type="text/javascript">
    //<![CDATA[
    dojo.addOnLoad(function () {
        sidep.tratamientoeducativo.actividadeseducativas.planespasados.planesPasados =
                new sidep.tratamientoeducativo.actividadeseducativas.planespasados.PlanesPasados();
    });
    // ]]>
</script>
```

Figura 20: Llamada al archivo JS PlanesPasados.

3.5.2 Internacionalización del lenguaje

Internacionalización (i18n) es la operación por la cual un programa o un conjunto de programas son capaces de soportar múltiples lenguajes.

Grails brinda soporte para internacionalización mediante el uso de archivos de recursos almacenados en la carpeta *Grails-app/i18n*. Estos archivos de recursos son ficheros de extensión *properties* de Java en los que se guarda las distintas versiones de cada mensaje en todos los idiomas soportados por nuestra aplicación. Cada idioma está definido en un fichero distinto que incluye en su nombre el *Locale* del idioma, por ejemplo:

- *messages_es_AR.properties* para los mensajes en español de Argentina
- *messages_en.properties* para los mensajes en inglés.
- *messages_es.properties* para los mensajes en español.

messages_es.properties

```
# Dialogos
dialog.habilidades.title = Habilidades especiales
dialog.nombres.title = Otros nombres
dialog.alias.title = Alias
dialog.ciudadanias.title = Ciudadanías
dialog.idiomas.title = Idiomas que conoce
dialog.oficios.title = Oficios
dialog.profesiones.title = Profesiones

# Botones
button.aceptar.label = Aceptar
button.adicionar.label = Adicionar
button.cancelar.label = Cancelar
button.atras.label = Atrás
button.siguiente.label = Siguiente
button.buscar.label = Buscar
button.nuevo.label = Nuevo
button.actualizar.label = Actualizar
button.verInternos.label = Ver internos

# Table
table.page.label = Página
table.of.label = de
table.displaying.label = Mostrando
domain.radio.label =
domain.ficha.label = Ficha
```

Figura 21: Fichero para los mensajes en español.

3.6 Implementación de la Capa de Control

Los controladores son los responsables de gestionar y coordinar el flujo lógico de la aplicación mediante la recepción de las acciones del usuario. Pueden interactuar directamente con una clase de dominio para realizar una operación de CRUD[18], re-direccionar al usuario a un GSP diferente, delegar una acción a un actor diferente (otro controlador o una clase de servicio), así como preparar y enviar la respuesta de nuevo a la vista. Siempre creando un nuevo controlador para cada solicitud. Se encontrarán ubicados en el paquete *grails-appIControllers* (Figura 4) manejando todo el flujo de trabajo de la capa Web Layer junto a las vistas.

18 Es un acrónimo para las operaciones básicas de Creación (Create), Lectura (Read), Actualización (Update) y Borrado (Delete).

```
package sidep.tratamientoeducativo.controller.actividadeseducativas.planpoliticoeducativo

import ...

class ActividadEducativaController {
    def grupoColectivoService
    def mediosActEduService
    def planPoliticoDeAnnoService
    def participanteActEduService
    def actividadEducativaService
    def index = {
        render(view: "/actividadeseducativas/planpoliticoeducativo/actividadeducativa")
    }
    def eliminarAct = {
        actividadEducativaService.delAct(params.id.toLong())
        render "true"
    }
    def cancelAddAct = {
        mediosActEduService.delAllItems()
        participanteActEduService.delAllItems()
        render "true"
    }
    def listarGrupos = {
        def grupos = [identifier: "id", items: grupoColectivoService.getGrupos()]
        render grupos as JSON
    }
    def guardarActividad = {...}
```

Figura 22: Controlador que gestiona una Actividad Educativa.

En la figura anterior se muestra el controlador destinado a gestionar las operaciones ligadas a una actividad educativa, en este se pone de evidencia el uso del patrón Singleton con la inyección de dependencias al declarar, por ejemplo: la variable *actividadEducativaService,* que crea una instancia del servicio ActividadEducativaService que permite acceder a los métodos definidos en este. Se definen diversas acciones que están destinadas a gestionar una actividad educativa:

- *index*: Esta acción se encarga de mostrar la vista que maneja los datos de una actividad educativa.
- *eliminarAct*: Es la encargada de eliminar una actividad educativa, se puede observar que el uso de la palabra reservada *return* es opcional en Grails.

- *listarGrupos:* Se encarga de listar los grupos pertenecientes a una actividad educativa, nótese el uso de la sentencia *render grupos as JSON* que permite devolver el resultado en este formato, usando un convertidor de Grails que es capaz de realizar esta acción.

3.7 Implementación de la Capa de Servicios

La responsabilidad de la implementación de la lógica de negocio en Grails es responsabilidad de los servicios. Un servicio es una clase de extensión *service* que se encuentra situado en la carpeta *grails-app/services* (Figura 5). En estos servicios se implementan las funcionalidades que respaldan las reglas del negocio del módulo, recibiendo los datos que se envían desde las vistas, consultando datos ya existentes, generando reportes y manejando los resultados.

```
class PlanDeTratamientoService implements TableService {
    static scope = "session"
    static transactional = true
    def PlanTratamiento planT
    def actPlan = -1

    @Override
    public List getProps(def params) {...}

    @Override
    public List getItems(Object params) {...}

    @Override
    public int getTotal(def params) {...}

    List listadoPlanes(def params) {...}

    boolean existePlanAnno() {...}

    def addPlan(Date fecha, int anno) {...}

    def finalizarPlan(List<ActividadTratamiento> listActividades) {...}

    def dellAllItems() {...}

    def finalizarActualizarPlan(List<ActividadTratamiento> listActividades) {...}
}
```

Figura 23: Servicio que gestiona las operaciones sobre un Plan de Tratamiento.

En este caso el servicio implementa de una interfaz llamada TableService que implementa los métodos getProps (), getItems () y getTotal (). En su conjunto estos métodos permiten la visualización de los elementos de cualquier entidad en una tabla.

- getProps (): en este método se definen los nombres de las columnas que se van a mostrar en la tabla. En la siguiente figura se muestra un ejemplo de la implementación de este método.

```
List getProps(def params) {
    ["nombre_grupo", "tipo_grupo", "manifestacion"]
}
```

Figura 24: Implementación del método getProps () del servicio GrupoUnidadService.

- getItems (): este método devuelve un listado de los elementos que se mostrarán en la tabla, creando por cada elemento un mapa donde se guardarán los distintos atributos del elemento. Las llaves del mapa deberán coincidir con las propiedades que se definen en el método getProps ().Se puede observar un ejemplo de la implementación de este método en la siguiente figura.

```
List getItems(Object params) {
    items = []
    def grupos = Grupo.findAllByUnidad(true, params)
    grupos.each {
        if (it.unidad) {
            def map = [:]
            map.nombre_grupo = it.nombreGrupo
            map.tipo_grupo = it.tipoGrupos
            map.manifestacion = it.manifestacion
            map.id = it.id
            map.version = it.version
            items << map
        }
    }
    return items
}
```

Figura 25: Implementación del método getItems () del servicio GrupoUnidadService.

- getTotal (): este método devuelve la cantidad total de elementos que se mostrarán en la tabla. Un ejemplo de esto se evidencia en la siguiente figura.

```
int getTotal(def params) {
    return Grupo.countByUnidad(true)
}
```

Figura 26: Implementación del método getTotal () del servicio GrupoUnidadService.

3.7.1 Transacciones a Nivel de Negocio

El uso de GORM asegura que no tengamos que hacer nada para asegurar la transacción en la implementación del negocio, pues por defecto todos los métodos públicos en los servicios son transaccionales. Grails implementa mecanismos de Spring AOP (Programación Orientada a Aspectos), cada uno de los cuales está envuelto por un proxy creado utilizando *TransactionProxyFactoryBean* de Spring, por lo que cuando se hace referencia a un servicio en realidad se hace referencia a ese proxy. Las transacciones son controladas por un objeto de tipo *transactionManager,* que es por defecto una instancia de *HibernateTransactionManager.* (18)

Grails también permite especificar si un servicio es transaccional o no, mediante la declaración de una variable booleana llamada *transactional*. Si esta variable tiene valor *true,* entonces todas las llamadas a métodos de nuestros servicios se encerrarán en una transacción, produciéndose un *rollback* automático si durante la ejecución se lanzara una excepción.

```
class PlanDeTratamientoService implements TableService {
    static scope = "session"
    static transactional = true
    def PlanTratamiento planT
    def actPlan = -1
```

Figura 27: Declaración de un servicio transaccional.

3.8 Seguridad

La seguridad en el módulo Actividades Educativas se logró mediante la utilización de Spring Security, la cual es manejada por el subsistema **sidep-admin.** Spring proporciona servicios integrales de seguridad para la plataforma Java EE. Sus principales áreas de seguridad son la autenticación y la autorización. La autenticación es el proceso mediante el cual se establece si un usuario, aplicación o sistema que pueda realizar una acción en la aplicación, es quien dice ser. La autenticación es cifrada, lo cual significa que la

contraseña es cifrada, se utiliza el SHA[19] (Secure Hash Algorithm) con una longitud de salida de 256 bits. La autorización se refiere al proceso de decidir si un usuario está autorizado a realizar una acción dentro de la aplicación. Para llegar al punto donde se necesita una decisión de autorización, la identidad del usuario ya ha sido establecida por el proceso de autenticación. (19)

En la autorización se utilizó el modelo de Control de Acceso Basado en Roles (RBAC por sus siglas en inglés). Los derechos de acceso se agrupan por rol, y el acceso se restringe a los usuarios que tienen autorizado asumir dicho rol. Para lograr la autorización se asignó un código a cada funcionalidad, se crearon los roles, y a estos se les asignaron las responsabilidades que cada cual podrá realizar. Por último, se utilizó una notación de seguridad la cual permite especificar en los controladores las funcionalidades a las cuales pertenecen, esto se puede hacer tanto a nivel de controlador como a nivel de acción. Con este mecanismo se logra que el usuario tenga acceso a las funcionalidades que su rol le permite realizar sobre la aplicación. Esta notación es brindada por el subsistema **sidep-admin**, pudiendo observarse en la siguiente figura.

```
package sidep.tratamientoeducativo.controller.actividadeseducativas.ejecucionactividad

import sidep.admin.seguridad.Security

@Security("registrarEjecucionActEdu")
class IndexEjecActController {
    def ejecucionActividadService
    def participantesPorActividadService
    def incidenciaPorActividadService
    def ausenteActEduService
    def index = {
        ausenteActEduService.delAllItems()
        participantesPorActividadService.delAllItems()
        incidenciaPorActividadService.delAllItems()
        render view: "/actividadeseducativas/ejecucionactividad/index"
    }
    def irEjecutarAct={
        def map=[:]
        map.id_actividad=params.act_seleccionada.toLong()
        redirect(controller: 'ejecucionActividad',action: 'index',params: map)
    }

}
```

Figura 28: Notación de seguridad.

[19] Algoritmo Ash Seguro

3.9 Validación del diseño de la solución

Cuando se mide la calidad de un producto de software es importante aplicar métricas que arrojen resultados útiles para la medición.

Los objetivos principales de las métricas orientadas a objetos son:

- Comprender mejor la calidad del producto.
- Estimar la efectividad del proceso.
- Mejorar la calidad del trabajo realizado en el nivel del proyecto.

Existen un conjunto de métricas de diseño orientado a objetos de las cuales se pondrán en práctica: Tamaño de clase, Niveles de árbol de herencia y Número de hijos.

Métrica Tamaño de clase (TC)

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

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

(20)

Estos dos valores son sumados de acuerdo a la clase que se analiza y el resultado es comparado en una tabla para determinar el TC de cada clase. (20)

A continuación se muestra la tabla 21, la cual incluye la clasificación de TC que puede tener una clase al obtener un valor de umbral, donde **n** es el valor que se obtiene resultado de la suma de los atributos y las operaciones.

Tamaño	Valores de los umbrales
Pequeño	n<=20
Medio	20 < n ≤ 30
Grande	n >30

Tabla 21: Umbrales para el tamaño de clases.

A continuación se muestran los resultados obtenidos a partir de la aplicación de la métrica tamaño de clase.

No	Clases	Cantidad de operaciones	Cantidad de atributos	Tamaño
1	AccionesActService	6	3	Pequeño
2	AccionesAseguramientoService	5	4	Pequeño
3	ActEduIncidenciaMediosService	1	2	Pequeño
4	ActividadEducativaService	10	4	Pequeño
5	ActividadParaAsegurarService	1	0	Pequeño
6	ActividadTratamientoService	10	6	Pequeño
7	AusenteActEduService	2	1	Pequeño
8	BuscarGrupoColectivoService	3	3	Pequeño
9	BuscarIntegrantesColectivoService	1	3	Pequeño
10	ConsultarActividadesPorPlanService	2	6	Pequeño
11	ConvenioEducativoService	3	2	Pequeño
12	DesincorporarTNRService	2	3	Pequeño
13	EjecucionActividadService	3	5	Pequeño
14	GrupoColectivoService	9	4	Pequeño
15	GrupoUnidadService	6	4	Pequeño
16	IncidenciaPorActividadService	3	1	Pequeño
17	IncidenciasService	5	2	Pequeño
18	IntegranteDeGrupoUnidadService	6	2	Pequeño
19	IntegrantePorGrupoColectivoService	3	1	Pequeño
20	IntegranteService	6	5	Pequeño
21	MedioAsegService	5	1	Pequeño
22	MediosActEduService	7	1	Pequeño
23	MediosActTratService	6	2	Pequeño
24	ParticipanteActEduService	8	5	Pequeño
25	ParticipantesPorActividadService	6	4	Pequeño
26	PlanDeAseguramientoService	4	3	Pequeño
27	PlanDeTratamientoService	6	1	Pequeño
28	PlanesPasadosService	5	1	Pequeño
29	PlanPoliticoDeAnnoService	4	2	Pequeño

Tabla 22: Representación del tamaño de las clases del diseño.

Se trabajo con un total de 29 clases con un promedio de 2,79 atributos por clases y 4,76 de operaciones, lo que arrojó como resultado los siguientes datos:

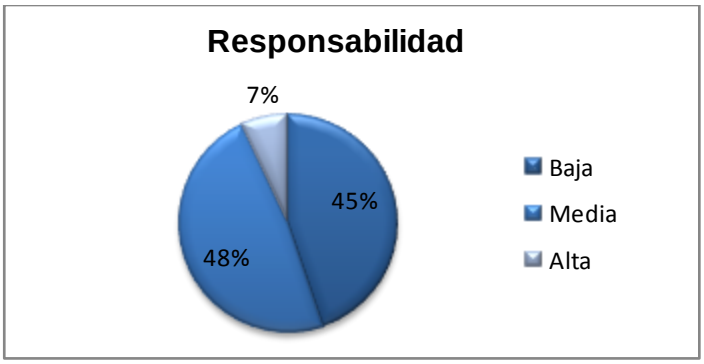

Figura 29: Responsabilidad de las clases.

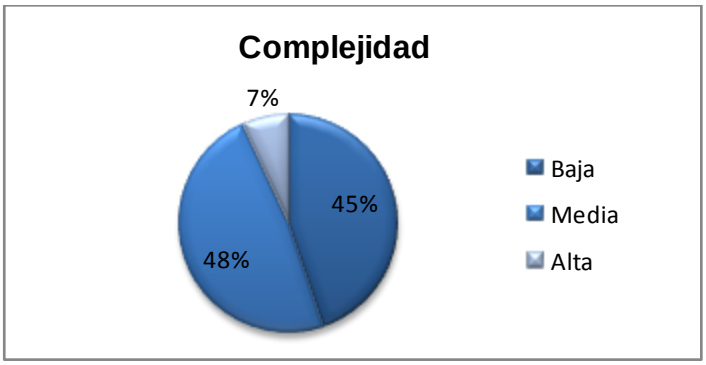

Figura 30: Complejidad de las clases.

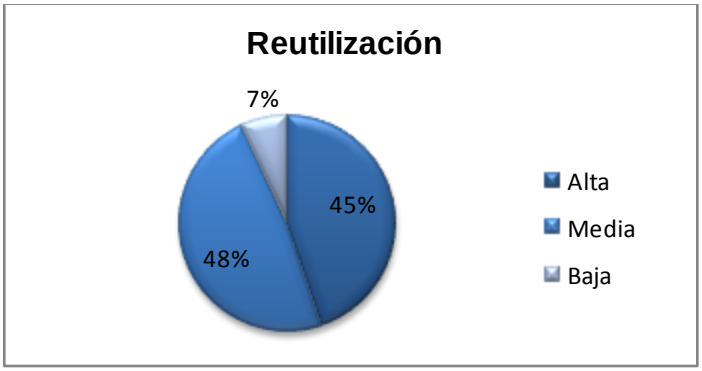

Figura 31: Reutilización de las clases.

Luego de la aplicación de esta métrica se obtuvo que el total de clases analizadas son pequeñas. Los resultados muestran que las mismas no tienen grandes responsabilidades, permitiendo la reutilización de

las clases, mostrando que el sistema tiene una implementación a nivel medio. Por tanto, los resultados obtenidos de la métrica son positivos.

Métrica de niveles de árbol de herencia.

La métrica de árbol de herencia de una clase A es su profundidad en el árbol de herencia. Si A se encuentra en situación de herencia múltiple la longitud máxima hasta la raíz será la profundidad de su herencia. (21)

Como resultado del análisis de la métrica, aplicada sobre 22 clases, se obtuvieron los siguientes resultados:

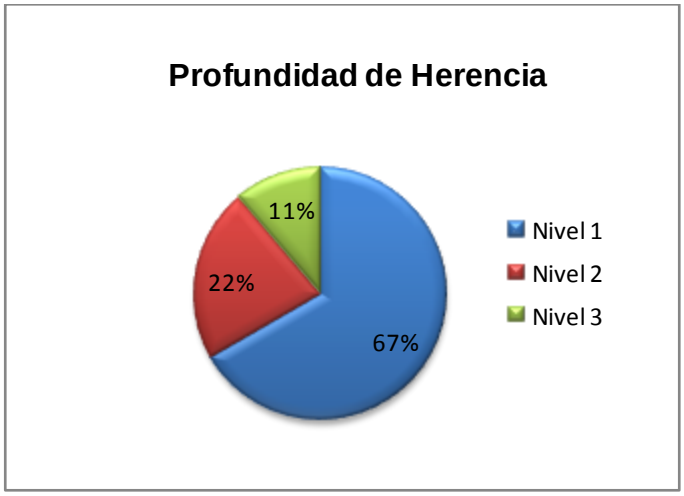

Figura 32: Niveles de profundidad de la herencia.

El resultado de esta métrica muestra que el máximo nivel de herencia presente en el diseño es 3. Por lo que se concluye que el diseño evita la herencia de muchos métodos por parte de las clases inferiores, y se disminuye su complejidad.

Métrica de número de hijos.

La métrica de número de hijos de una clase es el número de subclases inmediatamente subordinadas a una clase en la jerarquía. (21)

Para la realización de esta métrica se utilizó una escala de valores de 1 a 5 para definir las diferentes categorías asociadas a la medición de la abstracción y cohesión de las clases.

En las siguientes tablas se muestran los umbrales para la medición de la abstracción y cohesión de las clases.

Categoría	Criterio
Indefinida	> 5
Afectada	Entre 2 y 5
Definida	<= 2

Tabla 23: Umbrales para la medición de abstracción de las clases.

Categoría	Criterio
Baja	> 5
Media	Entre 2 y 5
Alta	<= 2

Tabla 24: Umbrales para la medición de la cohesión de las clases.

Para un total de 22 clases analizadas se obtuvieron los siguientes datos:

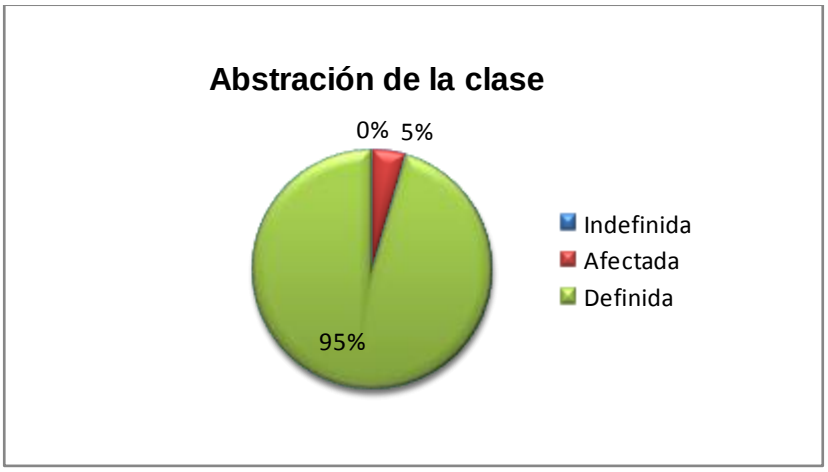

Figura 33: Abstracción de las clases.

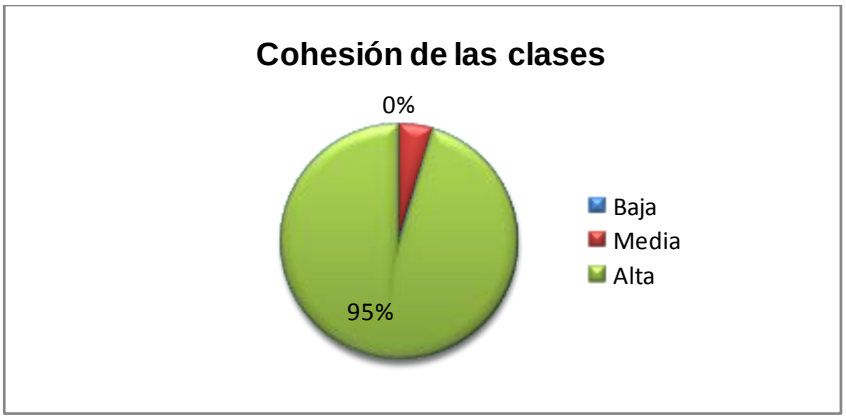

Figura 34: Cohesión de las clases.

Los resultados obtenidos de la realización de esta métrica muestran que el diseño realizado presenta abstracción entre las clases que permite la alta cohesión, teniendo en cuanta que se utilizó el patrón Alta Cohesión.

3.10 Pruebas

Las pruebas de software son los procesos que permiten verificar y revelar la calidad de un producto de software. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un sistema informático. Básicamente es una fase en el desarrollo de software, consistente en probar las aplicaciones construidas. Para determinar el nivel de calidad se deben efectuar unas medidas o pruebas que permitan comprobar el grado de cumplimiento respecto de las especificaciones iniciales del sistema. (20)

3.10.1 Tipo de Prueba

Dentro de los tipos de prueba que existen se encuentran las pruebas de función. Este tipo de prueba tiene como objetivo probar la habilidad del software para realizar el trabajo deseado. (22)

Metas:

- Verificar el procesamiento, recuperación e implementación adecuada de las reglas del negocio.
- Verificar la apropiada aceptación de datos.

3.10.2 Niveles de Prueba

La prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Se distinguen los siguientes Niveles de Pruebas: (22)

- Pruebas de desarrollador.
- Prueba Independiente.
- Pruebas Unitarias.
- Pruebas de Integración.
- Prueba del Sistema.
- Pruebas de Aceptación.

Al módulo Actividades Educativas se le aplicarán las pruebas de desarrollador.

La prueba de desarrollador indica los aspectos de diseño e implementación de las pruebas más adecuadas que debe llevar a cabo el equipo de desarrolladores. En la mayoría de los casos, la ejecución de la prueba se produce inicialmente con el grupo de pruebas del desarrollador que la diseñó e implementó; aunque es recomendable que los desarrolladores creen las pruebas de forma que estén disponibles para que las ejecuten grupos de pruebas independientes. (23)

Dentro de este nivel de prueba se le aplicará el método de prueba de caja negra.

3.10.3 Métodos de pruebas

- **Pruebas de Caja Negra**

Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo fundamentalmente del sistema sin tener mucho en cuenta la estructura interna del software. (24)

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de Caja Blanca.

Estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.

- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

3.10.4 Técnica de prueba

- **Casos de Prueba**

Una herramienta necesaria para la ejecución de las pruebas de función son los casos de prueba, que se definen como un conjunto de entradas con datos de prueba, unas condiciones de ejecución, y unos resultados esperados con el propósito de identificar y comunicar las condiciones que se llevarán a cabo en la prueba. Los casos de prueba son necesarios para verificar la aplicación exitosa y aceptable de los requisitos del producto. (23)

3.11 Resultados de la aplicación de las Pruebas de Caja Negra

A continuación se muestran los resultados de las pruebas de caja negra aplicadas al módulo Actividades Educativas.

No	Ubicación	No Conformidad	Estado	Criticidad
1	Registrar Actividad de Tratamiento/ modificar accion	Al seleccionar modificar acción no se muestra el formulario con los datos a modificar.	Resuelta	Baja
2	Registrar Actividad de Tratamiento/ insertar medio	El campo cantidad de medio permite entrar un número muy grande.	Resuelta	Baja
3	Plan politico/ registrar plan	El botón cancelar no realiza ninguna acción.	Resuelta	Media
4	Plan de tratamiento/modificar plan	Muestra un mensaje de alerta al seleccionar modificar plan.	Resuelta	Baja
5	Plan político/ registrar plan	Muestra el mensaje “Los campos en rojos son requeridos” y no marca los campos.	Resuelta	Baja
6	Plan de tratamiento/modificar plan	Muestra un error proveniente del JavaScript relacionado con un componente no definido.	Resuelta	Baja
7	Grupo del colectivo/eliminar	El botón eliminar en la tabla de grupos del colectivo no funciona.	Resuelta	Alta
8	Convenio Educativo/gestionar convenio	Los nombres de la columna en la tabla de convenios no están internacionalizados.	Resuelta	Baja
9	Consultar planes pasados/buscar planes	Error de java cuando se busca plan de tratamiento.	Resuelta	Alta

Tabla 25: Tabla de no conformidades del módulo Actividades Educativas

3.12 Conclusiones parciales

En este capítulo se presentaron los elementos más significativos de la implementación del módulo Actividades Educativa, como la implementación de las capas: de control, de servicios, de acceso a datos y de presentación. Además de tener en cuenta los elementos de seguridad brindados por el subsistema sidep-admin, mediante la utilización de Spring Security y el modelo de autorización utilizado por el proyecto. Se validó el diseño de la propuesta de solución mediante la utilización de métricas de diseño orientado a objetos: la métrica de tamaño de clases, niveles de árbol de herencia y número de hijos. Se realizó la validación de las funcionalidades del módulo mediante la utilización de las pruebas de función llevadas a cabo por el equipo de desarrolladores. Las cuales dieron como resultado un conjunto de no conformidades que fueron resueltas durante el proceso de desarrollo.

CONCLUSIONES GENERALES

Los objetivos propuestos para este trabajo fueron plenamente cumplidos a través de las actividades realizadas.

- Se analizó el sistema SACORE de la República de Cuba y el SIGEP de la República de Venezuela, mostrando que éstos no se ajustan a las necesidades del sistema penitenciario cubano, por lo que fue necesario implementar un nuevo sistema.
- Se realizó un estudio de las tecnologías y herramientas a utilizar en el diseño e implementación del módulo así como del análisis de los requisitos de software y del modelo de negocio correspondientes.
- Se diseñó el módulo Actividades Educativas, con el cual se obtuvo el Modelo de Diseño y en particular los Diagramas de Clases del Diseño y Diagramas de Interacción para cada uno de los casos de uso identificados.
- Se implementaron todas las funcionalidades definidas para el módulo Actividades Educativas durante la fase de inicio del proyecto con la utilización de las herramientas y tecnologías definidas, lográndose una aplicación funcional integrada al SIDEP.
- Se validaron las funcionalidades de la solución propuesta con la realización de las pruebas de función, demostrándose que cada una de las funcionalidades responde apropiadamente a los requisitos funcionales definidos.

RECOMENDACIONES

Apoyándonos en la investigación realizada y en los resultados obtenidos durante la elaboración de este trabajo, se recomienda lo siguiente:

- Realizar pruebas de aceptación con el cliente.
- Desarrollar el manual de usuario del módulo implementado para capacitar al personal que lo utilizará.
- Probar el rendimiento del módulo con grandes volúmenes de datos, pues actualmente está probado para una pequeña cantidad de datos que no pone al límite el rendimiento de la solución propuesta.

REFERENCIA BIBLIOGRÁFICA

1. **MINREX.** Sitio del Ministerio de Relaciones Exteriores de Cuba. [Online] http://www.cubaminrex.cu/CDH/61cdh/Derechos%20Humanos%20en%20Cuba/Derechos%20Humanos%20en%20Cuba_Sistema%20penintenciario.htm.

2. **Batista, Yanet del Risco.** *Funcionamiento de las actividades educativas en el SACORE.* 2012.

3. **ALBET, S.A.** *Manual de usuario Subsistema Atención Integral (Módulos Cultura, Deporte, Evaluaciones Técnicas).*

4. **Ivar JACOBSON, Grady RUMBAUGH, James BOOCH.** *El Proceso Unificado de Desarrollo de Software.* Ciudad Habana : s.n., 2002.

5. **Camy, Lázaro Issy.** *JavaScript.* 2002.

6. **Cupertino, J.R.** *El Lenguaje Unificado de Modelado.* California : s.n., 1998.

7. **Russell, Mathew A.** *Dojo The Definitive Guide.* 2008.

8. **Abián, Miguel Ángel.** *Java2EE vs Net.*

9. Groovy. A dynamic language for the Java platform. *Groovy.* [Online] SpringSource and the Groovy Community. [Cited: mayo 10, 2012.] http://www.groovy.codehaus.org/.

10. **Oracle.** Oracle. [Online] http://www.oracle.com/technetwork/database/enterprise-edition/documentation/database-093888.html.

11. *DosIdeas.* [En línea] http://www.dosideas.com/wiki/NetBeans.

12. **Embarcadero Technologies.** Database Tools and Developer Software. [Online] http://www.embarcadero.com.

13. **The Apache Software Foundation.** Apache Tomcat. [Online] http://www.tomcat.apache.org/.

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

15. **Din, Yisel Astiazarain.** *Especificación de casos de uso (Act. Educ.).*

16. **Calahorro, Nacho Brito.** *Manual de desarrollo web con Grails.* s.l. : Graeme Rocher, 2009. v1.0.4.

17. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 1999.

18. **Smith, Glen and Ledbrook, Peter.** *"Grails in Action".* s.l. : Manning Publications Co, 2009.

19. SpringSource. *SpringSource.* [Online] http://www.static.springsource.org/spring-security/site/docs/3.1.x/reference/introduction.html.

20. **Pressman, Roger S.** *Ingenieria del Software. Un enfoque práctico.*

21. **Arregui, Juan José Olmedilla.** *Revisión Sistemática de Métricas de Diseño Orientado a Objetos.* 2005.

22. **Natalia Juristo, Ana M. Moreno, Sira Vegas.** *Técnicas de Evaluación de Software.* 2005.

23. **Peña., Ing. Yadira Machado.** *Estrategia de Pruebas de función.*

24. **Pressman, Roger.** Cap 14 Técnicas de prueba.