Universidad de las Ciencias Informáticas

Facultad 4

Diseño e implementación de los módulos Educación y Trabajo del Sistema de Gestión Penitenciaria de la República Bolivariana de Venezuela

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

Autor: Angel Gustavo Suárez Braña

Tutor: Ing. Jorge Ernesto Martínez Cabrera

Cuidad de la Habana, junio de 2009

DECLARACIÓN DE AUTORÍA

Declaro ser el 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 17 días del mes de junio de 2009.

Angel Gustavo Suárez Braña

_____
Firma del Autor

Ing. Jorge Ernesto Martínez Cabrera

_____
Firma del Tutor

AGRADECIMIENTOS

A mis padres, mis hermanas y familia por el apoyo y la confianza depositados en mí.

A mi novia por su amor, paciencia y comprensión en todo momento.

A mi tutor por sus consejos y ayuda que fueron cruciales para desarrollar este trabajo.

A mis amigos por su incondicionalidad mostrada en todo momento.

A mis profesores por enseñarme tanto en tan poco tiempo, en especial los de programación.

A todas las personas que de una forma u otra me han ayudado en esta etapa de la vida.

A la Revolución por darme la oportunidad de realizar mi sueño.

DEDICATORIA

A mi mamá por todo su cariño y apoyo, por haberme guiado siempre en todos mis estudios y por ser a quien le debo todos mis resultados, sin su guía nunca hubiera llegado a este momento.

A Mariano e Irobeidis por su ejemplo, quienes se han ganado mi cariño y respeto.

A mi novia por apoyarme tanto y darme la fuerza para seguir adelante en todas las situaciones difíciles.

A mis profesores por formarme como profesional.

A todas las personas que a lo largo de mi vida me han querido y apoyado incondicionalmente.

Y a la Revolución por darme la oportunidad de realizar mi sueño.

RESUMEN

La tensa situación del Sistema Penitenciario venezolano condujo a que, en el marco de la cooperación bilateral Cuba-Venezuela, se contratara un proyecto de humanización del Sistema Penitenciario y dentro de este, el Sistema de Gestión Penitenciario (SIGEP). Entre los principales procesos a informatizar por el SIGEP en su primera versión, se encuentran Educación y Trabajo. Para la gestión de estos procesos se definieron sendos módulos que forman parte del núcleo del sistema: Clasificación y Tratamiento.

Este trabajo se basa en las actividades realizadas por el autor en los roles de diseñador e implementador como integrante del proyecto SIGEP. En él se muestra el diseño y la implementación de la solución brindada para los módulos Educación y Trabajo, a partir de los requisitos definidos con el cliente y el análisis de la arquitectura especificada para el SIGEP.

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

Como resultado de todas las actividades realizadas se integraron ambos módulos como componentes ejecutables al SIGEP y se validaron en un ambiente de trabajo real en un centro penitenciario en Venezuela.

Palabras claves:

Educación, Trabajo, SIGEP, Arquitectura, Diseño, Implementación.

TABLA DE CONTENIDOS

INTRODUCCIÓN

A partir de la llegada a la presidencia de la República Bolivariana de Venezuela del Comandante Hugo Rafael Chávez Frías, se inició una ola de transformaciones sociales, por lo que a finales del año 1999 se aprobaron nuevos cambios en la Constitución de la República Bolivariana de Venezuela, los cuales estaban encaminados a una mejora de la sociedad. En este marco nace el proyecto de Humanización del Sistema Penitenciario, producto del artículo 272 de la constitución venezolana, el cual estipula: "El estado garantizará un sistema penitenciario que asegure la rehabilitación del interno o interna y el respeto a sus derechos. Para ello, los establecimientos penitenciarios contarán con espacios para el trabajo, el estudio, el deporte y la recreación, funcionarán bajo la dirección de penitenciarístas profesionales con credenciales académicas universitarias (…)".La cooperación brindada por Cuba en este campo incluye la atención a la salud de los internos, asesoría especializada y el desarrollo de un sistema informático para gestionar y automatizar los procesos penitenciarios conocido por Sistema de Gestión Penitenciaria (SIGEP).

El SIGEP constituye la solución de software para la informatización de la gestión de los privados de libertad de la República Bolivariana de Venezuela. El sistema, como solución integral para la informatización de los procesos penitenciarios, abarca varias de las especialidades carcelarias dentro de las que se encuentran: Control Penal, Clasificación y Tratamiento, Custodia y Seguridad, Administración Penal y Salud.

Clasificación y Tratamiento es la especialidad penitenciaria cuya función es la de llevar el control de las actividades de reinserción de los privados de libertad. Dentro de estas actividades de reinserción se destacan las de Educación y Trabajo, las que contribuyen en gran medida a la reivindicación del recluso. Esta es una de las áreas claves de un establecimiento penitenciario, de la eficiencia del funcionamiento de esta, depende en buena medida, la garantía de preservar los derechos de los privados de libertad.

Una parte importante del SIGEP es gestionar toda la información referente a las actividades educativas y de trabajo que realizan los reclusos. En la actualidad, los establecimientos penitenciarios de la República Bolivariana de Venezuela, no cuentan con una organización adecuada de estos procesos, solo algunas funciones son asumidas, y cada penal lleva la información de una manera dispersa y no uniforme.

Dentro de los procesos a informatizar en el SIGEP se encuentran Educación y Trabajo, procesos de los cuales se han definido y validado con el cliente los requisitos a informatizar, restando el siguiente paso en el proceso de desarrollo del sistema: el diseño y la implementación.

A partir de esta situación se ha definido el siguiente **Problema a Resolver:**

¿Cómo diseñar e implementar los Módulos Educación y Trabajo, a partir de los requerimientos de software establecidos, teniendo en cuenta la arquitectura y tecnologías definidas para el proyecto SIGEP?

El **Objetivo General** de este trabajo es Diseñar e Implementar los Módulos Educación y Trabajo del SIGEP a partir del análisis de los requerimientos de software establecidos en acuerdo con el cliente y haciendo uso de las tecnologías y herramientas definidas para el proyecto. Por lo tanto los resultados esperados son los Modelos de Diseño e Implementación de los Módulos Educación y Trabajo, la validación de dichos modelos, y los módulos como componentes ejecutables integrados al SIGEP.

En consecuencia, el **Objeto de Estudio** es el proceso de desarrollo de los módulos Educación y Trabajo del Sistema Penitenciario venezolano y el **Campo de Acción** los modelos de Diseño e Implementación de los módulos Educación y Trabajo del SIGEP.

La realización de este trabajo está basado en el desempeño del autor en los roles de diseñador e implementador del proyecto SIGEP. Así en función de los objetivos y las actividades a realizar de acuerdo a los roles desempeñados, se han trazado las siguientes **Tareas a Cumplir**:

- Estudio de las tecnologías y herramientas a utilizar en el Diseño e Implementación de los Módulos Educación y Trabajo definidas en la arquitectura del SIGEP.
- Análisis de los requisitos de software y del modelo de negocio correspondiente a los módulos Educación y Trabajo.
- Diseño de la solución de software para los requisitos relacionados con los módulos Educación y Trabajo.
- Implementación del Diseño relacionado con los módulos Educación y Trabajo. Realizar pruebas unitarias y de integración.
- Integración de los módulos de Educación y Trabajo al SIGEP.

El documento consta de 3 capítulos más anexos.

El Capítulo 1 contiene la fundamentación teórica del trabajo de diploma. En él se realiza una descripción de las principales herramientas que se utilizaron, así como de la plataforma en la cual se trabajó y la metodología utilizada.

El Capítulo 2 aborda la arquitectura del SIGEP. Explica el modelo que se utiliza y las capas por las que está compuesto. Se analizan las actividades y pautas generales para el diseño e implementación de los módulos tratados dentro del SIGEP.

El Capítulo 3 explica las aplicaciones de patrones de diseño en el diseño de los módulos Educación y Trabajo, se presentan fragmentos de diagramas y código fuente, explicativos del trabajo realizado en el diseño e implementación de los módulos.

Los anexos muestran toda la documentación generada como resultado de los procesos de diseño e implementación de los módulos Educación y Trabajo.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capitulo se abordan algunos conceptos del Sistema penitenciario venezolano que se referencian como parte de la situación problemática y a lo largo del trabajo. Además se analizan antecedentes de software para sistemas penitenciarios en la región de Latinoamérica y las tecnologías, herramientas y tendencias de desarrollo a utilizar en el diseño e implementación de los procesos Educación y Trabajo del Sistema Penitenciario venezolano, en el marco del desarrollo del Sistema de Gestión Penitenciaria.

1.2 Sistemas Penitenciarios

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

Manuel Osorio, creador del diccionario de Ciencias Jurídicas, Políticas y Sociales, asocia al Sistema penitenciario con régimen penitenciario, definiendo este régimen como:"Conjunto de normas legislativas o administrativas encaminadas a determinar los diferentes sistemas adoptados para que los *penados*[1] cumplan sus penas". Se encamina a obtener la mayor eficacia en la custodia o en la readaptación social de los delincuentes. Esos regímenes son múltiples, varían a través de los tiempos; y van desde el aislamiento absoluto y de tratamiento rígido hasta el sistema de puerta abierta con libertad vigilada (Osorio, 1997).

Cada gobierno define la estructura de su sistema penitenciario de acuerdo a la legislación y las condiciones reales que posee. En el caso especifico de la República Bolivariana de Venezuela, tal sistema está constituido por la legislación vigente, los métodos que se emplearan para lograr su funcionamiento, las diferentes dependencias encargadas de su aplicación, los equipos de trabajo y la infraestructura carcelaria.

1.3 Sistemas Penitenciarios y Sistemas Informáticos

La infraestructura precaria y las dificultades en la estandarización de procesos penitenciarios son características comunes en los sistemas penitenciarios de América Latina e incluso en algunos países de Europa. Sin embargo, en algunos de ellos existen sistemas informáticos que sirven de apoyo a la gestión de los procesos en los centros penitenciarios y en la toma de decisiones a nivel centralizado.

[1] Penado: Persona del sexo femenino o masculino que se encuentra en un centro penitenciario en cumplimiento de una sanción firme de privación de libertad.

Muchos de estos sistemas han sido desarrollados en el marco de la cooperación con organismos internacionales o países con mayor experiencia en la rama.

Para el desarrollo de SIGEP se hizo un análisis de las características de los principales sistemas informáticos implantados en la región y en España con el fin de identificar los principales beneficios y limitantes que un sistema como este pueden producir. La realización de este análisis se baso en la información que cada uno de ellos proporciona en sus sitios digitales.

A continuación se abordan características de algunos de ellos:

1.3.1 Establecimiento Virtual en la Red (España)

Sitio web del proyecto: http://sourceforge.net/projects/epvnet

Establecimiento Penitenciario Virtual en Red (EPVNET). Es un proyecto de software libre para la gestión informatizada de centros penitenciarios.

Tiene como precedente en España algunas aplicaciones basadas en la arquitectura Cliente-Servidor y bajo entorno Windows que cubren algunas áreas o nominas de internos, empleados, la productividad de los talleres, tramitación de los expedientes penales y penitenciarios de los internos; pero con la limitante de no contemplar áreas de gestión como las que afectan al control de los movimientos de la población interna, las comunicaciones, el control de situaciones regimentales, control de acceso, etc.

El proyecto EPVNET es un software para la gestión de los internos y los empleados en cualquier centro penitenciario basado en la legislación penal española que garantiza el control de movimientos, de comunicaciones e informes. Comenzó como un proyecto aislado en el 2002 para dar una solución de gestión en el centro penitenciario de Valencia. Para el año 2004 se convirtió en un proyecto de software libre hospedado en sourceforge.net bajo licencia GNU/GPL[2]. Es una aplicación web desarrollada en PHP4 y PL/SQL, con PostgreSQL como gestor de base de datos.

1.3.2 Sistema Penitenciario del gobierno de Panamá.

Sitio web oficial http://sistemapenitenciario.gob.pa

Este sistema fue creado a principios de año 1997, como resultado de un proyecto financiado por las Naciones Unidas y el gobierno español en coordinación con la Dirección General de Sistemas penitenciarios de Panamá (DGSP). La finalidad del software es mantener en una base de datos los registros de los internos que están detenidos en los centros penales a nivel nacional. Es un sistema centralizado, con una sede que se enlaza con otras unidades operativas distribuidas en una parte de los centros penitenciarios, con el resto, mantiene un enlace mediante el sistema de actualización por disco de tres y media.

[2] GNU/GPL: Es una licencia creada por la Free Software Foundation y su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

El proceso se inicia con la captura de la información por parte del personal ubicado en el centro penal. Esta información se refiere a los datos personales del interno, antecedentes médicos, datos socioeconómicos y jurídicos y algún otro dato de importancia.

La información capturada se registra automáticamente en una base de datos Oracle que radica en la sede central y la cual está disponible para los diferentes departamentos que tiene acceso al sistema. Esto garantiza que se refleje de forma inmediata los cambios en el expediente del interno tales como traspasos de autoridad, traslados de un centro a otro, libertades, diligencias médicas, jurídicas y la realización del cómputo automático de la sentencia.

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

1.3.3 Sistema Penitenciario Bonaerense (Buenos Aires, Argentina)

Sitio web oficial *http://www.spb.gba.gov.ar*

Desde el año 2003 el Sistema Penitenciario de Buenos Aires, Argentina, posee un sistema informático especialmente desarrollado para mantener una base de datos de personas privadas de su libertad.

Este sistema que se encuentra implantado en solo una provincia argentina constituye un punto de avance en la recogida de información a individuos puesto que recopila sus datos personales, algunas incidencias de su expediente judicial y las novedades que lo relacionan. Toda la información recogida es emitida por Fax al registro general de internos desde las unidades penitenciarias.

Sin embargo la base de datos de privados de libertad se encuentra desactualizadas por el volumen de partes diarios a procesar y la deficiente comunicación procedente de los juzgados en cuanto a avances en la causa y modificaciones al expediente judicial. La información de actos de violencia no se recopila sistemáticamente y no posee un formato preestablecido.

1.3.4 Ventajas y Limitaciones

Los sistemas informáticos del gobierno de Panamá, de la provincia de Buenos Aires, (Argentina) y el proyecto libre español EPVNET tienen como principal limitante la definición de los procesos penitenciarios que los fundamentan y la infraestructura física sobre la cual se implantan. En la mayoría de los casos la falta de conectividad entre los centros penitenciarios imposibilita la instalación de sistemas centralizados que garanticen la integridad e inmediatez de la información. Además se evidencia una precaria y tardía comunicación entre el poder judicial y el sistema penitenciario lo cual conduce a desconocimiento de la situación jurídica actual de cada privado de libertad.

La mayoría de estos sistemas implementan términos de legislación vigente en los países para los cuales fueron definidos lo que los convierte en sistemas no portables a otro sistema penitenciario.

A pesar de sus muchas limitaciones estos sistemas sirven de apoyo a los procesos de sus sistemas penitenciarios y funcionan como herramienta de trabajo a nivel local e institucional.

1.4 Sistema Penitenciario de Venezuela

En la República Bolivariana de Venezuela, cuyo sistema penitenciario actual comparte las características básicas de los sistemas penitenciarios latinoamericanos, existen estrategias dispersas de colectar la información de los individuos privados de libertad en formato digital, ya sea en hojas de cálculo o documento de texto, con el fin de facilitar el trabajo de los funcionarios de cada penal, sin que medien formatos estándares para ello. No existe registro de sistema informático precedente que sirva como apoyo a los procesos penitenciarios, ni a la toma de decisiones a nivel centralizado.

Con el desarrollo del SIGEP en colaboración con instituciones especializadas cubanas se plantea la resolución a mucha de estas problemáticas. El resultado esperado de este convenio es un sistema informático centralizado que estandarice e implemente los procesos fundamentales en el sistema penal venezolano y se convierta en herramienta de trabajo para funcionarios en los centros penitenciarios y para la toma de decisiones a nivel institucional.

A partir de la estructura actual del sistema penitenciario venezolano se identificaron las áreas que el SIGEP informatizaría y se conformaron los sistemas y subsistemas que este contendría. En la figura 1.1 se muestran los sistemas y subsistemas del SIGEP, según (ARIAS, 2006) y las relaciones de dependencia entre ellos.

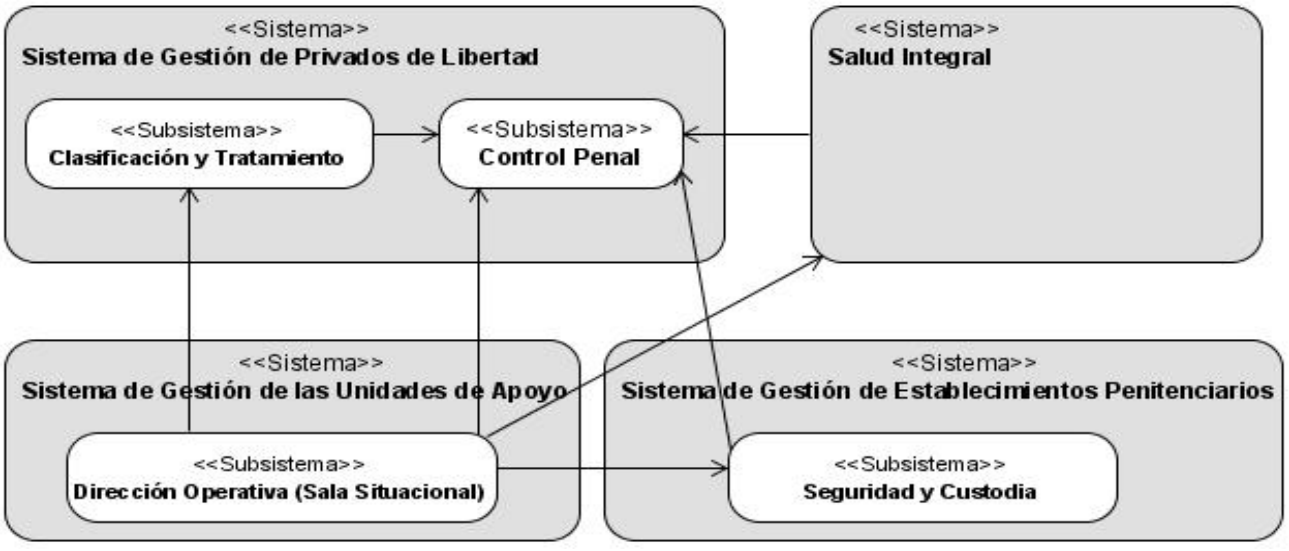


Figura 1.1 Sistemas y subsistemas que integran el SIGEP y sus relaciones de dependencia.

1.4.1 Clasificación y Tratamiento

Uno de los procesos principales a implementar en el SIGEP es el de Clasificación y Tratamiento, el cual es el encargado de controlar las actividades de reinserción de los privados de libertad. El mismo incluye los procesos Educación, Trabajo, Deporte, Cultura, Vínculos, Clasificación y Evaluaciones Técnicas, a través de los cuales Clasificación y Tratamiento interactúa con las áreas de Control Penal y Sala Situacional.

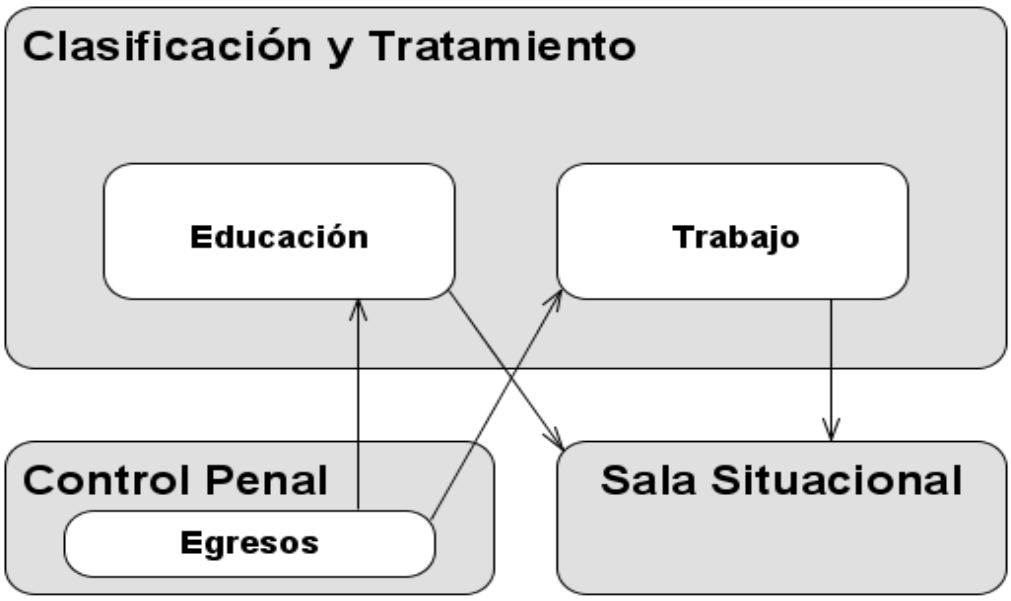


Figura 1.2 Mapa de relaciones entre procesos de Educación y Trabajo

La figura 1.2 muestra las relaciones internas de los procesos Educación y Trabajo. Llámese relaciones internas los procesos o entidades, dentro del sistema penitenciario con los que el proceso intercambia información. La dirección del flujo de información es representada por las saetas.

Como se aprecia en la Figura 1.2, Clasificación y Tratamiento contiene Educación y Trabajo, procesos que se abordan a continuación por constituir parte del objeto de estudio del trabajo de diploma.

1.4.2 Proceso Educación

Las actividades educativas que se desarrollan en los centros penitenciarios venezolanos pueden ser de dos tipos, formales y no formales.

Formales

Las actividades educativas formales se refieren a las que elevan el nivel de escolaridad de los privados de libertad (primaria, secundaria, bachillerato, universidad), y tiene como

objetivo garantizar a los privados de libertad la prosecución de su educación, a través de un proceso de enseñanza aprendizaje que promueva el libre desenvolvimiento de sus competencias cognitivas, así como también el desarrollo de su potencial humano, logrando con ello la superación personal y la modificación de su conducta, lo cual contribuye a su reinserción social.

No Formales

Las actividades educativas no formales son las que tienen como objetivo promover el desarrollo de habilidades, destrezas y aprendizaje en general, que permitan a los individuos incorporarse a las actividades productivas en función de sus intereses. Contribuir a la formación de individuos capaces de interrelacionarse con otros a fin de estimular con ellos los valores fundamentales donde se manifiestan el respeto, la responsabilidad, la libertad, la justicia, la honestidad y el trabajo Cooperativo.

1.4.3 Proceso Trabajo

Las actividades laborales que se realizan en los centro penitenciaros venezolanos se organizan en unidades productivas y tienen como objetivo crear, conservar y perfeccionar las destrezas, aptitudes y hábitos laborales de los individuos con el fin de prepararlos para las condiciones de trabajo en libertad, obtener un proyecto económico y fortalecer sus responsabilidades personales y familiares.

En resumen estos procesos generan una participación activa y disciplinada que puede permitir obtener, a los penados, la reducción de la pena.

1.5 Tecnologías y herramientas utilizadas en el desarrollo

La selección adecuada de herramientas y tecnologías a utilizar en el desarrollo de un software tiene estrecha relación con el tiempo de desarrollo y la calidad final del producto. Todas las herramientas utilizadas en la realización de SIGEP fueron definidas por el grupo de arquitectura así como las tecnologías y flujos de procesos. Es por ello que en este epígrafe no se definen los instrumentos utilizados sino que se realiza una identificación de cada uno de ellos y las ventajas que proporciona en el trabajo de diseño e implementación de los módulos tratados.

1.5.1 Plataforma de desarrollo

J2EE

Es un estándar de la industria para desarrollar aplicaciones empresariales portables, robustas, escalables y seguras utilizando *java* como lenguaje. J2EE define una arquitectura para desarrollar aplicaciones distribuidas complejas utilizando un modelo multicapas. La lógica de aplicación está dividida en componentes de acuerdo a su función. Estos pueden estar instalados en diferentes nodos de acuerdo a la capa a la que pertenezcan. Componentes de la capa cliente se ejecutan en la maquina cliente, los componentes *web* y

de la capa de negocio se ejecutan en el servidor de aplicaciones y otros componentes en sistemas legados o servidores de base de datos.

J2EE consiste en:

- **Una especificación:** define los requisitos que debe cumplir una implementación de un producto J2EE.
- **Un Modelo de Programación**: guía de diseño para desarrollar aplicaciones J2EE.
- **Una plataforma:** conjunto de *API*[3]*s*, tecnologías y herramientas de desarrollo.
- **Una implementación de referencia:** Implementación de ejemplo de los servicios brindados por la plataforma.
- **Una Suite de pruebas de compatibilidad:** certifica la compatibilidad de un producto con la especificación J2EE a través de varios tipos de pruebas.

La plataforma J2EE reduce significativamente el esfuerzo necesario por los desarrolladores de aplicaciones empresariales, proveyendo una robusta arquitectura que de otra forma tendrían que implementar ellos mismos. De esta manera los desarrolladores se pueden concentrar solo en la implementación de componentes que satisfagan las necesidades propias del negocio.

Entre las *APIs* de la plataforma J2EE utilizadas en el desarrollo del SIGEP se encuentran: Java-Servlets, Java Server Pages y JDBC.

Java-Servlets

Permite extender las capacidades de un servidor de aplicaciones que es accesible a través del modelo petición-respuesta. Generalmente es utilizado en ambientes *web* como es el caso del SIGEP. Los *servlets* proveen un mecanismo efectivo de interacción entre la lógica de negocio que se ejecuta en un servidor y las aplicaciones clientes.

Los *servlets viven* en un contenedor de *servlets* que se ejecuta en un servidor *web*. Este contenedor gestiona su ciclo de vida y traduce las peticiones, que un cliente *web* hace a través del protocolo *HTTP* [4], en objetos que las encapsulan. De igual forma, el contenedor traduce las respuestas de los *servlets* al protocolo *web* correspondiente.

[3] API, del inglés *Application Programming Interface*: Es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

[4] HTTP, del inglés *Hyper Text Transfer Protocol*: Es el protocolo sin estado que define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

Java Server Pages (JSP)

La tecnología JSP evolucionó de Java-Servlets, de hecho, una jsp se compila en una especie de servlet que es ejecutado en un contenedor de servlets. Una jsp es un documento texto que tiene contenido estático (texto plano, HTML, XML) y elementos que determinan cómo la página será construida dinámicamente. Las jsps separan la lógica de presentación de la lógica de aplicación, promoviendo la reutilización y la separación de roles en los equipos de desarrollo.

La mayoría de las vistas que se muestran al usuario en el SIGEP son documentos HTML generados a partir de la tecnología JSP.

JDBC

Brinda una interfaz para el acceso a bases de datos relacionales. JDBC generaliza las funciones más comunes de acceso a los datos, abstrayendo los detalles específicos de un proveedor de determinada base de datos. El resultado es un conjunto de clases e interfaces, que pueden ser utilizadas con cualquier gestor que tenga un controlador JDBC apropiado.

Esta API es utilizada por la mayoría de los frameworks de persistencia como Hibernate, Ibatis, etcétera.

1.5.2 Entorno integrado de desarrollo (*IDE*[5])

Eclipse

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

Eclipse es una plataforma libre sobre la cual se pueden acoplar herramientas de desarrollo de todo tipo mediante la implementación de *plug-ins*. Una de las características más importantes de Eclipse es su facilidad de extensión.

El IDE Eclipse es una de las herramientas que se engloban bajo el denominado Proyecto Eclipse. El Proyecto Eclipse aúna tanto el desarrollo del IDE Eclipse como de algunos de los *plug-ins* más importantes como el JDT (Java Development Tools), *plug-in* para el lenguaje Java, y el CDT (C-C++ Development Tooling), *plug-in* para el lenguaje C/C++. El *IDE* Eclipse es gratis y de código abierto pero sus *plug-ins* pueden no serlo dadas las características de su licencia “no viral”.

El sistema de *plug-ins* agiliza el trabajo del equipo de desarrollo en la implementación. Algunos de los *plug-ins* más utilizados en el SIGEP son:

[5] IDE, del inglés *Integrated Development Environment*: Programa compuesto por un conjunto de herramientas para un programador. Entorno de programación que ha sido empaquetado como un programa de aplicación consistente en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

- **Hibernate Tools:** Constituye un conjunto de herramientas para facilitar el uso del framework Hibernate. Las principales funcionalidades que brinda son: un editor de mapeos de los meta datos de la base de datos a las clases y una consola para la ejecución de consultas HQL. Permite realizar ingeniería inversa a partir de la base de datos de las clases y de los mapeos de las entidades. (http://www.tools.hibernate.org/)

- **Spring IDE v2.0 (2007):** Agiliza el trabajo con el framework Spring, principalmente en proyectos grandes, puesto que contiene un editor XML que permite autocompletado y un validador y visor de los beans configurados. Permite la búsqueda y visualización en forma de grafo de todos los beans y dependencias. (http://www.springide.org)

- **SDE v4.4.1 (2007):** Smart Development Environment (SDE) es una *herramienta CASE* [6] ampliamente integrada con Eclipse. Este software de modelado UML soporta el ciclo de vida completo del desarrollo de software. Permite la realización de todos los tipos de diagramas UML en Eclipse, realizar ingeniería inversa desde código Java a diagramas de clases, generar código Java, documentación y mantener el modelo y el código sincronizados durante el desarrollo. (http://www.visual-paradigm.com/product/sde/ec/)

- **Subclipse v1.0.1:** Subversion es un sistema de control de versiones utilizado para permitir el desarrollo de un sistema, a múltiples usuarios al mismo tiempo. Subclipse es un plug-in que permite la interacción con un servidor Subversion y la manipulación del proyecto que reside en este desde el ambiente de Eclipse. (http://subclipse.tigris.org/)

1.5.3 Estándares de codificación

Las convenciones de código son un conjunto de reglas a seguir para escribir código fuente uniforme y legible. Estas reglas comprenden cómo se deben utilizar el espaciado y los saltos de línea, como se deben nombrar las variables, clases y ficheros y como se deben escribir instrucciones especificas del lenguaje de programación.

Las convecciones ayudan a entender el código por lo que su utilización es útil a los desarrolladores en las actividades de mantenimiento a programas escritos por otros o por ellos mismos en el pasado. Que un programa funcione o no es en buena medida independiente de que este bien o mal escrito, sin embargo una mal escrito tiene una probabilidad mayor de no funcionar correctamente además de que es casi imposible de depurar o mantener.

[6] Herramienta CASE, del inglés Computer Aided Software Engineering: Aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo del mismo en tiempo y dinero.

Las convecciones de código utilizadas en el desarrollo del SIGEP son las definidas para ArBaWeb (PÉREZ, 2007) que incluyen las definidas para el lenguaje Java por Sun Microsystems en http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html. Para la documentación de las clases se utilizan las convecciones de Javadoc[7] adaptadas para el SIGEP.

1.5.4 Patrones de diseño de software

Los patrones de diseño de software son una solución probada para un problema general de diseño, en un contexto determinado. Encierran la experiencia que programadores e ingenieros han adquirido en la solución de problemas comunes. En el rol que se desarrolla en el trabajo de diploma, ayuda al autor a completar el diseño de una solución de manera rápida, flexible y segura.

Los patrones de diseño pueden incrementar o disminuir la capacidad de comprensión de un diseño o de una implementación, disminuirla al añadir accesos indirectos o aumentar la cantidad de código, incrementarla al regular la modularidad, separar mejor los conceptos y simplificar la descripción. Benefician la documentación y el mantenimiento de los sistemas pues proveen una especificación de los objetos y clases y sus relaciones, así como el objetivo del diseño.

Es importante notar que un patrón de diseño representa experiencia y conocimiento. No es software ejecutable por lo que, cada vez que se hace, debe ser implementado nuevamente.

Algunos de los patrones a los que se hace referencia en este documento por su utilización en la propuesta de solución técnica son:

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

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

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

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

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

1.5.5 Frameworks

Un framework es "una mini arquitectura reusable que provee una estructura y comportamiento genéricos para una familia de abstracciones de software […]" (KAISLER, 2005). Es un conjunto de componentes con interfaces bien definidas que interactúan entre sí para cumplir una tarea.

La *GoF*[8] define un *framework* como "un conjunto de clases que constituyen un diseño reutilizable para un tipo específico de aplicaciones".

Un framework no tiene funcionalidades de una aplicación específica, sino que las aplicaciones se construyen sobre ellos. Para usar un framework es necesario personalizarlo, extendiéndolo o componiendo las distintas instancias de sus componentes e insertando las funcionalidades específicas de la aplicación en ciertos puntos que el framework provee con ese fin. Luego el framework funciona *solo* invocando las rutinas específicas de la aplicación.

En el desarrollo de aplicaciones empresariales similares al SIGEP, el uso de frameworks se ha convertido en algo imprescindible porque implica ahorro en tiempo de diseño y código puesto que implementa un conjunto de patrones de diseño, lo que permite su reutilización como componentes de software y a su vez suelen implementar las partes más engorrosas y difíciles del dominio del problema permitiendo que el desarrollador se concentre en implementar las tareas propias de la aplicación.

Existen varias clasificaciones para los frameworks de acuerdo a diferentes criterios. A continuación se muestra una de estas clasificaciones según Taligent (KAISLER, 2005):

- **Frameworks de Aplicación:** Proveen un amplio rango de funcionalidades típicamente usadas en una aplicación. Interviene en varias capas de la aplicación como Interfaz de Usuario, Acceso a Datos, etcétera.

- **Frameworks de dominio:** Proporciona funcionalidades para un dominio específico de aplicación.

[8] GoF, del inglés *Gang of Four*: Es el nombre con el que se conoce a los autores del libro *Design Patterns*, referencia en el campo del diseño orientado a objetos. La 'Banda de los cuatro' se compone de los autores: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

- **Frameworks de soporte:** Se dirigen a dominios muy específicos dentro de una aplicación como manejo de memoria, reportes, etcétera.

Sobre la plataforma J2EE existen un conjunto amplio de frameworks que utilizan e integran las APIs que esta brinda. La mayoría de estos frameworks constituyen referencias para otras plataformas.

En el desarrollo de la arquitectura base del SIGEP se utilizó el framework de aplicación *Spring* que al igual que el framework de soporte *Hibernate* son muy populares, por sus múltiples ventajas, en el desarrollo de aplicaciones web dentro de la plataforma J2EE.

1.5.5.1 Framework Spring

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

Spring es un framework basado en la *Inversión de Control* [9] y en la *Programación Orientada a Aspectos*[10]. Se distribuye de forma libre y su código es abierto. El costo de utilizar Spring en términos de rendimiento tanto para la aplicación que se realiza como para el sistema y hardware que lo soporta es despreciable puesto que su consumo de recursos es mínimo. Es no intrusivo, o sea, las clases de una aplicación basada en Spring generalmente no dependen de las clases específicas del framework. Por estas características se clasifica como un framework ligero.

Permite configurar complejas aplicaciones a partir de componentes simples. En Spring los objetos de la aplicación se declaran en ficheros (normalmente en formato xml) y el framework se encarga de instanciarlos y configurarlos correctamente a través de la inyección de dependencias. Mediante esta técnica los objetos reciben pasivamente sus dependencias sin necesidad de crearlas o buscarlas, proporcionando un bajo acoplamiento entre los componentes de la aplicación.

Spring provee soporte para la programación orientada a aspectos permitiendo separar la lógica de la aplicación de los servicios de sistemas como la auditoría y la gestión de transacciones. Esta ventaja facilita que la implementación se centre en el negocio y no en otros tipos de servicios.

Módulos de Spring

Spring está dividido en módulos bien definidos (Figura 1.4), pero no obliga a hacer uso de todos ellos. Se puede elegir los módulos que se necesiten en el caso específico y buscar

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

[10] Programación orientada a aspectos (AOP): Paradigma de programación cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos.

otras opciones cuando Spring no satisfaga los requisitos. A través de ellos brinda puntos de extensión con varios de los frameworks y bibliotecas de la plataforma J2EE como Hibernate, JMS, JMX, RMI y Jax-RPC.

La estructura de los módulos de Spring, tal como se muestran en la figura, tiene como base el Core que contiene las funcionalidades fundamentales del framework. Sobre el Core se desarrollan los restantes módulos que a su vez pueden constituir la base de otros, por ejemplo los módulos Web y ORM.

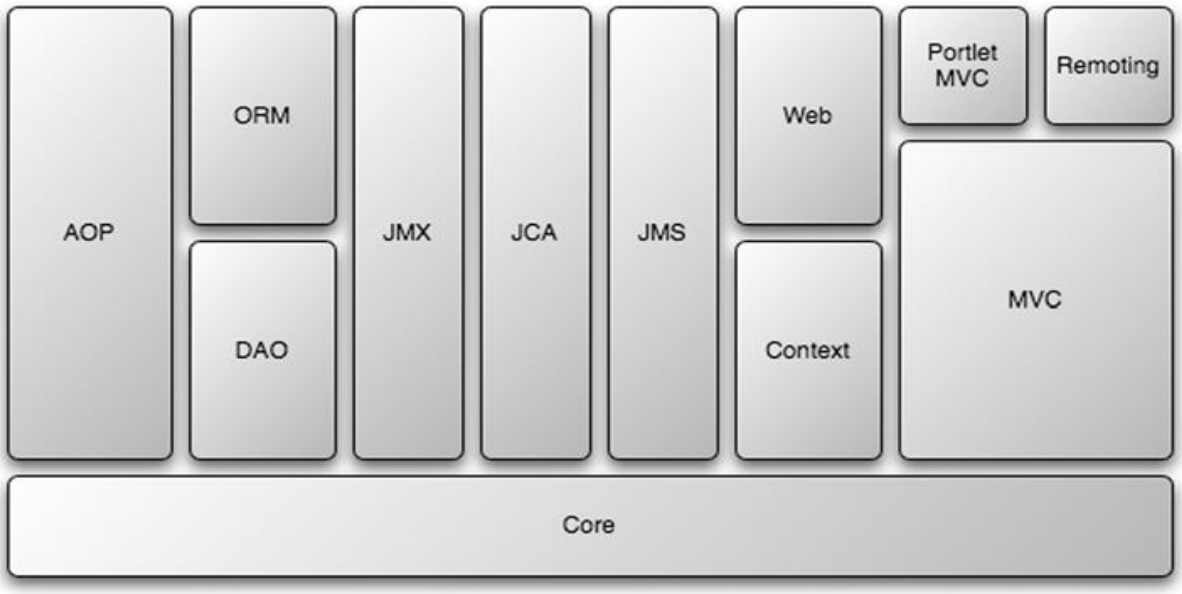


Figura 1.4 Módulos del Framework Spring (WALLS, 2007)

De todos sus módulos, los más utilizados en la implementación del SIGEP son:

- **Spring-ORM:** El módulo provee una forma conveniente para construir la capa de acceso a datos basados en el patrón DAO y se integra a las soluciones *ORM*[11] como Hibernate, Java Persistence API, Java Data Objects, Apache OJB, JDO, iBATIS SQL Maps y Oracle's TopLink. Brinda además algunos servicios, como la integración con el mecanismo de transacciones declarativas de Spring y el manejo transparente de excepciones.

- **Spring-MVC:** El modelo MVC es muy utilizado en la construcción de aplicaciones web. Spring se integra con frameworks de soporte MVC como Struts, JSF, WebWork y Trapestry, pero también provee su propia implementación del modelo MVC basada en el patrón Controlador Frontal.

[11] ORM, del inglés *Object Relacional Mapping*: Persistencia automática y transparente de objetos de una aplicación en una base de datos relacional utilizando meta datos que describen la correspondencia entre el objeto y las tablas de la base de datos.

1.5.5.2 Framework Hibernate

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

Hibernate es una framework ORM para la plataforma Java disponible además para la plataforma .NET con el nombre de NHibernate.

Distribuido bajo los términos de la licencia GNU LGPL ha ganado popularidad como herramienta de soporte a la capa de acceso a datos en el desarrollo de aplicaciones empresariales. Provee mapeo objeto-relacional básico, y otras características sofisticadas como caché y caché distribuida, carga perezosa y ávida.

Como todas las herramientas ORM, Hibernate busca solucionar el problema de la diferencia entre dos modelos ampliamente utilizados para organizar y manipular datos: el orientado a objetos en las aplicaciones y el relacional en las bases de datos. Para lograr esto el desarrollador debe especificar a Hibernate cómo es su modelo de datos. Con esta información Hibernate permite manipular los datos desde las aplicaciones operando sobre objetos con todas las características de la POO. Hibernate convierte los datos que define Java a los que define SQL, siendo transparente esta conversión para el implementador. Genera además las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias. También ofrece un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), y al mismo tiempo APIs para construir las consultas programáticamente.

Hibernate elimina la necesidad de parte del código de acceso a los datos, permitiendo que el desarrollador se concentre más en la lógica de negocio. Esta reducción de código representa un aumento de la productividad y permite que la capa de acceso a datos sea más entendible y fácil de mantener.

1.5.6 Bibliotecas

Dojo Toolkit

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

Es una biblioteca *JavaScript* [12] de código abierto. Resuelve problemas comunes y engorrosos en el desarrollo de aplicaciones con JavaScript como, por ejemplo, la disparidad del modelo de eventos de los distintos navegadores. Provee un conjunto de componentes de interfaz gráfica de usuario como calendarios y menús, que se pueden insertar de manera sencilla en páginas HTML. Estos componentes son utilizados en las interfaces del SIGEP, logrando agilizar el desarrollo al reutilizar código existente con un alto grado de terminación.

[12] JavaScript: Lenguaje de programación interpretado utilizado principalmente para la incorporación de comportamiento dinámico a documentos HTML.

1.5.7 JSON y JSON-RPC

JSON, acrónimo de (JavaScript Object Notation), es un formato ligero para el intercambio de datos, es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

JSON-RPC (JavaScript Object Notation - Remote Procedure Call) es una tecnología que permite hacer llamadas a procedimientos o métodos desde el cliente web.

La utilización de JSON como formato de intercambio y JSON-RPC para hacer llamadas a métodos remotos agiliza el intercambio de datos, y reducen el uso de peticiones y controladores innecesarios que solo eran utilizados para obtener y validar datos.

1.5.8 Gestor de Base de Datos

Oracle

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

Oracle es un sistema de administración de base de datos líder en la industria, utilizable para almacenar todo tipo de datos de negocio, incluyendo datos relacionales, documentos, multimedia, XML y datos de localización. Es fácil de desarrollar y administrar (ORACLE CORPORATION, 2005).

Oracle está definido como el gestor de base de datos a utilizar para el desarrollo y despliegue del SIGEP debido a que ofrece una excepcional disponibilidad, escalabilidad, fiabilidad y seguridad. Además proporciona un adecuado soporte para la réplica de los datos, necesario en el SIGEP, para la comunicación desde los servidores de los centros penitenciarios al servidor del Centro de Datos, que recoge toda la información del sistema a nivel nacional. La infraestructura del SIGEP en tiempo de despliegue identifica que en el Centro de Datos se utilice Oracle Enterprise Edition v10.2.0.3 y en cada centro penitenciario Oracle Standard Edition 10.2.0.3 que es más ligero y cuya licencia es menos costosa.

La principal desventaja de la utilización de Oracle como herramienta son los elevados precios de las licencias de software y del soporte técnico, lo que lo hace un gestor típico de sistemas informáticos de grandes compañías o instituciones gubernamentales. A ello se suma los elevados requisitos de hardware que tiene.

1.5.9 Metodología de Desarrollo

Rational Unified Process (RUP)

Su objetivo es asegurar la construcción de sistemas de software de alta calidad que satisfagan las necesidades de los usuarios finales y clientes cumpliendo con los cronogramas y presupuestos previstos. Es adecuada para sistemas con extensos cronogramas y equipos de desarrollo numerosos.

RUP es un proceso que se caracteriza por ser según (JACOBSON, 2004):

- **Dirigido por casos de uso**. Los casos de uso describen los requisitos funcionales del sistema desde la perspectiva del usuario y se usan para determinar el alcance de cada iteración y el contenido de trabajo de cada persona del equipo de desarrollo.

- **Centrado en la arquitectura**. La arquitectura permite ganar control sobre el proyecto para manejar su complejidad y controlar su integridad. Hace posible la reutilización a gran escala y provee una base para la gestión del proyecto.

- **Iterativo e Incremental**. Se divide en cuatro fases: Inicio, Elaboración, Construcción y Transición, y cada una de ellas se divide en iteraciones. En cada iteración se trabaja en un número de disciplinas haciendo énfasis en algunas de ellas. Las disciplinas propuestas por RUP son: Modelado del Negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, entre otras. Cada iteración añade funcionalidades al producto de software o mejora las existentes.

En la figura 1.3 se muestran los flujos de trabajo y las iteraciones que define RUP por las cuales debe transitar el desarrollo de un producto de software. Además RUP identifica una serie de roles para los trabajadores de cada flujo, dentro de los principales roles se encuentran: jefe de proyecto, ingeniero de requisitos, arquitecto, diseñador, implementador y probador.

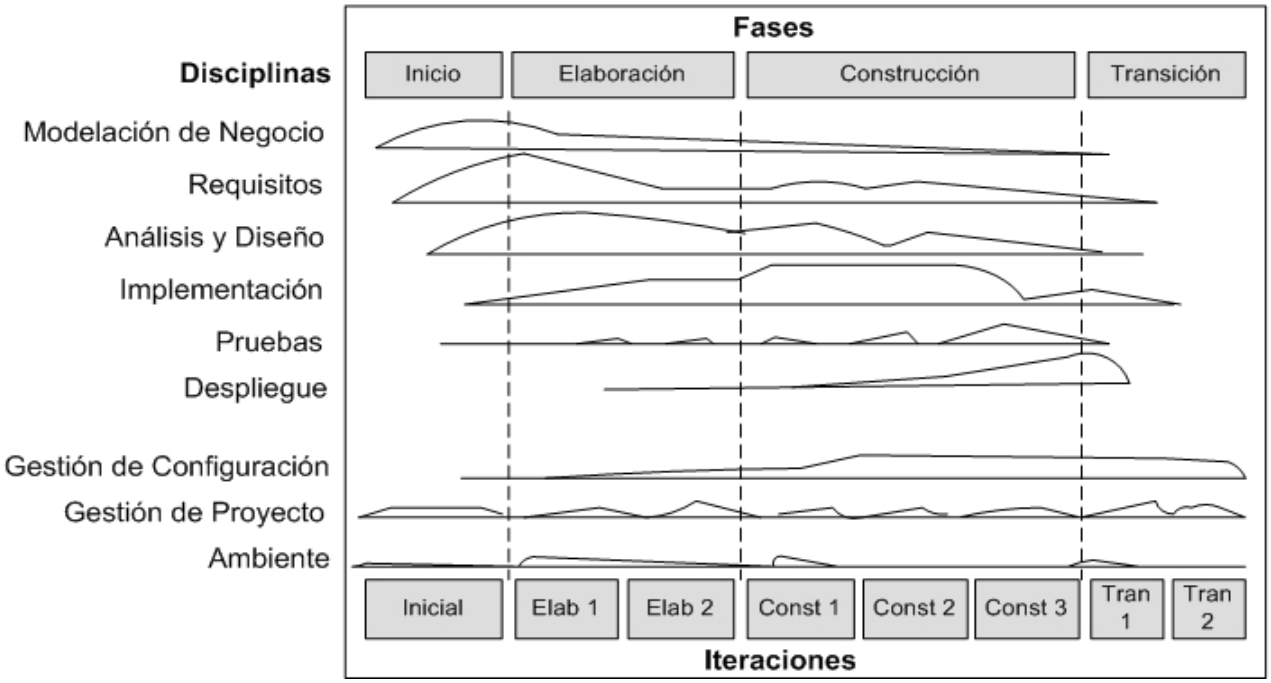


Figura 1.3 Fases y flujos de trabajo en RUP (RATIONAL SOFTWARE CORPORATION, 2003).

A continuación se describen las disciplinas de mayor interés de acuerdo a los roles desempeñados por el autor para la realización del trabajo de diploma.

Disciplina Análisis y Diseño

El objetivo de la disciplina Análisis y Diseño es transformar los requisitos de software en un diseño del sistema, desarrollar una arquitectura robusta y adaptar el diseño al ambiente de implementación. Los principales artefactos que propone RUP para esta disciplina son: Modelo de Análisis, Modelo de Diseño, Modelo de Datos y Modelo de Despliegue. Las entradas fundamentales de esta disciplina son las especificaciones de los requisitos de software obtenidos en la disciplina requisitos. Los trabajadores más relevantes involucrados en este flujo son el diseñador y el arquitecto.

El modelo de Datos describe la representación lógica y física de los datos persistentes utilizados por la aplicación. En casos de aplicaciones que usen sistemas gestores de aplicaciones que usen sistemas gestores de bases de datos relacionales, este modelo debe incluir representaciones para procedimientos almacenados, disparadores, etcétera.

Disciplina Implementación

En esta disciplina se define la organización del código en subsistemas de implementación. Su objetivo principal es la implementación del diseño, además de la realización de pruebas unitarias a los componentes y la integración de los resultados del trabajo de implementadores individuales en un sistema ejecutable.

1.5.10 Herramientas de modelado

1.5.10.1 Visual Paradigm

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

Visual Paradigm 3.1 es una suite completa de herramientas CASE. Independiente de la plataforma y dotada de una buena cantidad de productos o módulos para facilitar el trabajo durante la confección de un software y garantizar la calidad del producto final. Permite generar diagramas de:

- Casos de Uso
- Clases
- Secuencia
- Comunicación
- Estado
- Componentes
- Despliegue

- Objetos
- Interacción
- Entidad Relación
- ORM
- Procesos del Negocio
- Visión general

De los diagramas mencionados se utilizarán en este trabajo los diagramas de clases, interacción del Modelo de Diseño.

Visual Paradigm ofrece dos modalidades: el UML Edition y Smart Development Environment (SDE) para la integración con IDEs de desarrollo como el Eclipse, muy factibles en la realización de ingeniería directa e inversa y en la generación de la base de datos a partir de diagramas entidad relación y ORM.

1.5.10.2 ER/Studio

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

Embarcadero ER/Studio 7.1 es una herramienta de modelado de datos, fácil de usar y multinivel. Utilizada 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 de gran tamaño y complejidad.

Está equipado para crear y manejar diseños de bases de datos 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. Provee a los desarrolladores de una documentación basada en HTML, así como un repositorio para el modelado.

1.6 Conclusiones

Para el desarrollo de los procesos Educación y Trabajo se tuvo en cuenta sistemas similares en el área geográfica en el que se va a implementar finalmente al SIGEP. A partir de análisis de estos se concluyó que no implementan una solución portable para los procesos analizados, muy ligados al proceder específico del Sistema Judicial y Penitenciario venezolano.

SIGEP es una aplicación web de gestión empresarial cuyo equipo de desarrollo es numeroso. Basado en este hecho se utiliza como metodología de desarrollo de software RUP. Los flujos de trabajo en los que se basa la realización del presente trabajo son Análisis y Diseño e Implementación.

Las tecnologías a utilizar contienen tendencias recientes de la programación sobre la plataforma J2EE como la programación orientada a aspectos y la inyección de dependencias. Además los frameworks utilizados se consideran de referencia en su ámbito como lo es Spring como framework de aplicación e Hibernate como framework de soporte para la capa de acceso a datos. El IDE Eclipse con su sistema de plug-ins es una herramienta provechosa pues agiliza el trabajo de Implementación con estos framework y otros abordados en el capitulo.

La aplicación en desarrollo esta solicitada para que se ejecute sobre ambiente libre. De esta manera las herramientas y tecnologías utilizadas en su mayoría son libres y multiplataforma. Esto tiene como consecuencia positiva una reducción notable del costo del sistema por concepto de licencias de software y soporte.

CAPÍTULO 2: ARQUITECTURA DEL SISTEMA

2.1 Introducción

En el siguiente capítulo se describe la arquitectura que utiliza el SIGEP, en consecuencia la arquitectura sobre la cual se desarrollan los módulos Educación y Trabajo. Se abordan las capas por las que está compuesta y se brinda un breve análisis de estas. Se detalla el flujo de actividades a utilizar en el diseño e implementación de un módulo en el SIGEP.

2.2 Arquitectura del SIGEP

La arquitectura del SIGEP se definió en los inicios del sistema y se basó en ArBaWeb (Arquitectura Base sobre la Web), aplicando este framework a las condiciones del proyecto. Esta (la arquitectura) se encuentra organizada desde dos enfoques, uno vertical y otro horizontal, los cuales se describen a continuación.

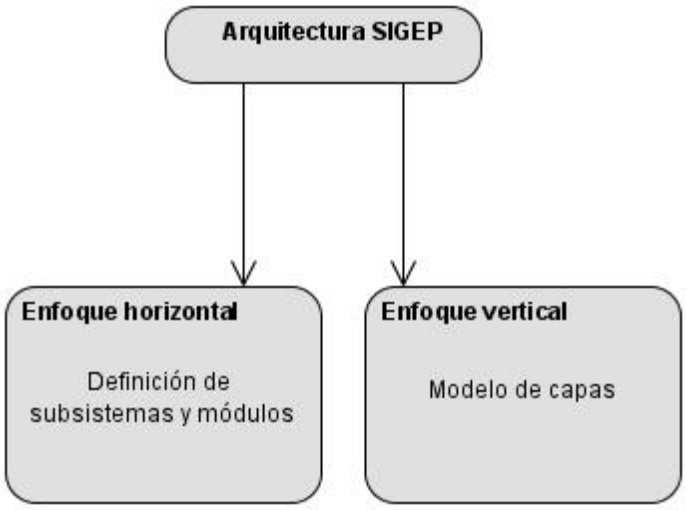


Figura 2.1 Enfoques de la arquitectura del SIGEP

2.2.1 Enfoque horizontal

El SIGEP está constituido por subsistemas y estos por módulos. Los subsistemas se clasifican de acuerdo al proceso que los identificó en subsistema común, subsistemas de negocio y de soporte como se muestra en la Figura 2.2.

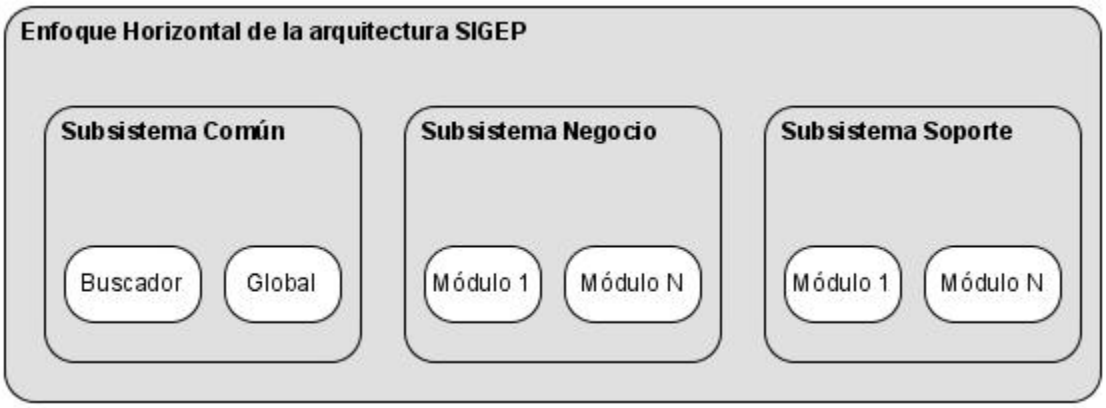


Figura 2.2 Enfoque Horizontal de la arquitectura del SIGEP

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

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

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

Independientemente de su origen, para cada subsistema se definen un conjunto de módulos que constituyen estructuras más atómicas donde recae la implementación de las funcionalidades. Ejemplo de ellos son los módulos Educación, Trabajo, Cultura, Deporte, Vínculos, Clasificación y Evaluaciones Técnicas que se encuentran dentro del subsistema Clasificación y Tratamiento.

2.2.2 Enfoque Vertical

El desarrollo de cada módulo del SIGEP responde a un modelo multicapas donde cada capa tiene funcionalidades y objetivos precisos, así su implementación se encuentra desacoplada de la programación de cualquier otra y la comunicación con una capa inferior ocurre a través de interfaces. Además de estar separadas lógica y estructuralmente, las capas se encuentran separadas de manera física. En cada módulo existe una estructura de paquetes con este fin, como define ArBaWeb (PÉREZ, 2007).

Para la realización del diseño de cada una de las capas es necesario conocer el dominio del módulo a partir del análisis de sus requisitos. Entiéndase como dominio el conjunto de

entidades persistentes y no persistentes, que van ser manejadas por todas las capas y que constituyen la representación estática de la institución en el sistema informático.

Los objetos de dominio no presentan lógica de negocio, sino que esta responsabilidad recae sobre los objetos de negocio los cuales a partir de ahora se tratarán como *manager*. Esto permite utilizar a los objetos de dominio como contenedores de información que se *mueven* entre las capas arquitectónicas de la aplicación.

La distribución de las capas y la interacción entre ellas se muestran en la Figura 2.3, donde las saetas indican la dirección de la dependencia y se muestra la relación, ya abordada, de todas las capas a las entidades del dominio del módulo.

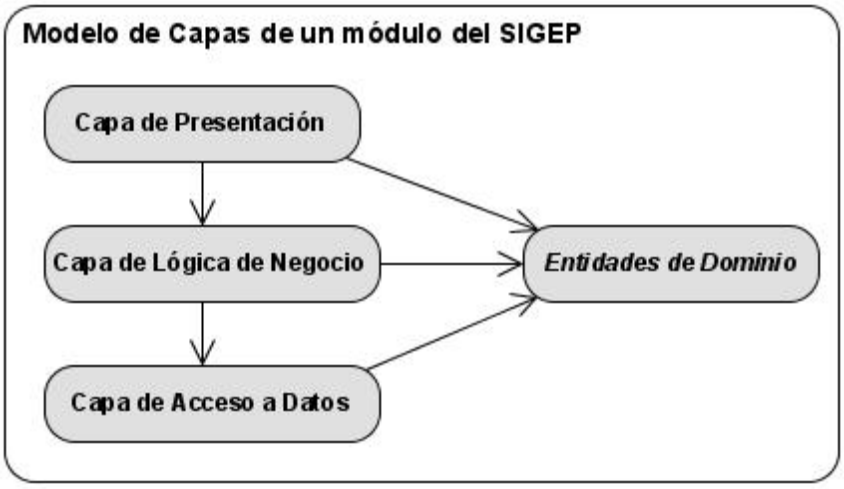


Figura 2.3 Modelo de capas de un módulo en el SIGEP

A continuación se realiza una breve descripción de cada una de las capas por las que está compuesto un módulo del SIGEP de acuerdo a la Figura 2.3.

Capa de Presentación

La capa de presentación define e implementa todo lo relacionado con la interfaz gráfica de usuario. Aquí residen la definición de las peticiones que el usuario puede realizar sobre la aplicación, los controladores que manejan el flujo web y la comunicación con las interfaces de la capa de negocio. Además se encuentran las vistas HTML, XML, PDF, XLS que se muestran al usuario y la implementación del comportamiento dinámico de los documentos HTML a través de Javascript.

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

Capa de Lógica de Negocio

La capa de negocio define e implementa las funcionalidades que responden directamente a los requisitos de manera que se conserve la integridad del sistema y de los datos. Está constituida por managers y sus interfaces.

Los métodos definidos en las interfaces de los managers se ejecutan "envueltos" en transacciones. Las transacciones a nivel de negocio garantizan la ejecución de un conjunto de acciones lógicas o que se reviertan en el caso de alguna anomalía (excepción). Las transacciones se aplican de forma declarativa en los ficheros de configuración del contexto de Spring.

Los métodos expuestos por la fachada (Facade) de esta capa son la única puerta de entrada posible al sistema, por lo que deben ofrecer toda la funcionalidad requerida por este. Esta capa se comunica con la capa de acceso a datos a través de sus interfaces para interactuar con el gestor, almacenar o recuperar de este.

Capa de Acceso a Datos

La capa de acceso a datos es la responsable de recobrar y persistir información desde y hacia la base de datos y de la comunicación con el gestor de base de datos. Esta capa contiene los objetos que encapsulan la lógica de acceso a datos (DAO) e interfaces brindadas para ser accedida desde la capa de negocio. Las implementaciones de los DAOs extienden clases de soporte del framework Spring para el uso de este patrón usando el framework ORM Hibernate, mientras que las interfaces se mantienen independientes de Spring e Hibernate.

2.3 Actividades que se realizan en el diseño e implementación de un módulo del SIGEP

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

El diseño e implementación de un módulo es desencadenado por la *Descripción del Prototipo de Interfaz de Usuario*[13] y el *Proceso Elemental de Negocio*[14] correspondientes. Esta documentación es la especificación de lo que el módulo debe hacer, establecido en acuerdo mutuo y en forma de contrato entre el cliente y el equipo de desarrollo durante la

[13] Descripción del Prototipo de Interfaz de Usuario: Documento que recoge las especificaciones de las funcionalidades a implementar por el SIGEP en sus módulos. Definido por la dirección del proyecto SIGEP.

[14] Proceso Elemental de Negocio: Documento que recoge las especificaciones del negocio asociadas a un módulo del SIGEP. Definido por la dirección del proyecto SIGEP.

captura de requisitos. Tomando como partida esta documentación las actividades a realizar, por el(los) diseñador(es) e implementador(es) son:

1. Análisis de las funcionalidades
2. Diseño de la capa de dominio
3. Diseño del modelo de datos
4. Diseño de la capa de negocio
5. Diseño de la capa de acceso a datos
6. Diseño de la capa de interfaz de usuario
7. Implementación de las entidades del dominio
8. Implementación de las interfaces de los managers
9. Implementación de la interfaz de la fachada
10. Implementación de la capa de acceso a datos
11. Implementación de los controladores
12. Implementación de las páginas JSP
13. Implementación de la lógica en el cliente

A continuación se detalla cómo se ejecutan cada una de dichas actividades:

1. Análisis de las funcionalidades

El análisis de las funcionalidades a implementar se realiza a partir de la documentación generada en la captura de requisitos, en taller de trabajo. El resultado es el entendimiento común de las funcionalidades que el sistema tiene que proveer y las principales restricciones a implementar.

En esta actividad se identifican puntos de contacto con otros subsistemas o módulos y se establece la forma de proceder en esa comunicación. También se definen las interfaces gráficas a partir de las pautas generales definidas para la aplicación y el flujo básico de navegación.

2. Diseño de la capa de dominio

El diseño de la capa de dominio constituye una entrada principal a las restantes actividades de diseño. En este punto se identifican las entidades que serán gestionadas por la capa de negocio, persistidas o recuperadas por la capa de acceso a datos y mostradas por la capa de presentación. También se identifican los nomencladores. Las clases del dominio se definen en el paquete *domain* del módulo.

La definición del dominio, en especial de las entidades persistentes sirve como una primera aproximación al diseño definitivo del modelo de datos.

En la Figura 2.4 se muestra el diagrama de clases de la capa de dominio del módulo Educación.

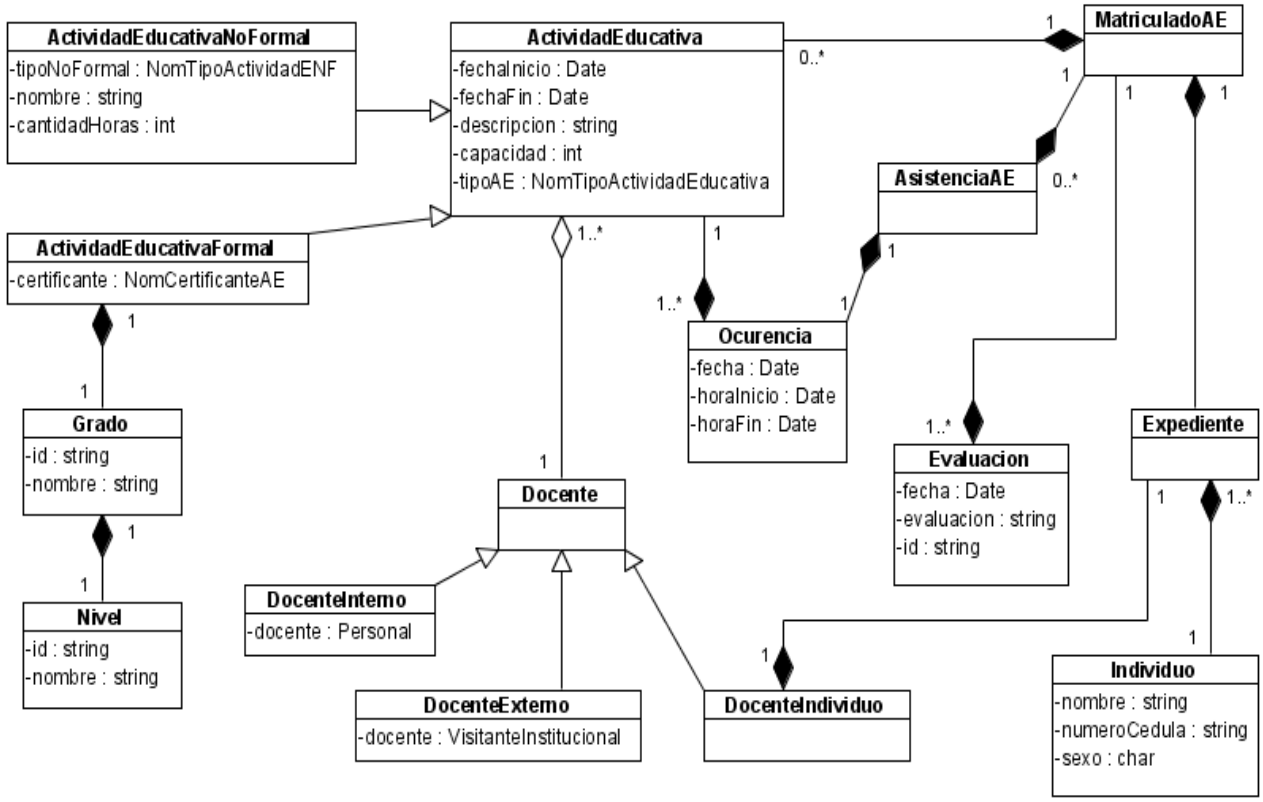


Figura 2.4 Diagrama de clases de la capa de dominio del modulo Educación

3. Diseño del modelo de datos

De esta actividad se obtiene como resultado el modelo de datos de cada módulo. La definición de las entidades de la capa de dominio aporta al modelo de datos una buena aproximación de la estructura estática de la base de datos. Esta estructura tiene que garantizar que el almacenamiento y recuperación de la información ocurra de manera adecuada y que se cumplan las restricciones identificadas, a través de la integridad referencial. En la Figura 2.5 se muestra el diagrama lógico de la base de datos correspondiente al módulo Educación, generado a partir de la definición de las entidades de la capa de dominio.

El modelo de datos es refinado en la medida en que se avanza en las actividades de diseño, ya porque sea necesario persistir nuevos elementos o se decida la utilización de procedimientos almacenados, vistas, restricciones y/o funciones definidas en el gestor de base de datos. La utilización de estos objetos de la base de datos es analizada detenidamente y se decide siempre que implique mayor rendimiento en el acceso a los datos y un acoplamiento mínimo al gestor de base de datos.

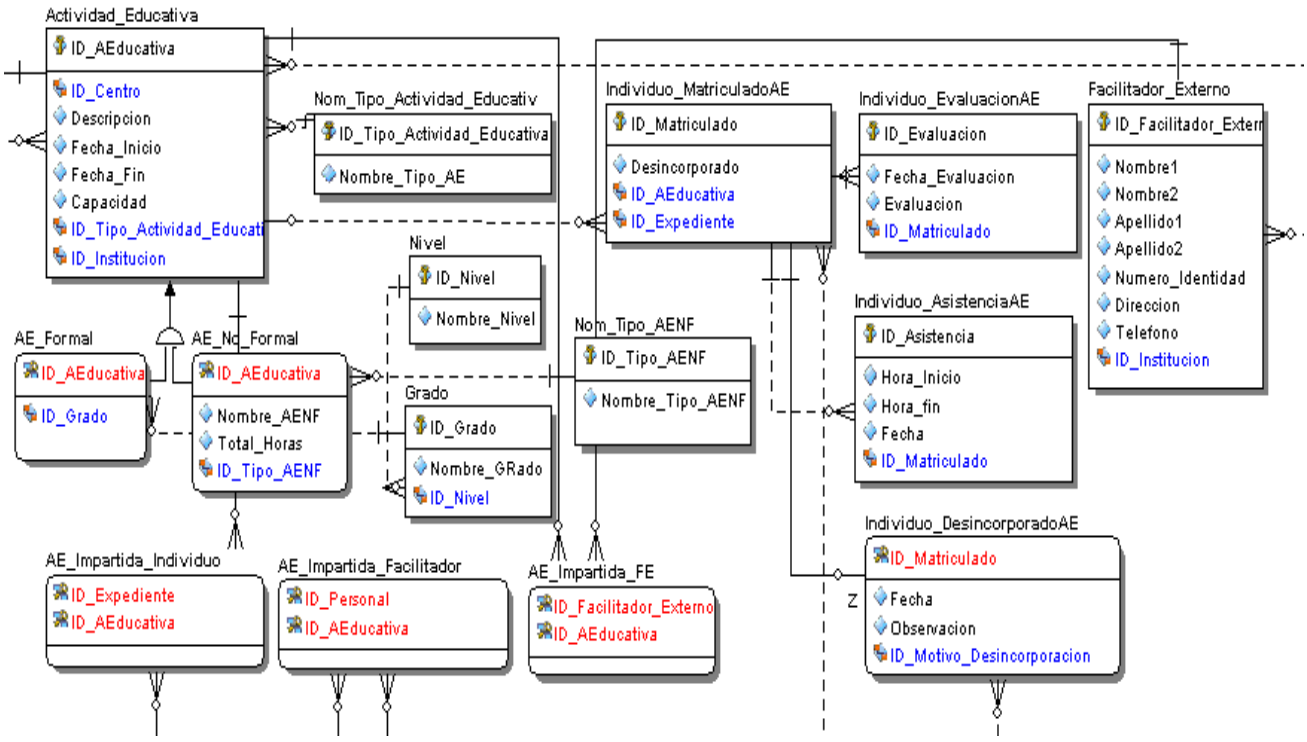


Figura 2.5 Diagrama entidad-relación del modulo Educación

4. Diseño de la capa de negocio

El diseño de la capa de negocio contiene las clases necesarias para cubrir las funcionalidades definidas para el módulo y expone, a través de una fachada, sus funcionalidades a la capa de presentación. Si se necesita la comunicación directa de otros módulos o subsistemas con el que se diseña, se define una fachada para ello y se exponen los servicios necesarios.

La fachada tiene la responsabilidad de conocer a los managers, los cuales son los verdaderos objetos de negocio. Los objetos de negocio se definen de acuerdo al agrupamiento lógico de funcionalidades que respondan a procesos de negocio, es decir, cada manager responde a un "quién" del escenario de la funcionalidad. Se definen también los eventos que pueden ser lanzados o escuchados y las excepciones a lanzar. En la Figura 2.6 se muestra el diagrama de clases correspondiente al diseño de la capa de negocio del módulo Educación a manera de ejemplo de lo explicado. Aquí se muestran la fachada del módulo Educación, las interfaces e implementaciones de los managers definidos, las excepciones que se pueden lanzar, la utilización de funcionalidades definidas en otros subsistemas y además de cómo se maneja el evento lanzado por el subsistema Control Penal para la desincorporación del individuo de las actividades educativas en las que se encontraba matriculado.

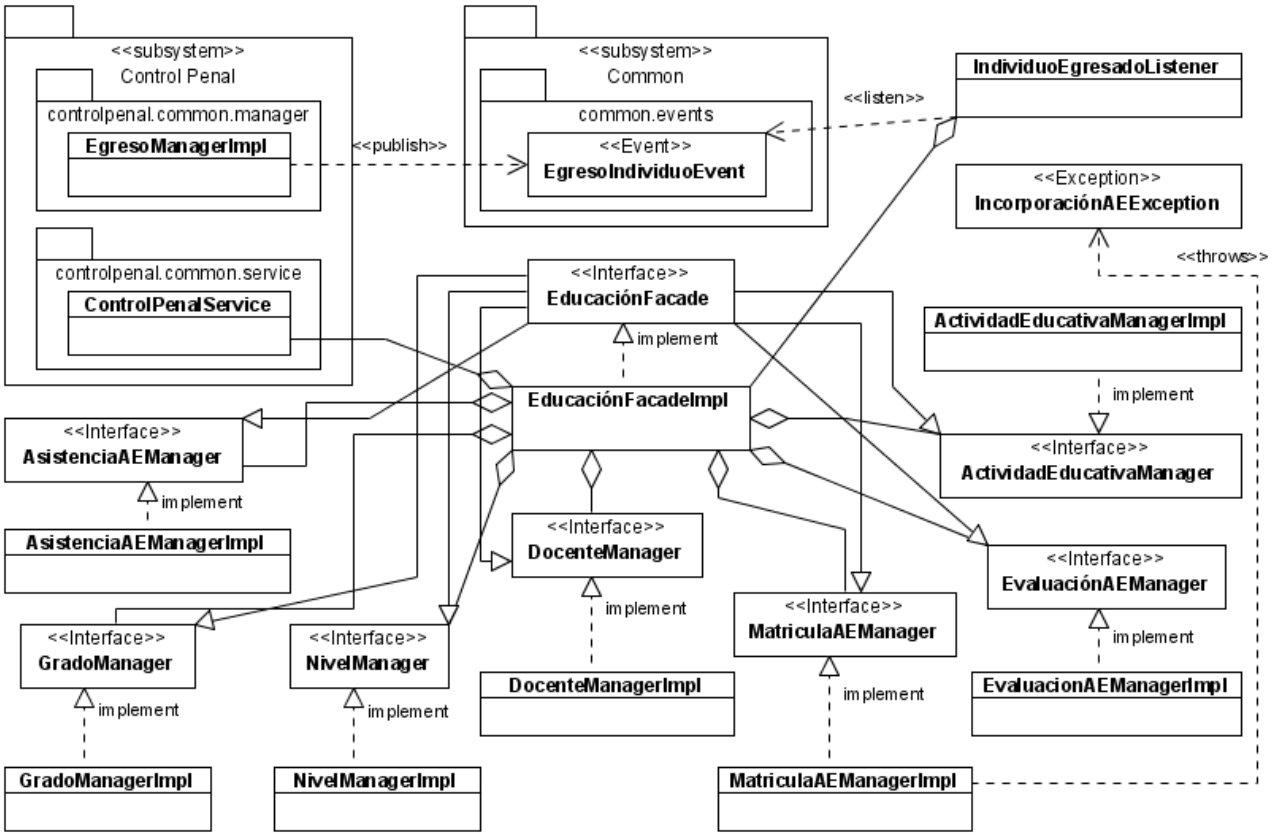


Figura 2.6 Diagrama de clases de la capa de negocio del modulo Educación

5. Diseño de la capa de acceso a datos

El diseño de la capa de acceso a datos se encuentra muy ligado al de la capa de negocio, dado que las funcionalidades de esta capa surgen como necesidades de la capa de negocio. Su diseño se hace de manera que exista el mínimo acople posible al gestor de base de datos, pero sin desaprovechar las potenciales que ofrece el gestor Oracle en casos en que sea ventajoso. El uso del framework ORM Hibernate posibilita en gran medida este desacoplamiento de la aplicación y la base de datos.

Las interfaces de la capa de acceso a datos definen las funcionalidades necesarias relacionadas con la persistencia y recuperación de datos del medio de almacenamiento. Las implementaciones de los DAOs heredan de la clase `cu.uci.arbaweb.dataaccess.dao.impl.AbstractBaseDAO` que a su vez extiende la clase `org.springframework.orm.hibernate3.support.HibernateDaoSupport` e implementa métodos comunes para todos los DAOs como son `findById`, `persist`, `delete`, `findAll`, `findByCriteria` y `findByExample` que se encuentran en la interfaz `cu.uci.arbaweb.dataaccess.dao.BaseDAO`

En la Figura 2.7 se muestra el resultado del diseño de la capa de acceso a datos del módulo educación y la comunicación directa entre un objeto de la capa de acceso a datos del módulo (`AsistenciaAEDAOImpl`) con funciones definidas en un paquete del gestor de base de datos. En este caso particular la función *horasAEIntervalo* que obtiene la cantidad de horas dedicadas a cada actividad educativa (dado un individuo y un intervalo de fecha), y la función *horasAEIntervaloTipo* que obtiene lo mismo que la anterior pero para un tipo de actividad educativa (formal, no formal).

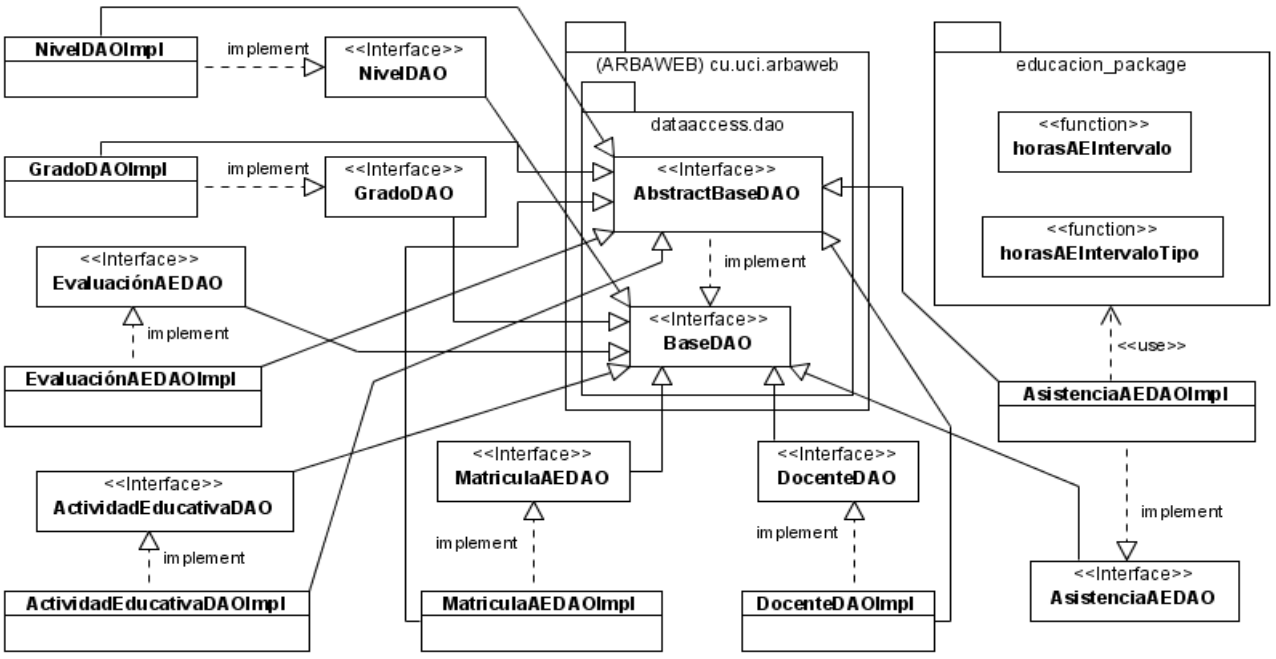


Figura 2.7 Diagrama de clases de la capa de acceso a datos del modulo Educación

6. Diseño de la capa de interfaz de usuario

En esta actividad se definen las vistas y el flujo de navegación, y a partir de esto, las posibles peticiones del usuario y los controladores que las van a atender. El diseño de los controladores generalmente implica diseñar otros componentes que cumplen funciones en el flujo de Spring-MVC como son validadores (*validator*), objetos de respaldo (*command*) y *property editors*. Finalmente se definen componentes del cliente, como son clases JavaScript, documentos HTML, imágenes, etcétera.

El uso de Spring-MVC es de gran beneficio pues este framework provee implementaciones bases de controladores que realizan funciones comúnmente necesarias en aplicaciones web. Por ejemplo, la clase `org.springframework.web.servlet.mvc.SimpleFormController` implementa el flujo de un formulario y la clase `org.springframework.web.servlet.mvc.AbstractWizardFormController` implementa el flujo de un asistente. Al definir los controladores se extienden estas clases agregando las funcionalidades específicas del formulario o asistente particular, con lo que se gana en productividad.

7. Implementación de las entidades del dominio

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

8. Implementación de las interfaces de los managers

Se implementan los métodos definidos para cada una de las interfaces de los managers, ajustándose a las funcionalidades previstas. Cada método implementado tiene que verificar la integridad de los datos e informar a la capa de interfaz cualquier eventualidad a través de las excepciones definidas. Si fuese necesario se implementa la publicación y manipulación de eventos.

Los managers implementados se configuran en el fichero de configuración del contexto de Spring correspondiente a la capa de negocio del módulo. Este fichero se encuentra en el paquete *configuration* y tiene por nombre: **sigep-[*subsistema*]-[*módulo*]-business-context.xml**.

En el Anexo 1 se muestra un ejemplo de la implementación del método `registrarActividadEducativa` perteneciente al manager `ActividadEducativaManagerImpl` que muestra cómo debe ser implementado un manager en el SIGEP.

8.1 **Pruebas unitarias a los managers**

Por cada manager se debe programar una prueba de caja blanca que verifique cada método implementado. Las clases de prueba de los managers se definen en el paquete `manager.impl.test` del módulo y deben heredar, directa o indirectamente, de la clase `org.springframework.test.AbstractDependencyInjectionSpringContextTests` de Spring y sobrescribir el método protected String [] getConfigLocations () definiendo la ubicación física de los ficheros de configuración del contexto de Spring en los que se encuentra el objeto que se está probando y sus dependencias, ya sean implementaciones reales o falsas.

Estas pruebas se benefician de la técnica de inyección de dependencias de Spring y se basan en el framework de pruebas *JUnit*[15].

9. **Implementación de la interfaz de la fachada**

Para la implementación de la interfaz de la fachada solo tiene que conocerse en cuales managers se encuentran implementadas las funcionalidades necesarias para comunicarlas a la capa de interfaz; de esta forma, la fachada no posee ninguna lógica de negocio, solo es experta del lugar donde radica la información que necesita y así la utiliza. El uso de fachadas simplifica el acceso de la capa de presentación a la capa de negocio, reduciendo el número de objetos con los que tiene que interactuar esta.

A continuación se muestra un fragmento de la implementación de la interfaz de la fachada `EducacionFacadeImpl`, que refleja la ausencia de lógica interna en la implementación del método `registrarActividadEducativa`.

```
public class EducacionFacadeImpl implements EducacionFacade {

	private ActividadEducativaManager actividadEducativaManager;

	public void setActividadEducativaManager(
			ActividadEducativaManager actividadEducativaManager) {
		this.actividadEducativaManager = actividadEducativaManager;
	}


	...
	public void registrarActividadEducativa(
			ActividadEducativa actividadEducativa) throws Exception {
		actividadEducativaManager
				.registrarActividadEducativa(actividadEducativa);
	}
}
```

[15] JUnit: Framework utilizado para realizar pruebas unitarias de aplicaciones Java. Creado por Erich Gamma y Kent Beck .Constituye un estándar para pruebas en la plataforma Java y de referencia para otras.

10. **Implementación de la capa de acceso a datos**

La implementación de la capa de acceso a datos comprende fundamentalmente la creación de los ficheros de mapeo de Hibernate (hbm.xml) y la implementación de los objetos de acceso a datos.

Los ficheros hbm se colocan en el paquete `dao.impl.map`. Para la creación de estos ficheros se utiliza el *plug-in* de Eclipse Hibernate Tools, herramienta que aumenta considerablemente la productividad en esta actividad.

La implementación de los DAOs se centra en la utilización de los APIs *Criteria* y *Example* del Framework Hibernate. Estos APIs son una poderosa herramienta para la creación de complejas consultas de forma relativamente fácil pero, en caso de que no satisfagan las necesidades del implementador, se usa el HQL (Lenguaje de consultas de Hibernate), que permite mayor flexibilidad y, en último caso, SQL. La utilización de las APIs mencionadas y del HQL es recomendada porque permite escribir código más sencillo, transparente y tolerante al cambio.

La implementación de las DAOs está acoplada de forma mínima al gestor de base de datos que, en el caso de SIGEP, es Oracle pero en ocasiones se apoya en la utilización de funciones o procedimientos almacenados que residen en este, como fue explicado en la actividad Diseño de la capa de acceso a datos.

Los DAOs implementados se configuran en el fichero **sigep-[*subsistema*]-[*módulo*]-dataaccesscontext.xml**. En la configuración de estos objetos se inyecta el *sessionFactory*[16] de Hibernate como se muestra en el siguiente ejemplo:

```
<bean id="actividadEducativaDAO"
      class="vnz.sigep.tratamiento.educacion.dao.impl.ActividadEducativaDAOImpl">
                  <property name="sessionFactory">
                        <ref bean="sessionFactory" />
                  </property>
</bean>
```

10.1 **Pruebas unitarias a los DAOs**

De forma similar a las pruebas de los managers, se deben efectuar pruebas unitarias a los DAOs. Las clases de prueba de los DAOs se definen en el paquete `dao.impl.test` del módulo y heredan de la clase `org.springframework.test.AbstractTransactionalDataSourceSpringContextTests` de Spring.

Al heredar de esta clase, los casos de prueba se ejecutarán en un contexto de transacciones a las que se puede hacer *rollback*, o *commit* en dependencia de los

[16] Session Factory: objeto que configura al framework Hibernate a partir de los ficheros de mapeo (hbm.xml) y el dialecto a utilizar, según el gestor de base de datos.

intereses de cada prueba. De esta forma se puede probar las funcionalidades de acceso a datos sin afectar los datos en la base de datos.

11. **Implementación de los controladores**

A partir del diseño realizado de la capa de presentación se programan los controladores que manejan el flujo web de la aplicación. Los controladores implementan directa o indirectamente la interfaz `org.springframework.web.servlet.mvc.Controller` para insertarse en el flujo de Spring-MVC.

Estos componentes son los encargados de la comunicación con la capa de negocio a través de su fachada. Son responsables de validar los datos de entrada de la aplicación, formatear los datos de salida y gestionar el flujo web, mostrando las vistas correspondientes. Se configuran en el fichero del contexto de Spring correspondiente a la capa de presentación: **sigep-[*subsistema*]-[*módulo*]-servlet.xml.**

En el Anexo 2 se muestra un ejemplo de la implementación de un controlador, en este caso, `MatricularIndividuosAEController` el cual realiza la incorporación de los individuos a una actividad educativa.

12. **Implementación de las páginas JSP**

Las páginas JSP construyen documentos HTML con los datos que el controlador le envía. En la programación de las JSP se usan fundamentalmente la biblioteca de etiquetas JSTL (Java Standard Tag Library) y las etiquetas de Spring.

El diseño gráfico de las JSP se realiza a través del uso de estilos que radican en el fichero *css*[17] definido para el SIGEP, donde se encuentran todos los estilos de la aplicación.

13. **Implementación de la lógica en el cliente**

En esta actividad se programa, utilizando el lenguaje JavaScript, validaciones en el cliente, y comportamiento de componentes gráficos como calendarios, tablas, botones y pantallas emergentes de error o información al usuario. También desde el cliente se lanzan peticiones al servidor usando AJAX[18] y JSON-RPC.

En el SIGEP se utilizan los componentes de la biblioteca javascript de componentes gráficos DojoToolkit (Dojo 0.42) los cuales se localizan en el módulo *widget* de Dojo.

[17] CSS, del inglés *Cascading Style Sheets*: Lenguaje formal utilizado para definir la presentación de un documento estructurado escrito en HTML o XML

[18] AJAX, acrónimo de Asynchronous JavaScript And XML: Es una técnica de desarrollo web que permite al usuario una mejor interacción con las aplicaciones web, evitando recargar en gran medida la aplicación cada vez que se realiza una petición al servidor web, por lo que el intercambio de información se produce en un segundo plano.

2.4 Conclusiones

En el capítulo se realizó un análisis de la arquitectura del SIGEP que contribuye a una mejor comprensión de la solución desarrollada.

La arquitectura del SIGEP basada en capas independientes estructuralmente posibilita la división del equipo de trabajo por roles que, unido al flujo de trabajo definido para la ejecución de un módulo, permite la programación de las capas simultáneamente. Cuando el trabajo sea realizado por una sola persona el flujo de trabajo constituye la guía para realizar actividades de forma orgánica y orientada a la obtención de resultados parciales. En cada actividad del flujo se ejemplificó el uso de los frameworks y tecnologías mencionados en el Capítulo 1 y la manera de integrarse para lograr el diseño y la implementación de los módulos tratados: Educación y Trabajo.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

3.1 Introducción

En este capítulo se soluciona los módulos Educación y Trabajo de acuerdo a su Diseño e Implementación. Se utiliza como lógica de presentación en el capitulo, la secuencia de actividades a realizar para el diseño e implementación de un modulo dentro del SIGEP vistas en el Capitulo 2, dentro de las actividades realizadas se explican las soluciones no triviales y tecnologías que se utilizaron para cada situación encontrada a partir del análisis de las funcionalidades a implementar. Toda la documentación generada como parte del proceso de diseño está anexada y referenciada en el cuerpo del documento.

Además se muestra el resultado de las validaciones de los módulos con el cliente final en sesiones de trabajo presenciales con especialistas funcionales y usuarios finales.

3.2 Análisis de las funcionalidades

A partir de las funcionalidades descritas para los módulos Educación y Trabajo en la documentación generada por la captura de requisitos resultó que las funcionalidades a implementar para ambos módulos son:

Modulo Educación

- Gestionar Actividades Educativas (Formales y No Formales)
- Gestionar Matricula de las Actividades Educativas (Formales y No Formales)
- Registrar Asistencia a las Actividades Educativas (Formales y No Formales)
- Registrar Evaluación de las Actividades Educativas (Formales y No Formales)
- Reporte: Individuos desincorporados de la actividades educativas

Modulo Trabajo

- Gestionar Unidades Productivas
- Gestionar Matricula de Unidad Productiva
- Registrar Asistencia a la Unidad Productiva
- Reporte: Individuos desincorporados de las unidades productivas

Las funcionalidades descritas como *Gestionar* realizan la inserción, eliminación y actualización de las entidades del dominio de sus respectivos módulos.

3.3 Diseño de la capa de dominio

Tanto el diagrama de clases del dominio del módulo Educación como el de Trabajo, se nutren del dominio del módulo Datos Personales del subsistema Control Penal. El diagrama de clases del dominio de Educación se muestra en el Anexo 3 y el correspondiente a Trabajo en el Anexo 4.

El diagrama de clases del dominio del módulo Trabajo tiene pocas entidades y es muy parecido al de Educación ya que ambos procesos realizan casi las mismas actividades, un aspecto a destacar en el dominio de Trabajo es que en una ocurrencia de una Unidad Productiva se puede especificar a la hora real en que llego un individuo a trabajar y a la hora que salió, pudiendo ser estas (hora llegada y hora salida) diferentes para cada individuo en una misma ocurrencia de la Unidad Productiva.

En la Figura 3.1 se muestra el diagrama de clases de la capa de dominio del módulo Trabajo.

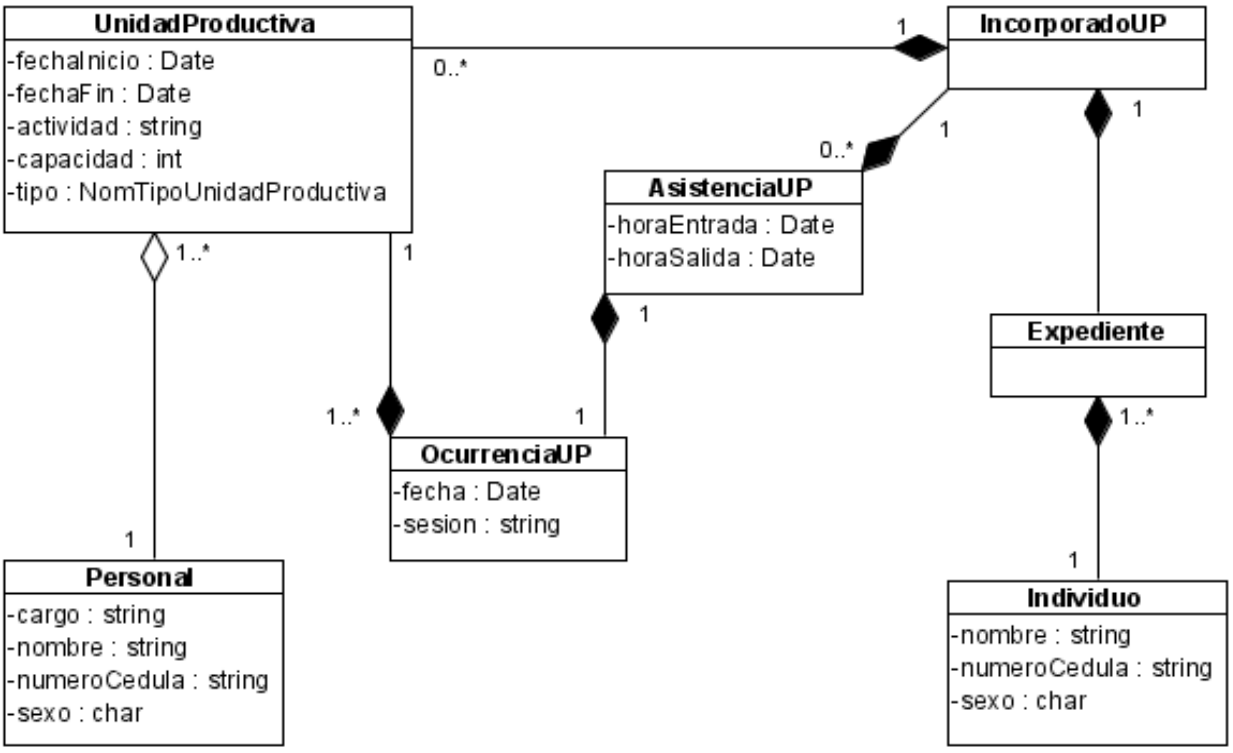


Figura 3.1 Diagrama de clases de la capa de dominio del módulo Trabajo

3.4 Diseño del modelo de datos

El modelo de datos del módulo Trabajo se representa en el Anexo 5. El modelo relativo al modulo Educación se refleja en la Figura 2.5 “Diagrama entidad-relación del módulo Educación”.

Un aspecto a destacar cuando se analiza el modelo de datos de Educación son los diferentes tipos de docentes que pueden impartir una actividad educativa (clases `DocenteExterno, DocenteInterno, DocenteIndividuo`) y que estos no son insertados al sistema cuando se va a adicionar una nueva actividad educativa, sino que estos se insertan en el sistema como parte del flujo de otros procesos como Visitas Institucionales, Administración del Personal Penitenciario o Ingreso de un individuo, procesos estos que forman parte de otros subsistemas del SIGEP. Un ejemplo de estos tipos de docentes es, para el caso de docente externo, un profesor de una universidad o de un programa social como Misión Robinson I, para el caso de docente interno seria un trabajador del sistema penitenciario venezolano y en el caso de un docente individuo se refiere a un recluso que por sus conocimientos ó nivel profesional es capaz de impartir alguna actividad educativa.

En el caso del modulo Trabajo, su modelo de datos aporta pocas tablas al modelo físico del SIGEP, sin embargo para su funcionamiento, el módulo necesita recuperar información de varias tablas del modulo Datos Personales del subsistema Control Penal.

En la Figura 3.2 se muestra el modelo de datos del módulo Trabajo.

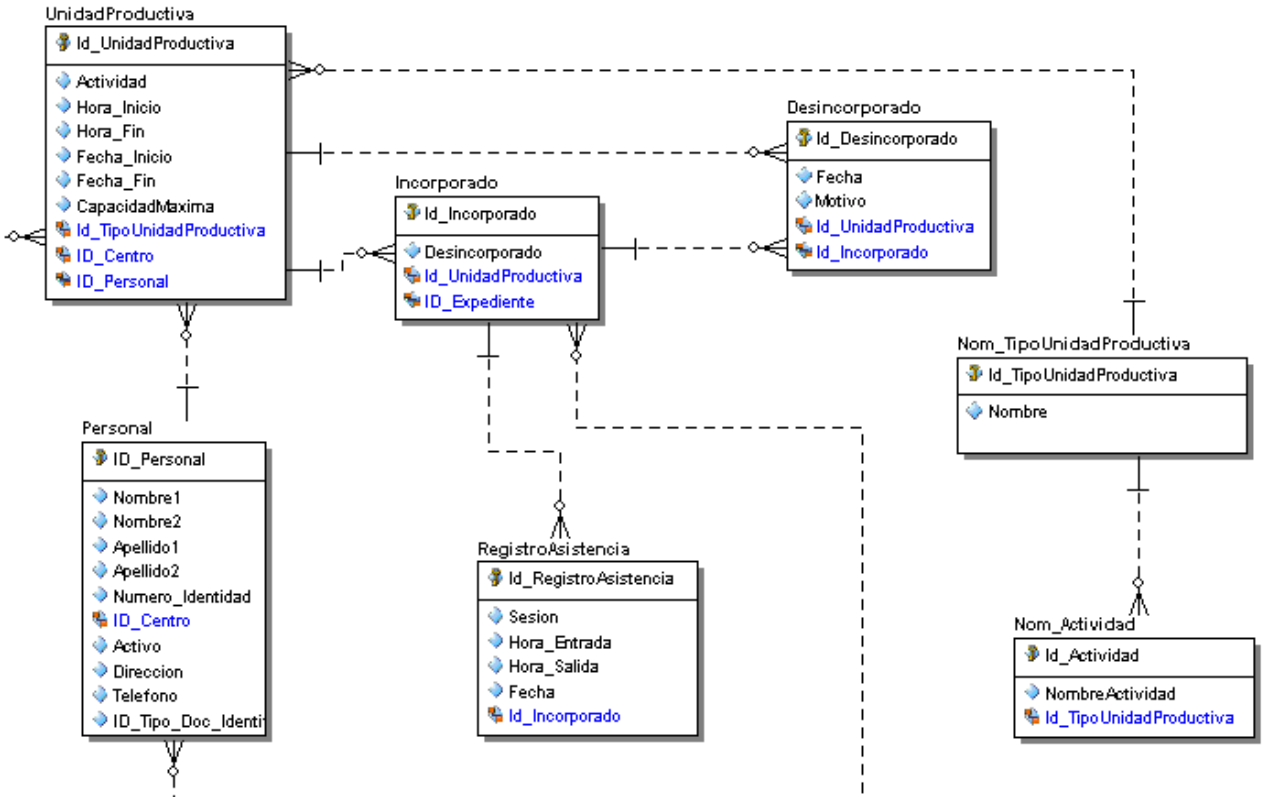


Figura 3.2 Modelo de datos del módulo Trabajo

3.5 Diseño e Implementación de la capa de negocio

Para el diseño de la capa de negocio de ambos módulos (Anexo 6) se tuvo en cuenta la relación que ellos tienen con otros módulos del SIGEP, las principales restricciones del negocio y las funcionalidades generales esperadas del sistema.

Partiendo de este análisis los aspectos más relevantes del diseño e implementación de la capa de negocio de los módulos Educación y Trabajo son: la llamada a métodos de la capa de negocio desde el cliente utilizando JSON-RPC, las transacciones y el manejo de eventos.

Llamada a métodos utilizando JSON-RPC

Tanto en Educación como en Trabajo es necesario obtener datos simples (un booleano, un entero) de la capa de negocio como parte de un proceso de validación en el cliente. Un ejemplo de esto seria, para el caso de Educación, en el momento de insertar una nueva ocurrencia a una actividad educativa (una ocurrencia es el control de la asistencia de los individuos a una actividad educativa en una fecha, hora inicio y hora fin determinadas), es necesario verificar que la nueva ocurrencia a insertar no exista ya, es decir que no existan ocurrencias que se den el mismo día a la misma hora de inicio. Para saber si dicha ocurrencia no existe ya, es necesario llamar a un método del negocio que dado la fecha y hora de inicio de la ocurrencia devuelva si ya existe una con esos datos o no, normalmente para obtener esa respuesta sería necesario que una clase controladora manejara esa petición y llamara al método del negocio, este daría un resultado y la controladora devolvería al cliente la respuesta. Con el uso de JSON-RPC este proceso sería más sencillo, rápido, e involucraría menos clases al crearse un objeto javascript que serviría de puente (*bridge*) entre el cliente y el negocio para realizar la petición de si existe o no la ocurrencia, sin necesidad de pasar por una clase controladora.

A continuación se muestra toda la configuración que se necesitaría para que dicha petición pueda ser ejecutada utilizando JSON-RPC.

Primero se declara el nombre del objeto *bridge* y dirección de la petición de dicho objeto en el fichero *loader.js,* fichero que se carga cuando se accede a la página principal de cualquier módulo. Dicho objeto realiza una petición al servidor y este le devuelve una lista de objetos donde cada objeto es una llamada a un método específico de una clase del negocio y tendría como nombre `sigep.[subsistema].[modulo].jsonrpc.[nombre-clase-negocio].[método]([parámetros])`. Cuando el objeto bridge realiza la petición se busca por todo el negocio las clases que tengan la anotación `@JSONExported(beanName = [nombre-para-acceder-a-la-clase], bridgeName = [nombre-bridge-a-buscar])` y dentro de estas los métodos que tengan la anotación `@JSONExportedMethod ,` una vez que se tienen estos datos es que crea el objeto `sigep.[subsistema].[modulo].jsonrpc.[beanName].[método]([parámetros])` y este es el

objeto que se ejecutaría desde el cliente.
A continuación se muestra las imágenes y código asociados a la explicación anterior para una mejor comprensión.

gestionarAsistencia.js | sigep-tratamiento-educacion-servlet.xml | EducacionFacadeImpl.java | loader.js | core.js

```
({
    "bridges":
    [
        {"moduleName":"administracion.seguridad","serverURL":"../administracion/seguridad/jsonrpc/jsonrpc.h
        {"moduleName":"controlpenal.ingresos","serverURL":"ingresos/ingreso/jsonrpc.htm"},
        {"moduleName":"controlpenal.egresos","serverURL":"egresos/egreso/jsonrpc.htm"},
        {"moduleName":"controlpenal.solicitudesdecisiones","serverURL":"solicitudesdecisiones/solicitudes/j
        {"moduleName":"controlpenal.datospersonales","serverURL":""},
        {"moduleName":"controlpenal.situacionjuridica","serverURL":"situacionjuridica/jsonrpc/jsonrpc.htm"}
        {"moduleName":"seguridadcustodia.common","serverURL":"../seguridadcustodia/jsonrpc.htm"},
        {"moduleName":"seguridadcustodia.gestioncupos","serverURL":"gestioncupos/jsonrpc/jsonrpc.htm"},
        {"moduleName":"seguridadcustodia.visitasfamiliares","serverURL":"visitasfamiliares/jsonrpc/jsonrpc.
        {"moduleName":"seguridadcustodia.visitasinstitucionales","serverURL":"visitasinstitucionales/jsonrp
        {"moduleName":"seguridadcustodia.pertenencias","serverURL":"pertenencias/jsonrpc/jsonrpc.htm"},
        {"moduleName":"seguridadcustodia.requisasdecomisos","serverURL":"requisasdecomisos/jsonrpc/jsonrpc.
        {"moduleName":"common.busqueda","serverURL":"../common/busqueda/criteriosRpc/jsonrpc.htm"},
        {"moduleName":"saludintegral.historiaclinica","serverURL":"historiaclinica/jsonrpc/jsonrpc.htm"},
        {"moduleName":"tratamiento.common","serverURL":"../tratamiento/jsonrpc.htm"},
        {"moduleName":"tratamiento.vinculos","serverURL":"../tratamiento/vinculos/jsonrpc.htm"},
        {"moduleName":"tratamiento.clasificacion","serverURL":"../tratamiento/clasificacion/jsonrpc.htm"},
        {"moduleName":"tratamiento.educacion","serverURL":"../tratamiento/educacion/jsonrpc/jsonrpc.htm"},
        {"moduleName":"tratamiento.trabajo","serverURL":"../tratamiento/trabajo/jsonrpc/jsonrpc.htm"},
        {"moduleName":"tratamiento.cultura","serverURL":"../tratamiento/cultura/jsonrpc/jsonrpc.htm"},
        {"moduleName":"tratamiento.deporte","serverURL":"../tratamiento/deporte/jsonrpc/jsonrpc.htm"},
        {"moduleName":"administracionpenal.capacidades","serverURL":"capacidades/jsonrpc/jsonrpc.htm"},
        {"moduleName":"administracionpenal.dotacion","serverURL":"dotacioninterno/jsonrpc/jsonrpc.htm"},
        {"moduleName":"seguridadcustodia.ubicaciones","serverURL":"ubicacion/jsonrpc/jsonrpc.htm"}
    ],
```

Figura 3.3 Fichero loader.js

Objeto de configuración que maneja la petición del objeto bridge:

```
<bean id="jsonRpcURLAgrupation"
      class="vnz.sigep.administracion.seguridad.util.urlagrupation.
                                                        URLAgrupation">
            <property name="name" value="json rpc" />
            <property name="pattern" value="/jsonrpc" />
            <property name="mappings">
                  <props>
                        <prop key="/jsonrpc.htm">
                              educacionJsonRpcController
                        </prop>
                  </props>
            </property>
</bean>
```

Objeto que se encarga de buscar todas las clases del negocio que tengan la anotación `@JSONExported(beanName = [nombre-para-acceder-a-la-clase], bridgeName = [nombre-bridge-a-buscar])` donde el atributo `bridgeName` sea igual al atributo `jsonBridge` de dicho objeto:

```
<bean id="educacionJsonRpcController"
            class="cu.uci.arbaweb.jsonrpc.web.JSONRPCController">
            <property name="jsonBridge">
                  <ref bean="bridgeEducacion"/>
            </property>                                    </bean>
```

Ejemplo de la clase `EducaciónFacadeImpl` que lleva la anotación `@JSONExported` y del método `existeOcurrencia` que lleva la anotación `@JSONExportedMethod :`

```
@JSONExported(beanName = "educacionFacade", bridgeName = "bridgeEducacion")
public class EducacionFacadeImpl implements EducacionFacade {

	...
	@JSONExportedMethod
	public Boolean existeOcurrencia(String idActividad, String fecha,
			String horaInicio) throws Exception {
		return asistenciaActividadEducativaManager.existeOcurrencia(
				idActividad, fecha, horaInicio);
	}
	...
}
```

Después de que se ejecute la búsqueda por todo el negocio y se obtengan todas las clases que tenga el *bridgeName* igual al *jsonBridge* definido en el objeto de configuración, se crean los objetos con el nombre de la clase y método a llamar y se mandan al cliente como objetos javascript para que sean utilizados cuando lo necesiten. Ejemplo de la utilización de un objeto de este tipo sería, el caso que se está tratando desde el principio de esta explicación, cuando se quiere entrar la ocurrencia a una actividad educativa que se tiene que verificar que dicha ocurrencia no existe ya. A continuación se ve la utilización de un objeto javascript que realiza una llamada a un método de la capa de negocio utilizando JSON-RPC para saber si una ocurrencia existe o no.

```
existeOcurrencia: function() {
	if (this.existeEnEliminadas())
		return false;
	var idActividad = dojo.byId("idae").value;
	var fecha=sigep.tratamiento.educacion.gestionarAsistencia.getSerializeDate
					(dojo.widget.byId("fecha").getValue());
	var hora = dojo.widget.byId("horaInicio").getValue();
	if (this.existeOcurrenciaEnNuevas())
		return true;

	/* llamada a un método del negocio desde un objeto javascript utilizando
		JSON-RPC */
	return sigep.tratamiento.educacion.jsonrpc.educacionFacade.existeOcurrencia
					(idActividad, fecha, hora);

},
```

Transacciones a nivel de negocio

Las transacciones son una parte importante en el desarrollo de aplicaciones empresariales. Las transacciones se encargan de que los recursos y datos de una aplicación se mantengan en un estado consistente, de acuerdo con las reglas de negocio. Cada método definido en la interfaz de la capa de negocio generalmente se debe ejecutar como una transacción para garantizar la consistencia de la información en la base de datos.

Para garantizar la ejecución transaccional de todas las funcionalidades implementadas en los objetos de negocio de los módulos Educación y Trabajo se utiliza las facilidades que para ello brinda el framework Spring.

La forma de gestionar las transacciones con Spring es independiente de la implementación real del código transaccional. Para aplicaciones como el SIGEP, que tienen una única fuente de datos, Spring puede usar el soporte para transacciones proporcionado por el mecanismo de persistencia, en este caso Hibernate. La clase `org.springframework.orm.hibernate3.HibernateTransactionManager` une una sesión de Hibernate al hilo de ejecución e implementa los métodos `getTransaction`, `commit` y `rollback` definidos en la interfaz `org.springframework.transaction.PlatformTransactionManager`. A través de estos tres métodos se crea una transacción o se accede a la transacción en curso y se hace `commit` y `rollback` a las transacciones. La instancia de `HibernateTransactionManager` requiere de una referencia al `sessionFactory` de Hibernate. A continuación se muestra la configuración de este objeto en el fichero de configuración del contexto de Spring.

```
<bean id="transactionManager"
      class="org.springframework.orm.hibernate3.HibernateTransactionManager">
      <property name="sessionFactory">
            <ref bean="hibernateSessionFactory" />
      </property>
</bean>
```

Spring provee de un soporte para la gestión de transacciones de forma declarativa. Las transacciones se declaran en el fichero de configuración de su contexto y son aplicadas en forma de *aspectos*. De esta forma hay que escribir muy poco o ningún código de manejo de transacciones. Con la etiqueta `<aop:advisor>` se declara el *pointcut* (punto en que se aplicará el aspecto, generalmente una expresión regular que representa un conjunto de métodos de una clase). Con la etiqueta `<tx:advice>` se declara el aspecto específico, o sea, cómo se va a aplicar la transacción. A continuación se muestra un ejemplo de la aplicación de transacciones de forma declarativa a un objeto de negocio:

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

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

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

La definición del punto donde se aplica la transacción está escrita utilizando la sintaxis de *AspectJ*[19] (http://www.eclipse.org/aspectj). La expresión `execution` significa *cuando el método sea ejecutado.* La expresión entre paréntesis representa los métodos a los que se aplicará la transacción, en este caso: `* *..*ActividadEducativaManagerImpl.*(..)`. El primer `*` significa *cualquier tipo de retorno*; la expresión `*..*ActividadEducativaManagerImpl` significa *cualquier clase cuyo nombre termine en* `ActividadEducativaManagerImpl` y la expresión `.*(..)` significa *cualquier método con cualesquiera parámetros*.

Al no especificarse el nivel de aislamiento de la transacción mediante el atributo `isolation` de la etiqueta `<tx:method>`, se toma el nivel de aislamiento por defecto del medio de almacenamiento, que en el caso de Oracle es *lectura confirmada* (*read committed*) que evita la lectura de datos sucios. Con el atributo `rollback-for` se definen las excepciones que, en caso de ser lanzadas, se desea que hagan fallar (*rollback*) la transacción.

Manejo de eventos

El SIGEP informatiza procesos que a nivel de negocio y del propio sistema desencadenan la ejecución de otros procesos en otras áreas. Ejemplo de esto es el registro del egreso de un individuo en el módulo Egresos del subsistema Control Penal que implica que los módulos Educación, Trabajo, Cultura y Deporte del subsistema Tratamiento ejecuten acciones de desincorporación del individuo de las actividades educativas, unidades productivas, actividades deportivas y culturales.

En consecuencia, es necesario que los componentes de las capas de negocio escuchen eventos que otros componentes deberán lanzar para ejecutar acciones. En el caso de los módulos Educación y Trabajo el componente que escucha el evento `EgresoIndividuoEvent` es el objeto de negocio `IndividuoEgresadoListener`, evento que es lanzado por el objeto de negocio `EgresosManagerImpl` del módulo Egresos del subsistema Control Penal.

[19] AspectJ: Lenguaje de programación orientado a aspectos, construido como una extensión del lenguaje Java.

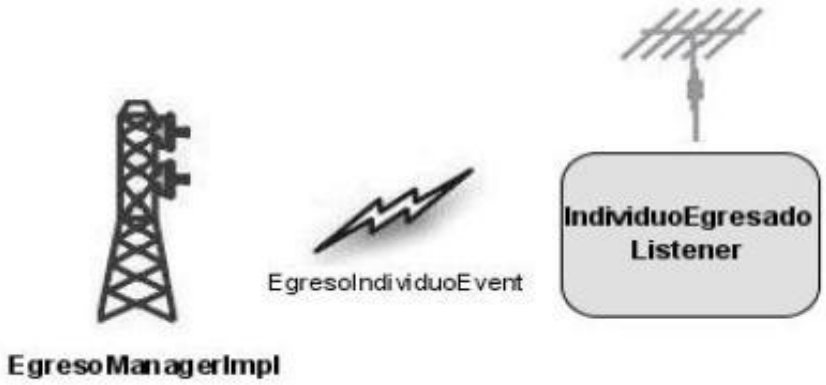


Figura 3.4 Utilización del modelo de eventos de Spring en los módulos Educación y Trabajo

Para el manejo de eventos en SIGEP se utiliza el modelo de eventos que propone el framework Spring para la implementación del patrón *Observer,* lo que conduce a que la responsabilidad del programador sea solo la implementación de la lógica de aplicación. Las desventajas del uso del modelo de eventos de Spring están en que implica un acoplamiento del código específico de la aplicación al framework, al ser necesario implementar varias de sus interfaces y que, en dicho modelo, todos los objetos interesados en escuchar eventos son notificados de todos los eventos lanzados en la aplicación; por lo que son responsables de identificar si cada evento escuchado es interesante para él. Una mejor aproximación sería que cada evento sea notificado solo a los objetos interesados en escuchar ese tipo de evento en particular. Esta aproximación puede implementarse dentro del modelo de eventos de Spring creando un manejador de eventos general que distribuya los eventos solo a los interesados, en dependencia del tipo de eventos. No obstante esta limitante no tiene impacto en el rendimiento de la aplicación porque en el SIGEP se lanzan pocos eventos.

Aunque en los módulos Educación y Trabajo solo se escuchan eventos, no se lanzan, a continuación se describe el proceso de creación, publicación y escucha de un evento en el SIGEP, para una mejor comprensión del mismo.

Creación del Evento

Después que los eventos son definidos se colocan en el subsistema común de la aplicación ya que deben ser accesibles desde distintos subsistemas. Cada evento tiene como atributo datos que a los objetos que lo escuchen brindan información relevante sobre la acción ejecutada. Las clases eventos heredan de la clase `org.springframework.context.ApplicationEvent` para insertarse en el modelo de eventos de Spring como lo muestra el siguiente fragmento de la definición del evento lanzado al egresar un individuo.

```
package vnz.sigep.common.global.events;

...
public class EgresoIndividuoEvent extends ApplicationEvent {

      private String idIndividuo;

      private Date fechaEgreso;

      private NomMotivoEgreso nomMotivoEgreso;

      public EgresoIndividuoEvent(String idIndividuo, Date fechaEgreso, ...) {

            this.idIndividuo = idIndividuo;
            ...
      }
      ...
}
```

Lanzar el evento

El ciclo de vida de los componentes de la aplicación es manejado por el contenedor de Spring, encargado de la notificación de la ocurrencia de los eventos a los objetos interesados en escucharlos. Lanzar un evento se traduce en su publicación en el contenedor de Spring a través del método `publishEvent` de la clase `org.springframework.context.ApplicationContext`. Para poder hacer esto, el objeto de negocio responsable de lanzar el evento debe ser "consciente del contenedor", o sea, debe implementar la interfaz `org.springframework.context.ApplicationContextAware`. Esta interfaz tiene un único método llamado `setApplicationContext` a través del cual se provee a los objetos que la implementen de una referencia al contenedor de Spring.

A continuación se muestra un fragmento de la implementación del objeto `EgresoManagerImpl` que muestra la forma de publicar un evento, en este caso: `EgresoIndividuoEvent`.

```
package vnz.sigep.controlpenal.egresos.manager.impl;
...
public class EgresoManagerImpl implements EgresoManager, ApplicationContextAware
{
...
      private ApplicationContext context;
      /**
      * Registra el egreso de un individuo por decisión judicial.
      */
      public void registrarEgresoPorDecisionJudicial(
                  EgresoDecisionJudicial egresoDecisionJudicial)
            throws Exception {
            ...
            // publicar el evento EgresoIndividuoEvent
            this.notificarEgresoIndividuo(individuoEgresado, fechaEgreso, ...);
      }
      ...
```

```
	/**
	* Publica el evento EgresoIndividuoEvent.
	*/
	private void notificarEgresoIndividuo(String idIndividuo, Date fecha, ...)
	{
		ApplicationEvent evento = new EgresoIndividuoEvent(idIndividuo,
		fecha, ...);
		context.publishEvent(evento);
	}
	...
	/**
	* A través de este método, el objeto se hace "consciente" del
	* contenedor de Spring.
	*/
	public void setApplicationContext(ApplicationContext context)
		throws BeansException {
		this.context = context;
	}
	...
}
```

Escuchar el evento

Los objetos interesados en escuchar eventos solo tienen que implementar la interfaz `org.springframework.context.ApplicationListener` que tiene como único método el `onApplicationEvent`, a través del cual será notificado de la ocurrencia de los eventos de la aplicación. En este método se implementa la lógica del manejo del evento. La implementación de la clase interesada (`IndividuoEgresadoListener`) en escuchar el evento `EgresoIndividuoEvent` sería como sigue:

```
package vnz.sigep.tratamiento.educacion.listener;
...
public class IndividuoEgresadoListener implements ApplicationListener {

	private EducacionFacade educacionFacade;

	public void setEducacionFacade(EducacionFacade educacionFacade) {
		this.educacionFacade = educacionFacade;
	}

	public void onApplicationEvent(ApplicationEvent event) {
		if (event instanceof EgresoIndividuoEvent) {
			try {
				educacionFacade

	.desincorporarIndividuoPorEgresoPenal((EgresoIndividuoEvent) event);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
}
```

3.6 Diseño e Implementación de la capa de acceso a datos

El diseño de la capa de acceso a datos de ambos módulos se muestra en el Anexo 7.

Es necesario en algún momento mostrar la cantidad de horas dedicadas a actividades educativas o productivas que tiene un individuo para poder realizar resúmenes de horas dedicadas a estas actividades a nivel de centro o para darle una mejora o reducción de la sanción a un individuo.

Decidir cuales individuos pueden obtener una mejora o reducción de la sanción, así como tener estadísticas a nivel de centro de las horas dedicadas a actividades educativas o productivas, constituye un aporte esencial del SIGEP de ayuda a sus usuarios pues sustituye un tedioso proceso manual de revisión. Para obtener estos datos se determino la implementación de funciones, que radicaran en el gestor de base de datos, de manera que se pudiese garantizar un acceso rápido a esa información, teniéndose en cuenta que son consultadas varias tablas, algunas de las cuales pueden contener miles de registros en crecimiento exponencial en el tiempo de implantación del sistema. En este caso, la utilización de funciones que aprovechan el poder del gestor de base de datos Oracle, redujo considerablemente el tiempo de respuesta del sistema. En la Figura 2.7 se representa la utilización de las funciones implementadas desde los objetos de la capa de acceso a datos del módulo Educación.

La implementación de ambos diseños se auxilió de las facilidades brindadas por el framework Hibernate como el lenguaje HQL y el API Criteria como ya se ha especificado en capítulos anteriores.

3.7 Diseño e Implementación de la capa de interfaz de usuario

Para la realización del diseño y la implementación de la capa de interfaz de usuario en el proyecto SIGEP, se debe tener dominio de un conjunto de tecnologías entre las que se encuentran: HTML, JavaScript, JSP y Spring-MVC. Para la implementación de esta capa en los módulos Educación y Trabajo fue necesaria la utilización de todas estas tecnologías de acuerdo a los requisitos visuales del usuario del sistema.

A continuación se detallan las soluciones más significativas en el diseño e implementación de esta capa. Toda la documentación generada por esta actividad se recoge en el Anexo 8**.**

3.7.1 Diseño de la funcionalidad Registrar Actividad Educativa

Para el diseño de la funcionalidad *Registrar Actividad Educativa* (Ver Anexo 8.1.1) se encontraron características especiales. Se debe brindar la posibilidad al usuario de registrar actividades educativas de dos tipos, con datos específicos para cada una, en relación al tipo de actividad. Por ejemplo, de la actividad educativa formal interesa registrar el tipo de actividad formal y el grado o nivel de la misma; de la actividad educativa no

formal se necesita registrar el tipo de actividad no formal, el nombre y la cantidad de horas que tendrá.

En el momento en que se diseña el módulo Educación no se tiene conocimiento exacto si las actividades educativas serán solamente de dos tipos y de los datos que se necesitaran registrar de cada una de ellas y, aunque no es un requisito permitir crear nuevos tipos de actividades educativas en tiempo de ejecución, sí se necesita hacer un diseño lo suficientemente flexible que permitiera incluir, en futuras iteraciones, nuevos tipos de actividades educativas sin modificar el código escrito anteriormente.

La aparición de nuevos tipos de actividades educativas para la capa de acceso a datos no implicaría cambios en el código escrito en iteraciones anteriores, solo se ejecutaría el mapeo de la nueva entidad en la configuración de Hibernate. Para la capa de negocio es transparente el hecho de que existan diferentes tipos de actividades educativas y que se crearán nuevas en el futuro pues esta capa solo conoce la base de la jerarquía de las actividades educativas y delega en la capa de acceso a datos la responsabilidad de conocer la representación de los datos en el medio de almacenamiento.

En la capa de presentación debe programarse un “Formulario para el registro de las actividades educativas”, en efecto la utilización de *formularios* es muy común en aplicaciones web. Para la gestión del formulario se define un controlador que hereda de una clase que Spring-MVC brinda con ese fin (Figura 3.5). Este controlador es el responsable de construir la página del formulario con los datos que este necesite y de recolectar los datos introducidos por el usuario y con ellos crear una instancia de la actividad educativa correspondiente para ser enviada a la capa de negocio.

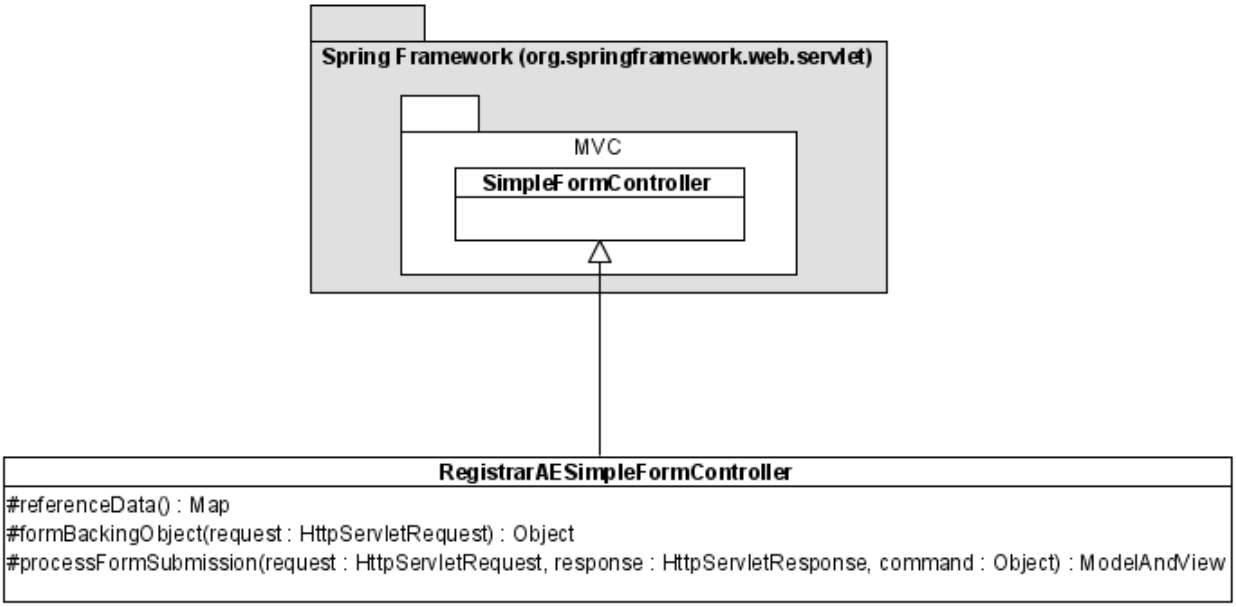


Figura 3.5 Uso de la clase SimpleFormController en la creación de formularios

El framework Spring para el caso específico de clases controladoras que manejen a páginas que utilicen formularios, brinda la clase `SimpleFormController`, de la cual los métodos más utilizados son `processFormSubmission`, `formBackingObject` y `referenceData`. En el método `processFormSubmission`, el controlador debe crear una instancia de una Actividad Educativa en dependencia del tipo que se esté registrando y mandarla a la capa de negocio. En el método `formBackingObject`, el controlador debe crear el objeto de respaldo (*command* en el léxico de Spring-MVC) que contiene los datos a mostrar en la página del formulario, estos datos dependen del tipo de actividad educativa que se esté registrando. En el método `referenceData`, el controlador debe devolver datos adicionales que se necesitan para la construcción del formulario a mostrar.

En el siguiente diagrama de secuencia se muestra la solución propuesta para la funcionalidad *Registrar Actividad Educativa*.

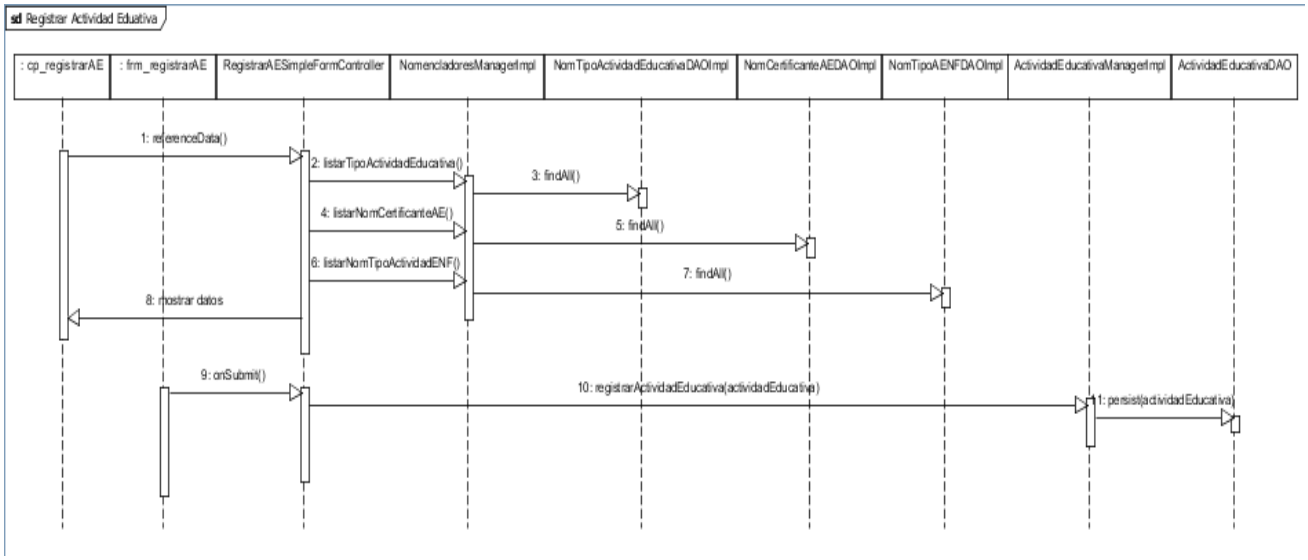


Figura 3.6 Diagrama de secuencia de la funcionalidad registrar actividad educativa

Dado el diseño propuesto a continuación se muestra como seria la implementación del mismo para la funcionalidad *Registrar Actividad Educativa*.

Implementación de la clase controladora RegistrarAESimpleFormController

En la clase controladora que maneja el formulario para registrar una nueva actividad educativa, es necesario implementar los métodos `referenceData` y `processFormSubmission`, el primero es necesario porque es el que va a brindar al formulario los datos necesarios para poder realizar el registro de una actividad (ejemplo: tipo de actividad educativa no formal, tipo de certificante, etc.) y se ejecuta cuando se va a mostrar el formulario antes de introducirles los datos de una actividad educativa. La

implementación de este método en la clase `RegistrarAESimpleFormController` quedaría de la siguiente manera:

```
package vnz.sigep.tratamiento.educacion.web;
...
public class RegistrarAESimpleFormController extends SimpleFormController {
    ...
    protected Map<String, Object> referenceData() throws Exception {
      Map<String, Object> model = new HashMap<String, Object>();
      model.put("tipoAE", nomencladoresFacade.listarTipoActividadEducativa());
      model.put("certificantes", nomencladoresFacade.listarNomCertificanteAE());
      model.put("tipoAENoFormal",
                            nomencladoresFacade.listarNomTipoActividadENF());
      return model;
    }
    ...
}
```

El otro método que es necesario implementar en la clase controladora que maneja el formulario para registrar una nueva actividad educativa es `processFormSubmission`, este es el método que se ejecuta cuando se quiere registrar una nueva actividad educativa y en el mismo se crea una instancia de la clase `ActividadEducativa` con los datos enviados desde el formulario, para enviársela a la capa de negocio quien se encargara de ejecutar la lógica de registrar una actividad educativa. Ejemplo de la implementación de este método seria:

```
package vnz.sigep.tratamiento.educacion.web;
...
public class RegistrarAESimpleFormController extends SimpleFormController {
    ...
    protected ModelAndView processFormSubmission(HttpServletRequest request,
                HttpServletResponse response, Object command) throws Exception
    {

        ActividadEducativaCommand actividadEducativaCommand =
                              (ActividadEducativaCommand) command
        ActividadEducativa actividadEducativa;
        if (actividadEducativaCommand.getIdTipo().equals("formal")) {
            actividadEducativa = new ActividadEducativaFormal();
            if (!actividadEducativaCommand.getIdGrado().equals("-1")) {
                Grado grado = new Grado();
                grado.setId(actividadEducativaCommand.getIdGrado());
                ((ActividadEducativaFormal)ActividadEducativa).setGrado(grado);
            }
            if (!actividadEducativaCommand.getIdCertificante().equals("-1")) {
                NomCertificanteAE certificante = new NomCertificanteAE(
                actividadEducativaCommand.getIdCertificante());
                ((ActividadEducativaFormal)
                            ActividadEducativa).setCertificante(certificante);
            }
        } else {
            actividadEducativa = new ActividadEducativaNoFormal();
            if (!actividadEducativaCommand.getIdTipoNoFormal().equals("-1")) {
                  NomTipoActividadENF tipoActividadENF = new
                                                    NomTipoActividadENF();
```

```
                tipoActividadENF.setId(actividadEducativaCommand.
                                                    getIdTipoNoFormal());
                ((ActividadEducativaNoFormal) ActividadEducativa)
                                        .setTipoNoFormal(tipoActividadENF);
            }
            ((ActividadEducativaNoFormal) actividadEducativa).
                            setNombre(actividadEducativaCommand.getNombre());
            if(StringUtils.hasText(actividadEducativaCommand.getCantidadHoras()))
                ((ActividadEducativaNoFormal) actividadEducativa).
                    setCantidadHoras(Integer.parseInt(
                    actividadEducativaCommand.getCantidadHoras()));
        }

        If (StringUtils.hasText(actividadEducativaCommand.getCapacidad()))
        actividadEducativa.setCapacidad(Integer.parseInt(
                            actividadEducativaCommand.getCapacidad()));
        NomTipoActividadEducativa tipoActividadEducativa = new
                                                NomTipoActividadEducativa();
        tipoActividadEducativa.setId(actividadEducativaCommand.getIdTipo());
        actividadEducativa.setTipo(tipoActividadEducativa);
        actividadEducativa.setDescripcion(actividadEducativaCommand.
                                                        getDescripcion());
        actividadEducativa.setInstitucion(actividadEducativaCommand.
                                                        getInstitucion());
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        if (!actividadEducativaCommand.getFechaFin().equals(""))
            actividadEducativa.setFechaFin(sdf.parse
                                (actividadEducativaCommand.getFechaFin()));
        if (!actividadEducativaCommand.getFechaInicio().equals(""))
            actividadEducativa.setFechaInicio(sdf.parse
                            (actividadEducativaCommand.getFechaInicio()));

        if (!actividadEducativaCommand.getIdDocente().equals("")) {
            if (actividadEducativaCommand.getTipoDocente().equals("1"))
                actividadEducativa.setDocente(new DocenteInterno(null,
                        new Personal(actividadEducativaCommand.getIdDocente())));
            if (actividadEducativaCommand.getTipoDocente().equals("2"))
                actividadEducativa.setDocente(new DocenteIndividuo(null, new
                   Expediente(new Individuo(actividadEducativaCommand.
                                                        getIdDocente()))));
            if (actividadEducativaCommand.getTipoDocente().equals("3"))
                actividadEducativa.setDocente(new DocenteExterno(null, new
              VisitanteInstitucional(actividadEducativaCommand.getIdDocente())));
        }

        if (actividadEducativaCommand.getId().equals("") ||
                                actividadEducativaCommand.getId() == null)

            educacionFacade.registrarActividadEducativa(actividadEducativa);

            return null;
    }
    ...
}
```

3.8 Integración de los módulos Educación y Trabajo al SIGEP

Una vez concluida la implementación y prueba por el equipo de calidad de los módulos Educación y Trabajo, estos se empaquetan en ficheros *JAR*[20] y se declaran componentes estables del SIGEP para ser sometidos a pruebas de aceptación con el cliente y así pasar a formar parte de la línea base del software.

En la Figura 3.7 se muestra el diagrama de componentes de Tratamiento donde los módulos Educación y Trabajo ya empaquetados se comunican con otros módulos a través de interfaces.

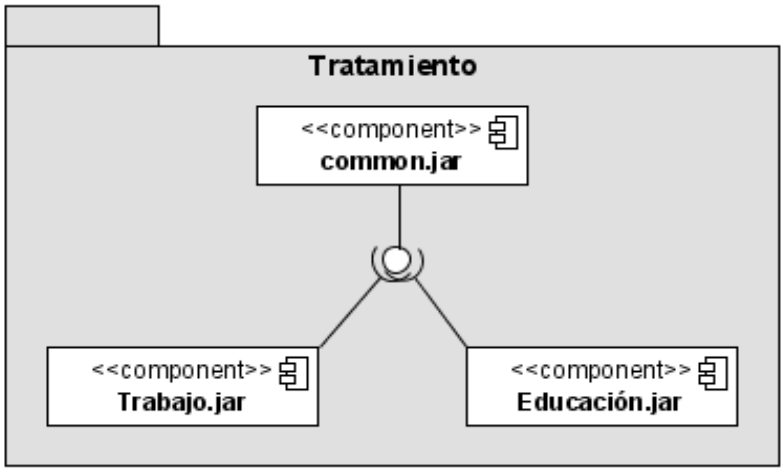


Figura 3.7 Educación y Trabajo como componentes del SIGEP

3.9 Resultados de las validaciones con el cliente

Los módulos Educación y Trabajo del SIGEP han sido verificados en dos ocasiones en pruebas de aceptación con usuarios finales y especialistas funcionales y una más, realizada en prueba piloto en un ambiente real en el Centro Penal la Mínima de Carabobo del estado Valencia. Los módulos pertenecen a la primera versión del SIGEP que se encuentra desplegada en el centro mencionado y en la Dirección General de Servicios Penitenciarios de la República Bolivariana de Venezuela desde Junio de 2008.

Los resultados en todas las pruebas realizadas han sido satisfactorios, no obstante, en todos los casos se han detectado no conformidades por parte de los clientes lo que se representa en la Figura 3.8. La mayoría de estas no conformidades han estado relacionadas con las interfaces gráficas de los módulos.

[20] JAR, del inglés *Java Archive*: Tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje Java.

En sentido general los usuarios, especialistas funcionales e informáticos de la institución han quedado satisfechos con el trabajo realizado.

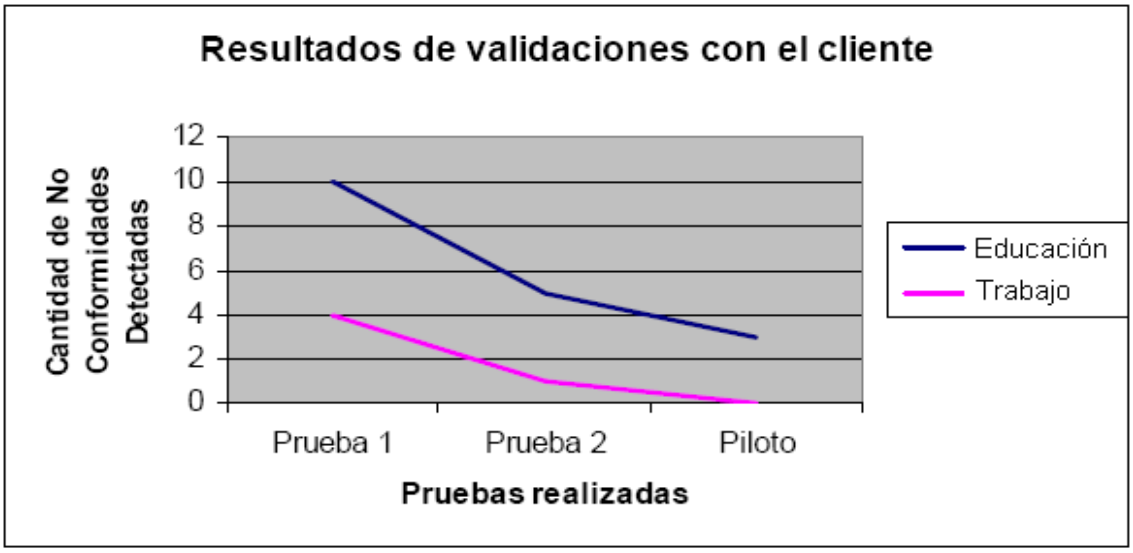


Figura 3.8 Resultados de validaciones con el cliente

3.10 Conclusiones

El diseño e implementación de los módulos Educación y Trabajo se realizó siguiendo el flujo de trabajo definido para el SIGEP.

En varios casos se encontraron situaciones que llevaron al límite la arquitectura definida, que condujeron a la toma de importantes decisiones de diseño en función del rendimiento. Se utilizaron patrones de diseño para promover la flexibilidad y posibilitar la facilidad de extensión.

Se implementaron funcionalidades de gran impacto para el cliente como el cálculo de las horas dedicadas por un individuo a actividades educativas o productivas, además del proceso de desvincular a un individuo automáticamente de dichas actividades cuando es egresado del sistema.

CONCLUSIONES

Como resultado del presente trabajo, se realizó el diseño y la implementación de los módulos Educación y Trabajo, además de la integración de estos al SIGEP. Ambos módulos fueron probados con resultados satisfactorios en condiciones reales de trabajo en el centro penitenciario Mínima de Carabobo durante el piloto de la segunda versión del SIGEP.

Con los artefactos generados en las actividades realizadas se lograron cumplir los objetivos propuestos para este trabajo. Se logró diseñar e implementar el conjunto de requisitos definido para ambos módulos permitiendo puntos de extensión para futuras iteraciones del software.

RECOMENDACIONES

Se recomienda la realización de un monitoreo constante del rendimiento de los módulos en la medida en que la cantidad de datos que el sistema maneja vaya creciendo, puesto que actualmente el sistema está probado para una cantidad limitada de datos que no ponen al límite la solución propuesta.

BIBLIOGRAFÍA

ARIAS, ARTURO .C y KNIGHT, HUMBERTO. Descripción preliminar de la solución de software propuesta. UCI, 2005

ARIAS, ARTURO. C. Proyecto Técnico de Asesoría Especializada, Colaboración Médica Odontológica, Comunicación Institucional y Solución Tecnológica para apoyar la modernización del Sistema Penitenciario de la República Bolivariana de Venezuela. ALBET, 2006.

BETANCOURT, ARACELIS. Descripción del proceso de negocio. Subsistema Clasificación y Tratamiento. Módulo Educación, ALBET SIGEP, 2007.

BETANCOURT, ARACELIS. Descripción de funcionalidades. Subsistema Clasificación y Tratamiento. Módulo Educación, ALBET SIGEP, 2008.

VEGA, YANET. Descripción del proceso de negocio. Subsistema Clasificación y Tratamiento. Módulo Trabajo, ALBET SIGEP, 2007.

BETANCOURT, ARACELIS. Descripción de funcionalidades. Subsistema Clasificación y Tratamiento. Módulo Trabajo, ALBET SIGEP, 2008.

BUSCHMANN, FRANK y otros. Pattern-Oriented Software Architecture, Wiley, 2001.

KAISLER, STEPHEN. H. Software Paradigms. 2005.

GOF. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

JACOBSON, IVAR.; BOOCH, GRADY y otros. El Proceso Unificado de Desarrollo de Software. Empresa Poligráfica de Holguín, 2004. 435 p.

ORACLE CORPORATION. Oracle Database Documentation Library. Oracle, 2005.

OSORIO, MANUEL. Diccionario de Ciencias Jurídicas, Políticas y Sociales. Heliasta,1997.

PÉREZ, IÓSEV y GONZÁLEZ, LUIS. A. ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB.UCI, 2007 p.

PERRONE, PAUL J. y otros. Building Java Enterprise Systems with J2EE, 2000.

RATIONAL SOFTWARE CORPORATION, Rational Unified Extended Help, 2003.

WALLS, CRAIG y BREINDENBACH, RYAN. Spring in Action, Second Edition, 2007.

ANEXOS

Anexo 1. Implementación del método *registrarActividadEducativa* del manager ActividadEducativaManagerImpl del modulo Educación.

```
public void registrarActividadEducativa(
                ActividadEducativa actividadEducativa) throws Exception {

      InputAssert.notNull(actividadEducativa,
                "actividad educativa es requerido");
      InputAssert.notNull(actividadEducativa.getDocente(),
                "actividadEducativa.docente es requerido");
      InputAssert.notNull(actividadEducativa.getTipo(),
                "actividadEducativa.tipo es requerido");
      InputAssert.hasText(actividadEducativa.getTipo().getId(),
                "actividadEducativa.tipo es requerido");

      // poner centro actual
      actividadEducativa.setCentro(centroManager.obtenerCentro());

      ActividadEducativa actividadEducativaBD = null;

      if (!StringUtils.hasText(actividadEducativa.getId())) {
            // persistir actividad educativa
            actividadEducativaBD = actividadEducativaDAO
                        .persist(actividadEducativa);
            // persistir docente
            Docente docente = actividadEducativa.getDocente();
            docente.setActividadEducativa(actividadEducativaBD);
            docenteDAOs.get(docente.getClass()).persist(docente);
      } else /*si la actividad educativa tiene id, se esta actualizando*/{
            actividadEducativaBD = actividadEducativaDAO.findById(
                        actividadEducativa.getId(), false);
            // actualizar datos
            actividadEducativaBD.setTipo(actividadEducativa.getTipo());
            actividadEducativaBD.setDescripcion(actividadEducativa
                        .getDescripcion());
            actividadEducativaBD.setInstitucion(actividadEducativa
                        .getInstitucion());
            actividadEducativaBD.setFechaInicio(actividadEducativa
                        .getFechaInicio());
            actividadEducativaBD.setFechaFin(actividadEducativa.getFechaFin());
            actividadEducativaBD
                              .setCapacidad(actividadEducativa.getCapacidad());
            if (actividadEducativaBD.getTipo().getId().equals(
                        getIdActividadEducativaFormal())) {
                  // actualizar datos formal
                  ((ActividadEducativaFormal) actividadEducativaBD)
                                    .setGrado(((ActividadEducativaFormal)
                                          actividadEducativa).getGrado());
                  ((ActividadEducativaFormal) actividadEducativaBD)
                        .setCertificante(((ActividadEducativaFormal)
                              actividadEducativa).getCertificante());
            } else if (actividadEducativaBD.getTipo().getId().equals(
                  getIdAactividadEducativaNoFormal())) {
                  // actualizar datos no formal
                  ((ActividadEducativaNoFormal) actividadEducativaBD)
                        .setTipoNoFormal(((ActividadEducativaNoFormal)
```

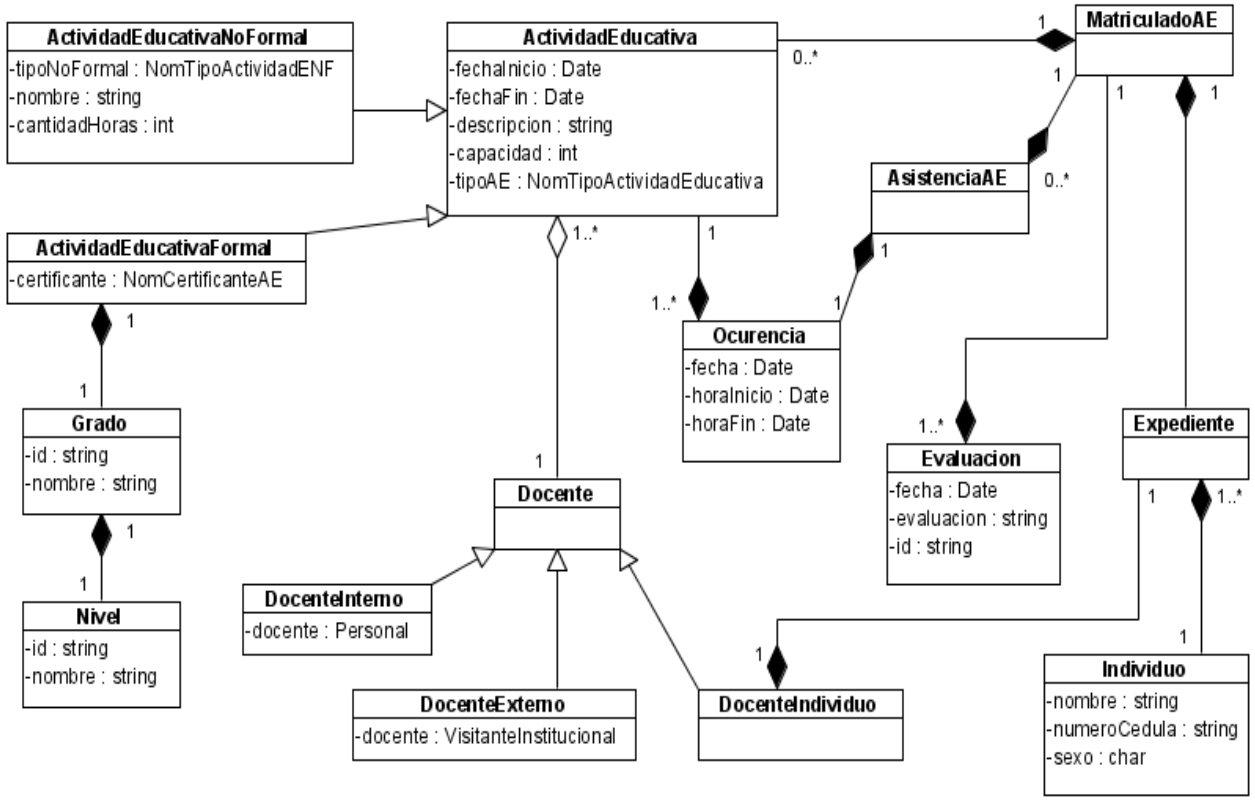
```
                        actividadEducativa).getTipoNoFormal());
                ((ActividadEducativaNoFormal) actividadEducativaBD)
                        .setNombre(((ActividadEducativaNoFormal)
                                actividadEducativa).getNombre());
                ((ActividadEducativaNoFormal) actividadEducativaBD)
                                .setCantidadHoras(((ActividadEducativaNoFormal)
                                        actividadEducativa).getCantidadHoras());
        }
        // eliminar docente anterior (si es distinto del docente actual)
                Docente docenteAnterior = actividadEducativaBD.getDocente();
                Docente docente = actividadEducativa.getDocente();
                docente.setActividadEducativa(actividadEducativaBD);
                if (!docente.equals(docenteAnterior)) {
                        if (docenteAnterior != null) {
                                docenteDAOs.get(docenteAnterior.getClass()).delete(
                                                docenteAnterior);
                        }
                        // poner docente actual
                        actividadEducativaBD.setDocente(docente);
                        docenteDAOs.get(docente.getClass()).persist(docente);
                }
        }
}
```

Anexo 2. Implementación del controlador *MatricularIndividuosAEController*

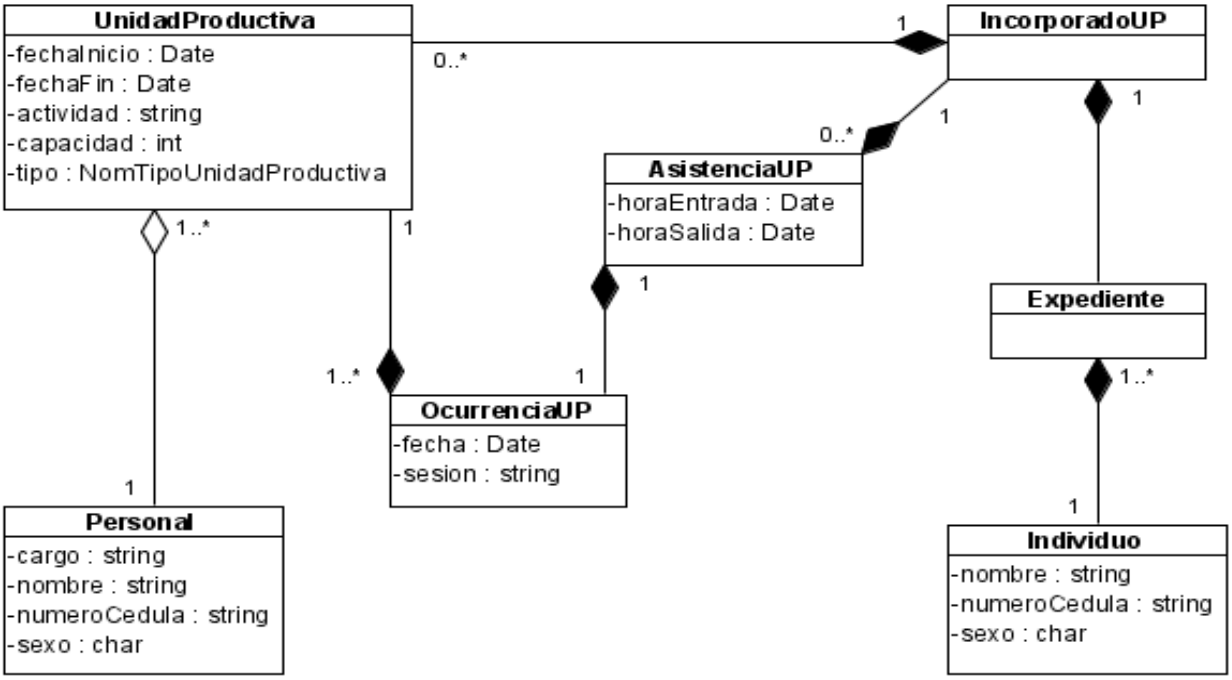
```
public class MatricularIndividuosActividadesEducativasController extends
            AbstractController {

      private EducacionFacade educacion;

      public void setEducacion(EducacionFacade educacion) {
            this.educacion = educacion;
      }

      @Override
      protected ModelAndView handleRequestInternal(HttpServletRequest request,
                  HttpServletResponse response) throws Exception {

            JSONArray added = JSONArray.fromString(request.
                                                getParameter("added"));
            String actividad = request.getParameter("actividad");
            ActividadEducativa actividadEducativa = new ActividadEducativa(
                                                            actividad);

            List<MatriculadoActividadEducativa> matriculados = new
                                    ArrayList<MatriculadoActividadEducativa>();
            for (Integer i = 0; i < added.length(); i++) {
                  MatriculadoActividadEducativa matriculado = new
                                            MatriculadoActividadEducativa();
                  String idIndividuo = added.getJSONObject(i).getString("id");
                  Expediente expediente = null;
                  try {
                        expediente = educacion.buscarExpedienteActivoPorIndividuo
                              (new Individuo(idIndividuo));
                  } catch (Exception e) {
                        response.getWriter().write("{error:1, errorMsg: ''}");
                        return null;
```

```
                }
                matriculado.setExpediente(expediente);
                matriculado.setActividadEducativa(actividadEducativa);
                if (!educacion.estaMatriculado(matriculado)) {
                      matriculados.add(matriculado);
                }
          }
          try {
                if (matriculados.size() > 0)
                      educacion.matricularIndividuos(matriculados);
          }catch (IncorporacionActividadEducativaException e) {
                e.printStackTrace();
                response.getWriter().write("{error:2, errorMsg: '" +
                                                e.getMessage() + "'}");
                return null;
          } catch (Exception e) {
                e.printStackTrace();
                response.getWriter().write("{error:3, errorMsg: 'Error
                                                desconocido'}");
                return null;
          }

          response.getWriter().write("{error:0}");
          return null;
    }
}
```

Anexo 3. Diagrama de clases del dominio del módulo Educación

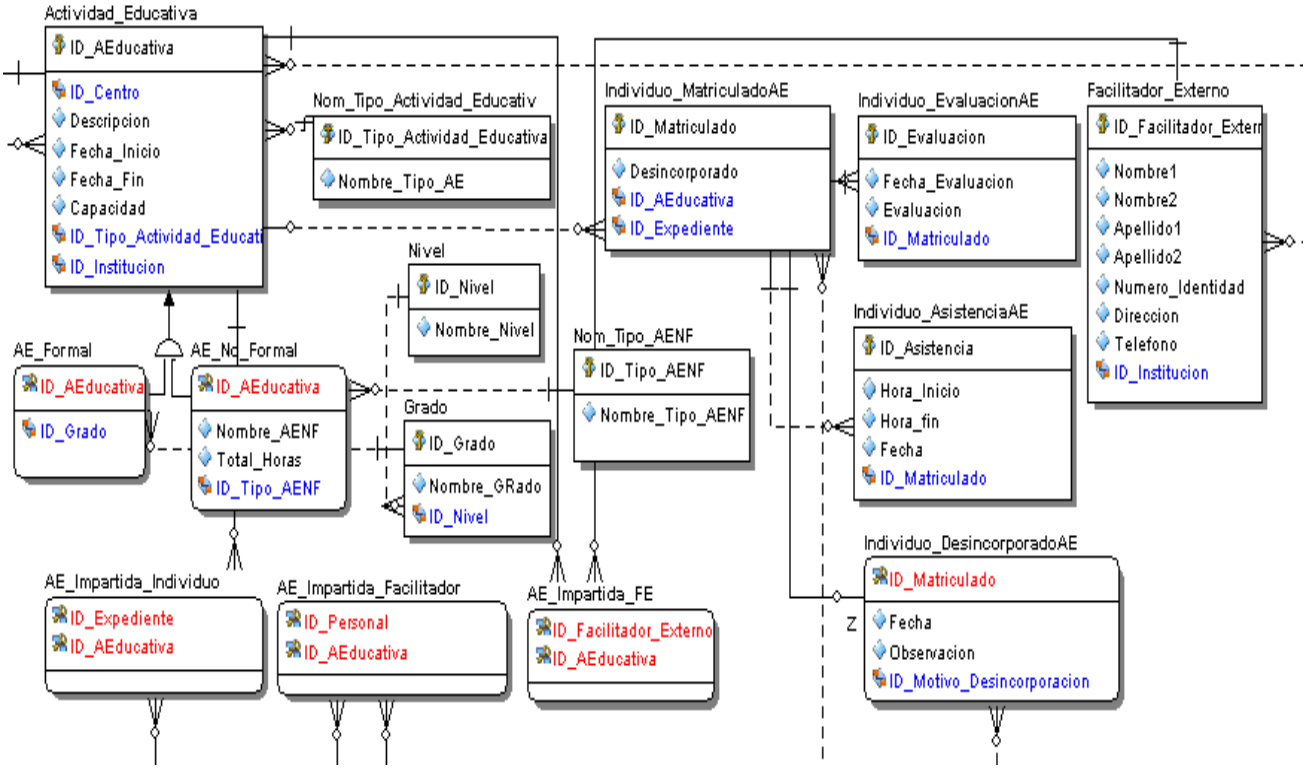


Anexo 4. Diagrama de clases del dominio del módulo Trabajo

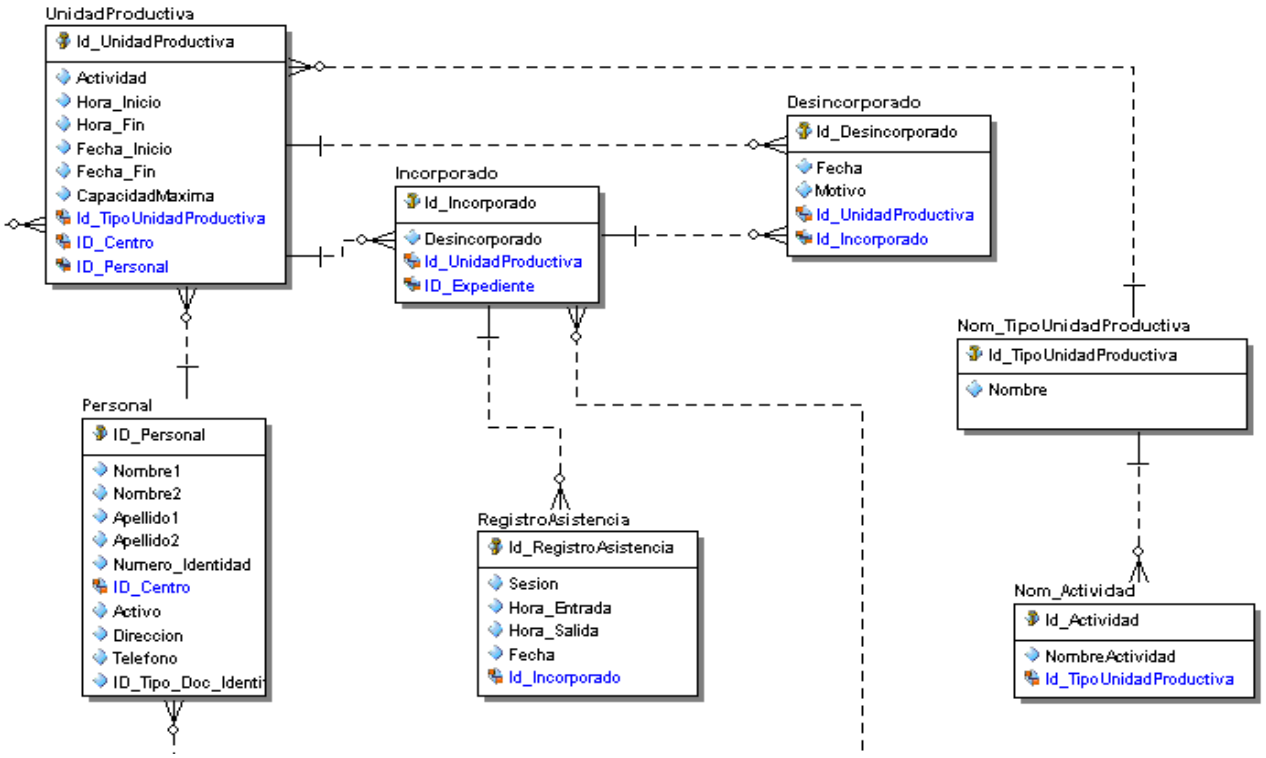


Anexo 5. Diseño del Modelo de datos de los módulos Educación y Trabajo

5.1 Modelo de datos de Educación

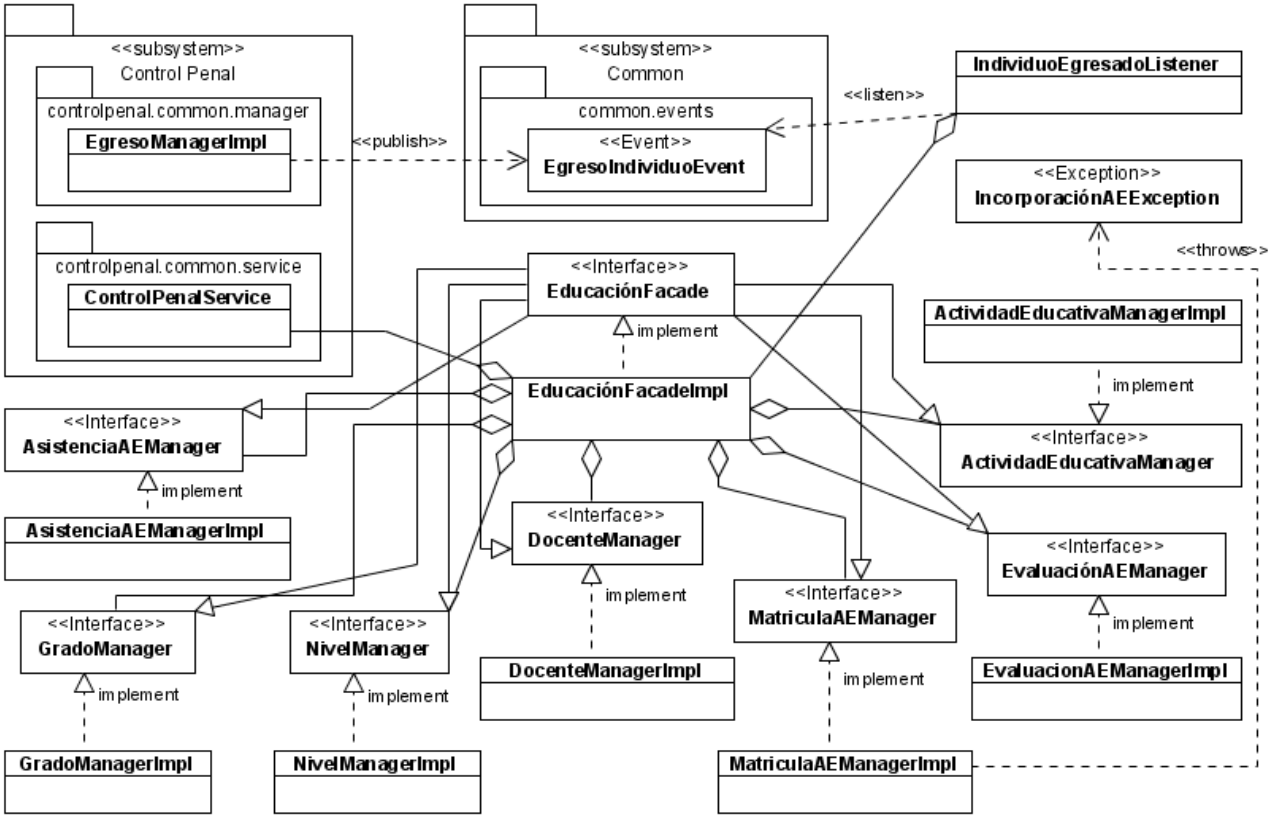


5.2 Modelo de datos de Trabajo

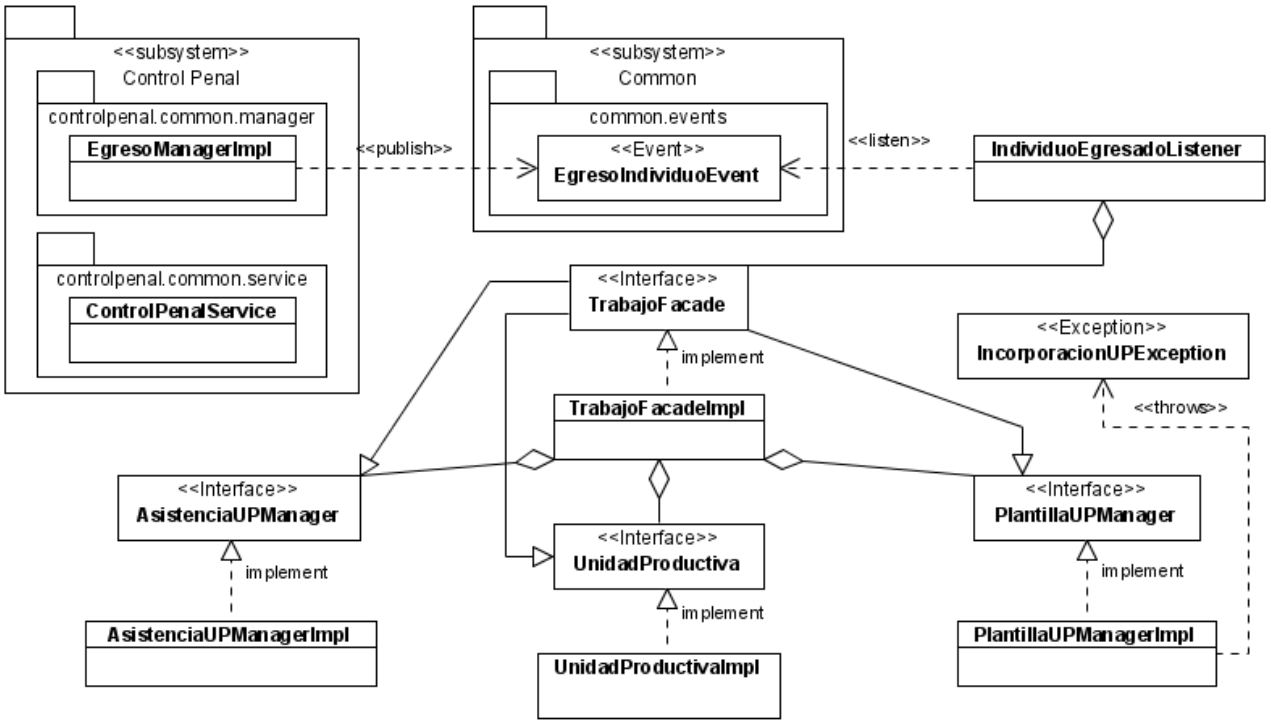


Anexo 6. Diseño de la capa de negocio de los módulos Educación y Trabajo

6.1 DCD: Capa de negocio Educación

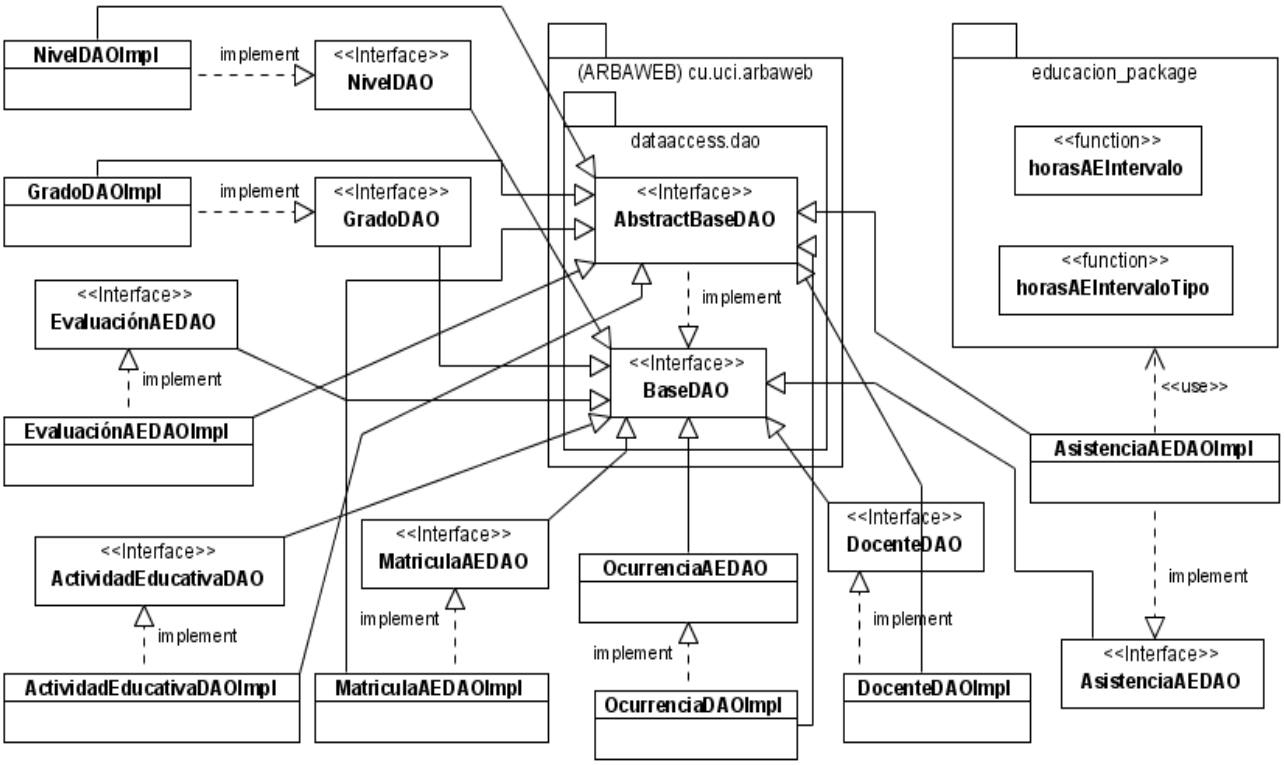


6.2 DCD: Capa de negocio Trabajo

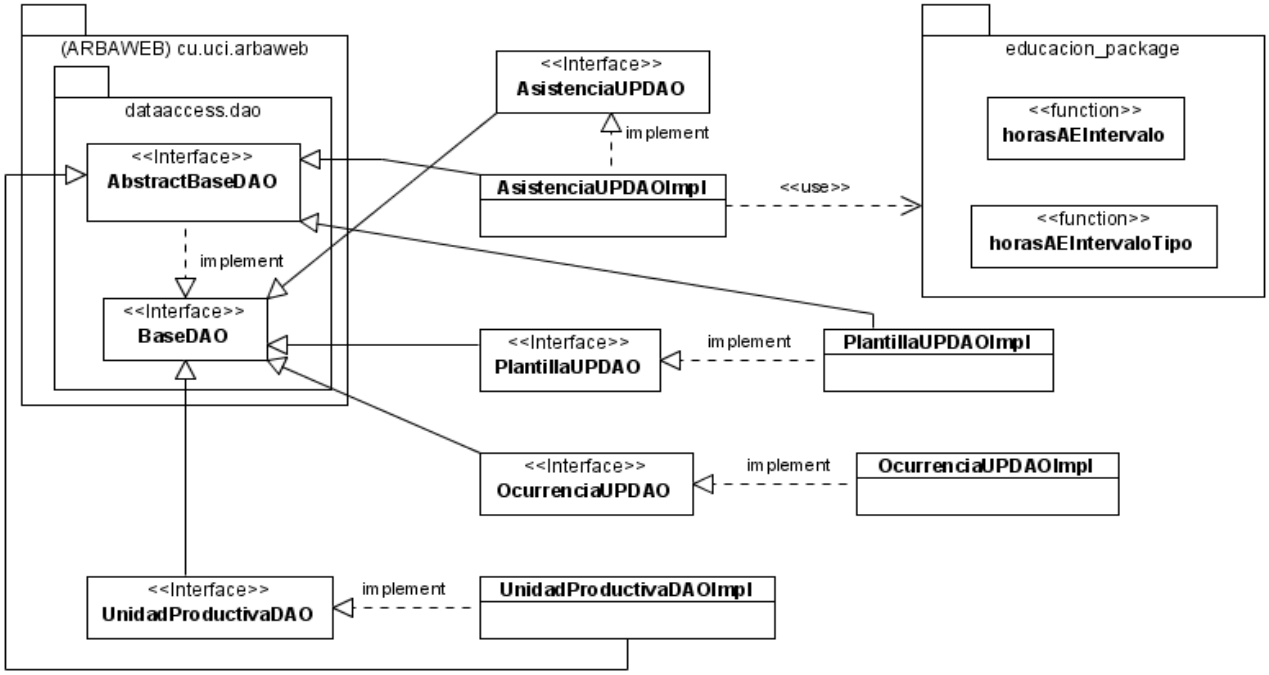


Anexo 7. Diseño de la capa de acceso a datos de los módulos Educación y Trabajo

7.1 DCD: Capa de Acceso a Datos Educación



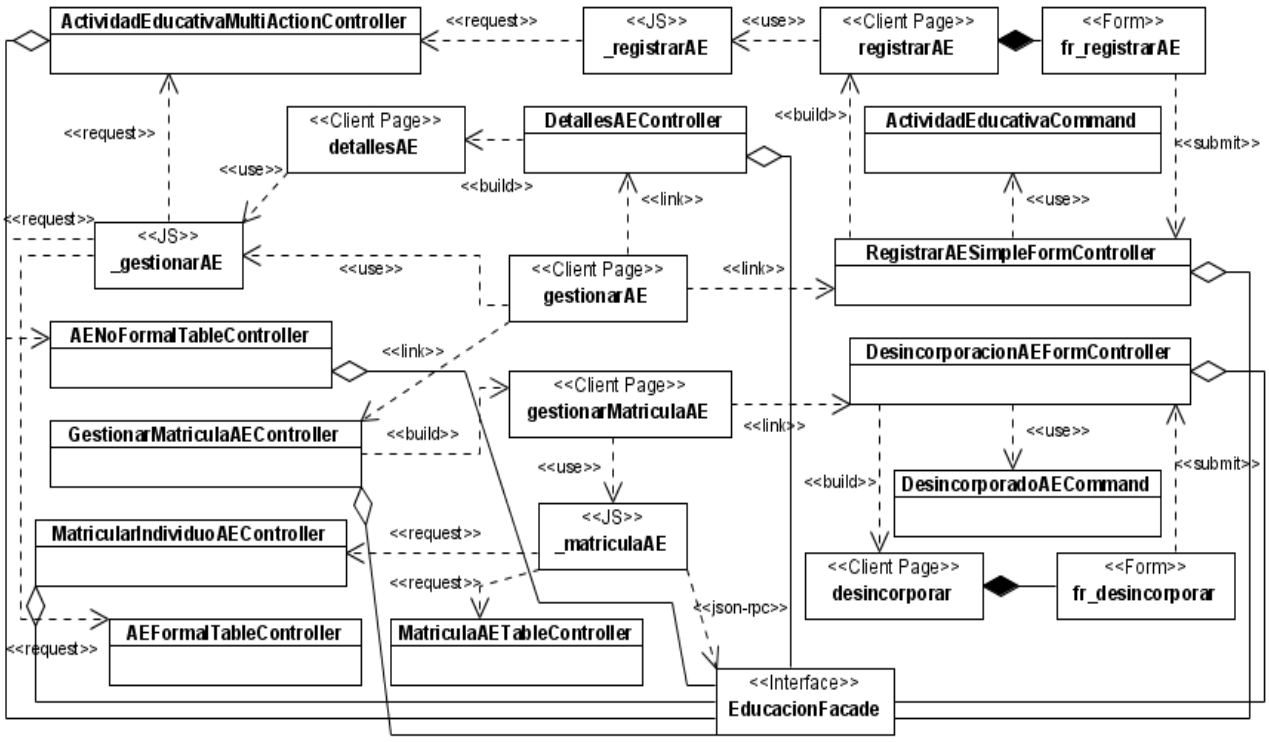
7.2 DCD: Capa de Acceso a Datos Trabajo



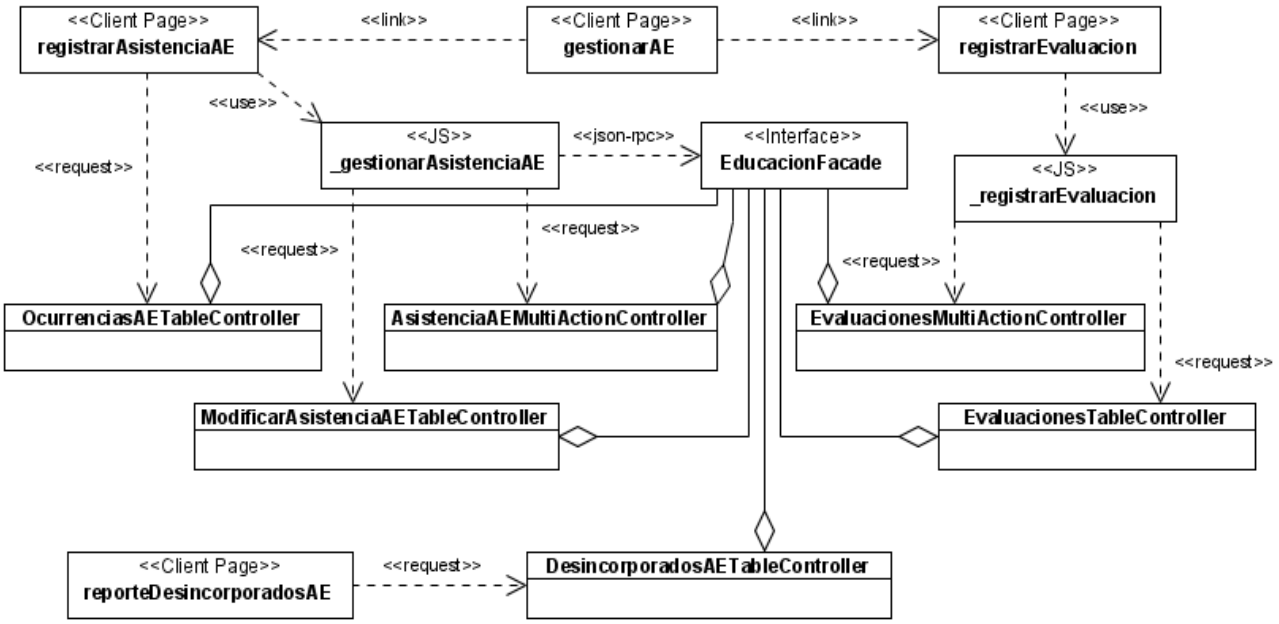
Anexo 8. Diseño de la capa de presentación de los módulos Educación y Trabajo

8.1 Capa de presentación del módulo Educación

8.1.1 DCD: Funcionalidades Gestionar Actividades Educativas y Matricula

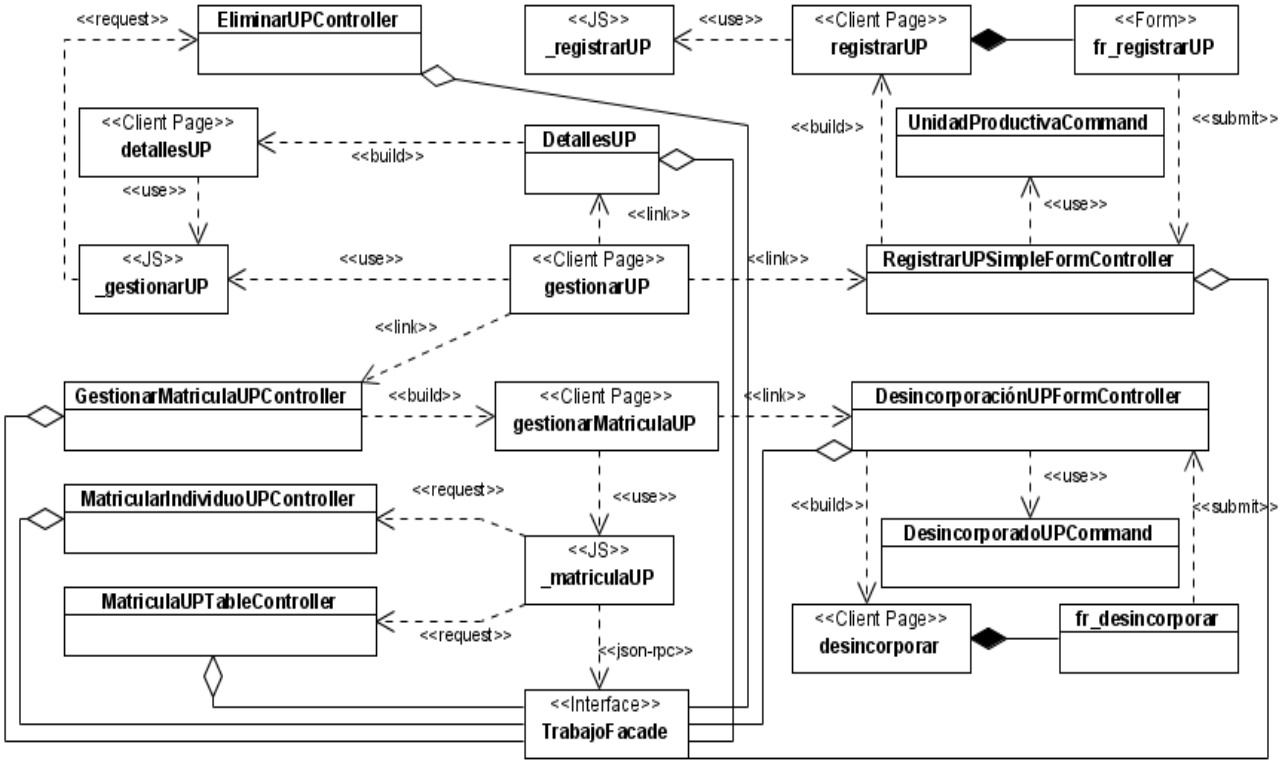


8.1.2 DCD: Funcionalidades Registrar Asistencia y Evaluación, Reporte de Desincorporados

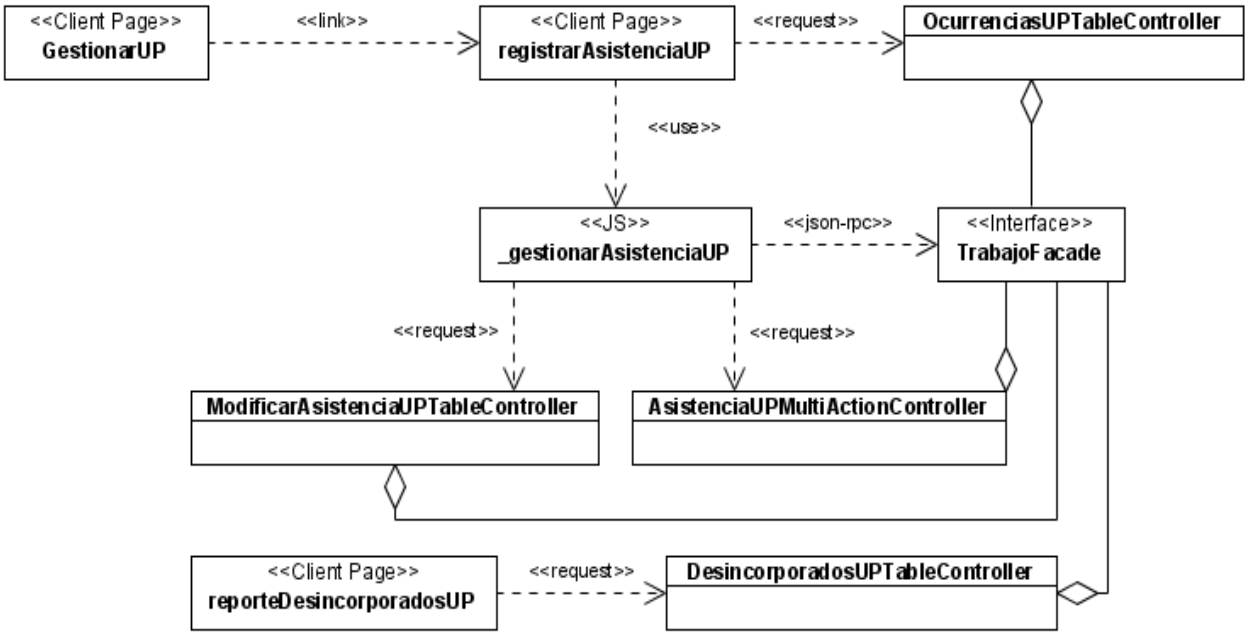


8.2 Capa de presentación del módulo Trabajo

8.2.1 DCD: Funcionalidades Gestionar Unidades Productivas y Matricula



8.2.2 DCD: Funcionalidades Registrar Asistencia y Reporte de desincorporados



GLOSARIO

D

1. **DGSP:** Dirección General de Servicios Penitenciarios

J

2. **J2EE:** Java Enterprise Edition, es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje de programación java con arquitectura de N niveles distribuida.

3. **JDBC:** Java Database Connectivity, es un API que permite la ejecución de operaciones sobre base de datos desde el lenguaje de programación Java.

P

4. **PHP4:** Hypertext Pre-Procesor versión 4, lenguaje de programación interpretado diseñado originalmente para la creación de páginas web dinámicas.

5. **PL/SQL:** Lenguaje de programación incrustado en Oracle y PostgreSQL que soporta todas las consultas y manipulación de datos que se usan en SQL, pero incluye nuevas características.

6. **PostgreSQL:** Sistema de gestión de base de datos relacional orientada a objetos de software libre.

S

7. **SIGEP:** Sistema de Gestión Penitenciaria