Universidad de las Ciencias Informáticas
Facultad 4

Diseño e implementación de los módulos Decisiones y Egresos del Sistema de Gestión Penitenciaria de la República Bolivariana de Venezuela

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

Autores: Yadira Benavides Zaila

Juan Carlos Gómez Correa

Tutor: Ing. Madelín Haro Pérez

Ciudad de La Habana, mayo de 2008

DECLARACIÓN DE AUTORÍA

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

Para que así conste firmamos la presente a los ____ días del mes de mayo del año 2008.

Yadira Benavides Zaila

Juan Carlos Gómez Correa

Ing. Madelín Haro Pérez

DATOS DE CONTACTO

Graduada en el 2002 de la carrera Ingeniería Informática en el ISPJAE.

Ha trabajado en la Empresa de Servicios Informáticos (ESI) de Cienfuegos y a partir de enero de 2003 en la UCI.

Ha sido tutora de tesis durante los 5 cursos de la UCI y cotutor o consultor. Los temas trabajados han estado relacionados con el ciclo de vida del software (análisis, base de datos, arquitectura, prototipos, requisitos,…), Gestión de Proyecto, Seguridad de sistemas informáticos y Linux.

Ha participado en eventos como Fórum de Ciencia y Técnica, UCIENCIA y Universidad 2008.

Opta actualmente por la categoría docente de asistente y es integrante en la maestría de Gestión de Proyecto.

Se ha vinculado a los proyectos productivos en el rol de Jefe de Proyecto, Analista y Jefe de Capacitación

Ha impartido cursos de postgrado a funcionarios de empresas, profesores y trabajadores de la UCI y a los Directores de los IPI.

Ha impartido cursos de capacitación en dos proyectos de producción en el extranjero.

Ha recibido curso de postgrado en el instituto TATA en la India recibiendo el certificado internacional de habilidades.

Es asesora del Departamento Docente Central de Práctica Profesional.

AGRADECIMIENTOS

Agradecemos el apoyo de nuestra tutora en medio de sus múltiples tareas.

A todos aquellos que nos aportaron conocimientos y experiencia en el mundo de Java: Iósev, Ramón e Iván muchas gracias por su ejemplo.

A Yanet Vega por estar dispuesta, siempre, a discutir sobre requisitos de software.

A nuestros padres por su apoyo incondicional.

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 Penitenciaria (SIGEP). Entre los principales procesos a informatizar por el SIGEP, en su primera versión, se encuentran Decisiones y Egresos. Para la gestión de estos procesos se definieron sendos módulos que forman parte del núcleo del sistema: Control Penal.

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

Para darle cumplimiento a las funcionalidades esperadas por ambos módulos se realizó un diseño flexible basado en patrones. De esta forma permite los cambios que se producen en medio de las transformaciones que lleva el gobierno venezolano. 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 CLAVE

Decisiones, Egresos, Recluso, SIGEP, Arquitectura, Diseño, Implementación

TABLA DE CONTENIDOS

INTRODUCCIÓN

La situación del sistema penitenciario en la República Bolivariana de Venezuela es una de las más graves en Latinoamérica. Retardo procesal, infraestructura inadecuada y obsoleta, violencia interna, corrupción y desconocimiento son algunas de las características que hacen del Sistema Penitenciario venezolano actual un problema político – social (ARIAS, 2005).

Los expedientes de los reclusos se llevan en papel y no hay formatos estándares para la información que genera cada establecimiento penitenciario hacia la Dirección General de Custodia y Rehabilitación del Recluso (DGCRR). Los movimientos e incidencias de los individuos dentro del sistema penitenciario, no son notificados adecuada y oportunamente a las instancias relacionadas como tribunales, órganos del Ministerio Público, Embajadas y familiares provocando desconocimiento sobre el cumplimiento de las leyes penitenciarias.

La DGCRR de la República Bolivariana de Venezuela no cuenta con aplicaciones informáticas para el registro y monitoreo constante de los datos asociados a la estancia del recluso en cada centro y en el sistema penitenciario en general, dígase situación jurídica actual e histórica, datos personales del individuo, ingresos y egresos realizados, decisiones judiciales sobre él ejecutadas o pendientes de ejecución y novedades (ARIAS, 2006).

Las decisiones u órdenes, que representan la comunicación del poder judicial con el sistema penitenciario, generalmente son manipuladas de forma irregular. La forma de registrar las decisiones en los expedientes penitenciarios difiere entre los establecimientos, provocando omisiones de datos significativos y valoraciones de la orden original esenciales en el cumplimiento exacto de esta. El almacenamiento de los documentos físicos de la decisión en los expedientes y en el archivo del centro, suele realizarse sin orden cronológico causando fallas en el proceso manual de revisión de expedientes próximos a la ejecución de decisiones, situación crítica para aquellas decisiones que impliquen salida del centro o del sistema penitenciario (egreso).

El egreso de los individuos por cambio de régimen o libertad definitiva del sistema penitenciario no se garantiza inmediatamente cuando se ha recibido una orden informando al respecto. La comprobación de los procesos legales pendientes o ejecuciones por cumplir, necesaria en estos casos, es un tedioso proceso de análisis manual del expediente del individuo cuya estructura en papel puede ser disímil. Además, la forma de proceder cuando se encuentran procesos legales pendientes varía de un establecimiento a otro dejando muchas veces la decisión final a manos del director del establecimiento, en infracción latente de la ley. Las notificaciones de egreso generadas en todos los establecimientos no tienen el mismo formato.

En los casos de egreso por defunción de un individuo, el procedimiento que se sigue no es homogéneo en los centros penitenciarios. Algunos declaran la defunción sin que las instituciones forenses autorizadas la decreten, otros esperan la documentación científica según lo establecido.

A partir de la situación de los procesos en el Sistema Penitenciario Venezolano y la experiencia cubana en esa área, se decide, en el marco de la cooperación bilateral, la realización de un software que apoye la gestión del sistema penitenciario venezolano actual y estandarice algunos de los procedimientos haciendo cumplir la legislación venezolana, este software es el Sistema de Gestión Penitenciaria (SIGEP). Este sistema cuenta con el núcleo Control Penal del cual derivan una serie de procesos que tributan a la gestión de las estancias de los reclusos en el sistema penitenciario. Dentro de esos procesos se encuentran Decisiones y Egresos. De estos dos procesos se han definido y validado con el cliente, la DGCRR, los requisitos a informatizar restando, para estos módulos, la secuencia siguiente en el proceso de desarrollo del sistema automatizado: el diseño y la implementación.

Dentro del desarrollo como sistema del SIGEP nos plateamos como problema: ¿Cuál es el diseño e implementación de los módulos Decisiones y Egresos, a partir de los requerimientos de software establecidos, teniendo en cuenta la arquitectura y tecnologías definidas para el proyecto SIGEP?

El objetivo central de nuestro trabajo es diseñar e implementar los módulos Decisiones y Egresos 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 Egresos y Decisiones, la validación de dichos modelos y los módulos como componentes ejecutables integrados al SIGEP.

En consecuencia, el objeto de estudio del presente trabajo es el proceso de desarrollo de los módulos Decisiones y Egresos del Sistema Penitenciario Venezolano y el campo de acción los modelos de Diseño e Implementación de los módulos Decisiones y Egresos del SIGEP.

La realización de este trabajo está basado en el desempeño de los autores en los roles de diseñador e implementador del proyecto SIGEP. Así en función de los objetivos y de las actividades a realizar de acuerdo a los roles desempeñados, nos trazamos las siguientes tareas de investigación:

- Estudio de las tecnologías y herramientas a utilizar en el diseño e implementación de los módulos Decisiones y Egresos definidas en la arquitectura del SIGEP.
- Análisis de los requisitos de software y del modelo de negocio correspondientes a los módulos de Egresos y Decisiones.

- Diseño de la solución de software para los requisitos relacionados con el proceso de registro de decisiones.
- Diseño de la solución de software para los requisitos relacionados con el proceso de dar egreso a un individuo privado de libertad por cualquiera de sus variantes.
- Implementación del diseño realizado relacionado con el proceso de registro de decisiones. Realizar pruebas unitarias y de integración.
- Implementación del diseño realizado relacionado con el proceso de dar egreso a un individuo privado de libertad. Realizar pruebas unitarias y de integración.
- Redacción de la documentación correspondiente al diseño y la implementación en el documento del trabajo de tesis.
- Integración de los módulos de Egresos y Decisiones 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 Egresos y Decisiones. Detalla la solución a las dependencias de otros módulos con Decisiones y Egresos, a través de la utilización de eventos. 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 proceso de diseño e implementación de los módulos Decisiones y Egresos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se abordan algunos conceptos del Sistema Penitenciario venezolano que se referencian como parte de la situación problémica 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 Decisiones y Egresos del Sistema Penitenciario venezolano, en el marco del desarrollo del Sistema de Gestión Penitenciaria (SIGEP).

1.2 Sistemas Penitenciarios

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

Manuel Osorio, creador del diccionario de Ciencias Jurídicas, Políticas y Sociales, asocia el Sistema Penitenciario con régimen penitenciario, definiendo éste 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 específico de la República Bolivariana de Venezuela, tal sistema está constituido por la legislación vigente, los métodos que se emplearán para lograr su funcionamiento, las diferentes dependencias encargadas de su aplicación, los equipos de trabajo y la infraestructura carcelaria.

1.3 Régimen penitenciario

Las leyes venezolanas distinguen dos regímenes posibles para el cumplimiento de condenas: Intramuros y Extramuros. Para cada individuo el sistema judicial decide el régimen adecuado tomando

[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.

en cuenta historial judicial, salud y precedentes delictivos. Los cambios de regímenes se expresan a través del otorgamiento o revocatoria de Fórmulas Alternativas de Cumplimiento de Pena (FACP).

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

Régimen Extramuros o abierto se refiere a los centros que albergan a los individuos que se encuentran bajo cumplimiento de medidas restrictivas de libertad. Los centros de Extramuros son las Unidades Técnicas de Apoyo al Sistema Penitenciario (UTASP) y los Centros de Tratamiento Comunitario (CTC), donde el individuo no se encuentra a tiempo completo porque generalmente está insertado al trabajo y a la sociedad directamente.

En Venezuela existe preferencia por el Régimen Extramuros debido al hacinamiento y falta de estructuras para mantener a los reclusos en los locales cerrados y a que el individuo se encuentra insertado a la sociedad lo que favorece a su reinserción como ciudadano pleno.

1.4 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. 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 del 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 basó en la información que cada uno de ellos proporciona en sus sitios digitales. A continuación se abordan características de algunos de ellos.

1.4.1 Sistema penitenciario del gobierno de Panamá

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

Este sistema fue creado a principios del 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 almacenado en una base de datos los registros de los internos que están detenidos en los centros penales a nivel nacional. Es un sistema centralizado, con una sede que se enlaza con otras unidades operativas distribuidas en una parte de los centros penitenciarios, con el resto se mantiene un enlace mediante el sistema de actualización por disco de tres y media.

El proceso se inicia con la captura de la información 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 tienen acceso al sistema. Esto garantiza que se refleje de forma inmediata los cambios en el expediente del interno tales como traspasos de autoridad, traslado de un centro a otro, libertades, diligencias médicas, jurídicas y la realización del cómputo automático de la sentencia.

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

1.4.2 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 desactualizada 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.

1.4.3 Establecimiento Penitenciario Virtual en 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: nóminas 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 accesos, etcétera.

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

1.4.4 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 la legislación vigente en los países para los cuales fueron definidos lo que los convierte en sistemas no portables a otro sistema penitenciario.

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

1.5 Sistema Penitenciario Venezolano

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 documentos 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 muchas de estas problemáticas. El resultado esperado de este convenio es un sistema informático centralizado que estandarice e implemente los procesos fundamentales en el sistema penitenciario venezolano y se convierta en herramienta de trabajo para funcionarios en los centros penitenciarios y para la toma de decisiones a nivel institucional.

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.

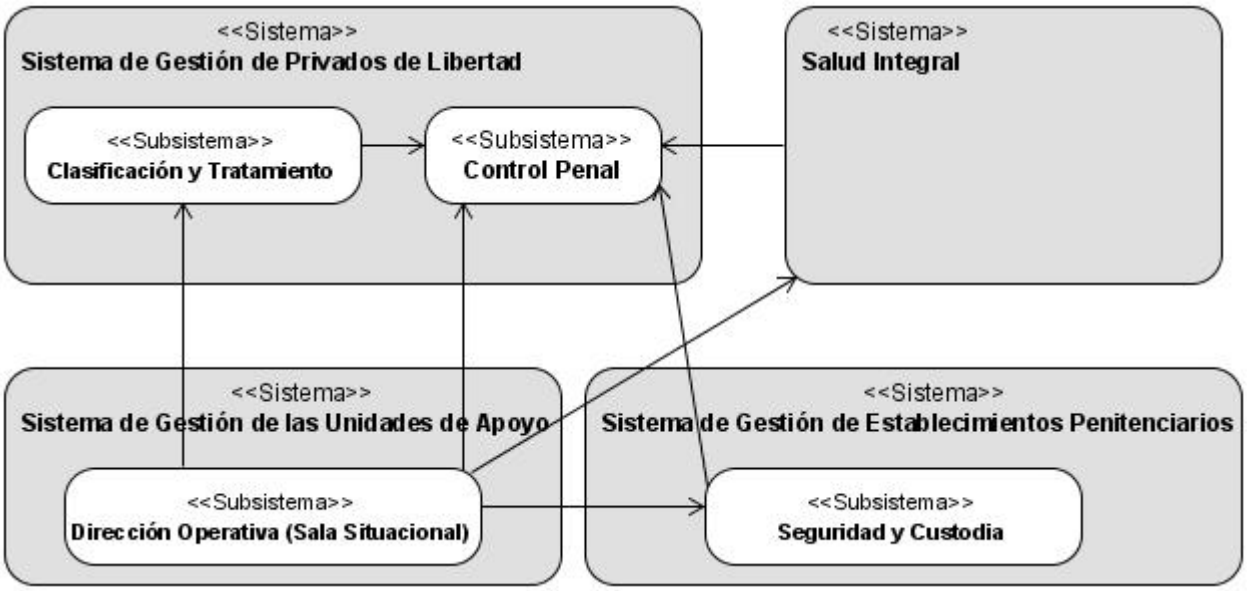


Figura 1.1 Sistemas y subsistemas que integran el SIGEP y sus relaciones de dependencia.

1.5.1 Control Penal

El núcleo de los procesos a implementar en el SIGEP es el área de Control Penal definido como “el proceso del Sistema Penitenciario encargado de garantizar el cumplimiento de la legalidad, la organización, control y tramitación de la documentación legal del procesado o penado durante el ingreso, reingreso, permanencia y egreso de los lugares de internamiento.”(PÉREZ, 2007)

Control Penal contiene los procesos Ingreso, Datos Personales, Decisiones y Egresos que se vinculan directamente con el individuo, su situación jurídica y su tránsito por el sistema penitenciario. A través de sus procesos, Control Penal interactúa con las áreas de Seguridad y Custodia, Tratamiento y Salud Integral desencadenando la mayoría de los procesos del sistema penitenciario.

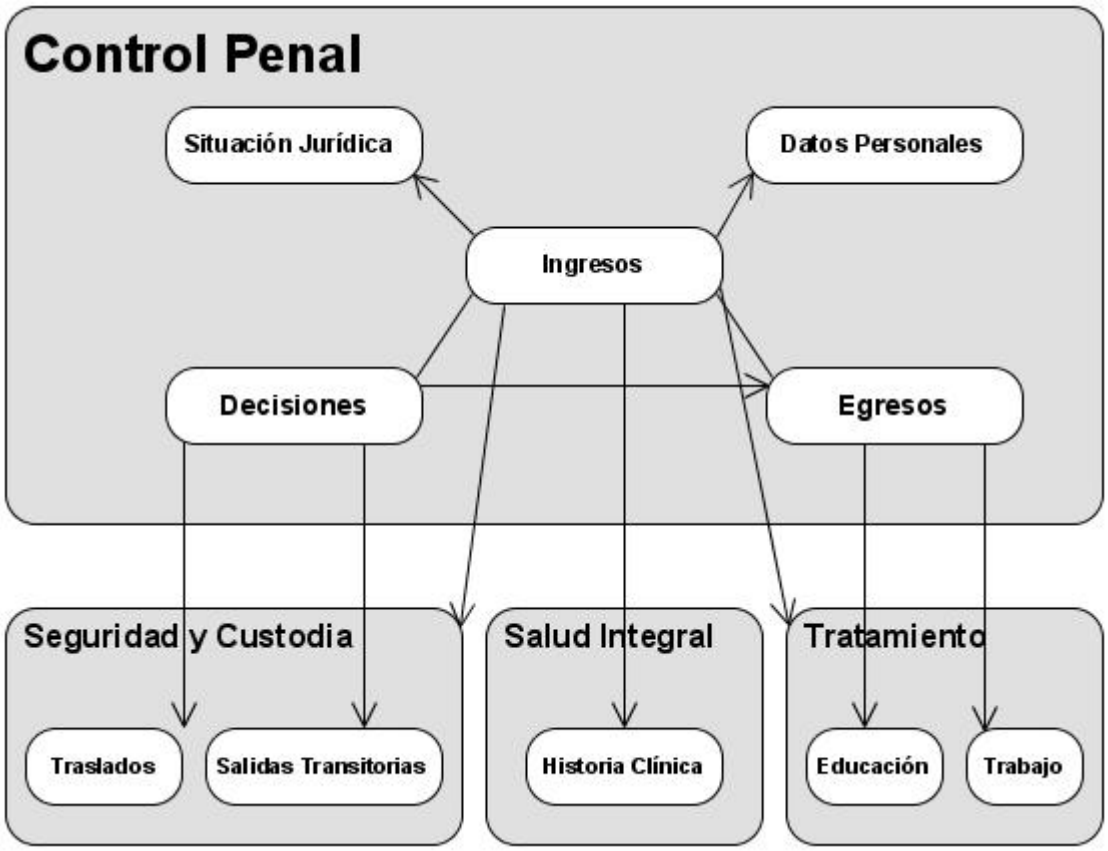


Figura 1.2 Mapa de relaciones entre procesos del sistema penitenciario venezolano

La Figura 1.2 muestra las relaciones internas de algunos procesos del sistema penitenciario venezolano. Llámese relaciones internas a todos aquellos 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, la ausencia de estas indica un flujo de información en ambas direcciones.

Como se aprecia en la Figura 1.2, Control Penal contiene Decisiones y Egresos, procesos claves como desencadenadores de otros procesos en el sistema penitenciario. Estos se detallan a continuación por constituir parte del objeto de estudio del trabajo de diploma.

1.5.2 Proceso Decisiones

En el marco del sistema judicial y penitenciario venezolano las decisiones u órdenes son las acciones dictaminadas por las instancias competentes del poder judicial (Tribunales, Órganos del Ministerio Público, Órganos de Seguridad Ciudadana y Coordinaciones Regionales), sobre los individuos que se encuentran bajo la custodia de la Dirección General de Servicios Penitenciarios (DGSP).

Las decisiones son el recurso que expresa la comunicación entre el poder judicial y el sistema penitenciario en cumplimiento de la ley venezolana. Se comunican a través de documentos de carácter legal que son enviados al centro donde se encuentre el individuo y se anexan a su expediente judicial. Estos documentos se reproducen para ser almacenados en los archivos del establecimiento porque implican el desarrollo de procesos propios de los centros penitenciarios, de acuerdo al régimen penitenciario correspondiente, como traslados, salidas transitorias, egresos o ingresos (VEGA, SW-SIGEP-06).

Las decisiones tienen ejemplificación y se clasifican cuando se aplican a otro proceso. Más adelante, en el epígrafe de Proceso Egreso, se demuestra la relación de este módulo con las salidas de los reclusos de los centros penitenciarios.

1.5.3 Proceso Egresos

El egreso de los individuos sometidos a una medida privativa o restrictiva de libertad en el sistema penitenciario venezolano se traduce en la salida del recluso del Establecimiento Penitenciario, CTC o UTASP donde se encuentre vinculado.

Los egresos posibles en el sistema penitenciario venezolano, según la legislación vigente, se dividen en egresos por evasión o fuga, defunción y decisión judicial.

El egreso por evasión ocurre cuando un individuo sale sin autorización del centro penitenciario al que se encuentra vinculado porque tuvo las condiciones propicias para ello, por ejemplo deficiente vigilancia. La evasión no es considerada un delito por lo que no representa un agravante para el tiempo de condena del recluso en caso de ser capturado.

El egreso por fuga ocurre cuando un individuo hace uso de la fuerza o la extorsión para salir del Establecimiento Penitenciario o no se persona al centro de Extramuros al cual se encuentra asociado.

En las leyes venezolanas la fuga constituye un delito que es agravante en caso de la captura del fugado.

En el egreso por evasión o fuga es de obligatorio cumplimiento la notificación de los hechos a las autoridades civiles y militares, así como la emisión de la Orden de Captura del individuo y la Requisitoria que contiene todos los datos físicos y conductuales del evadido o fugado.

El egreso por defunción se refiere, como el nombre lo indica, a los casos de fallecimiento; estos pueden producirse dentro de los centros penitenciarios o fuera de ellos, por ejemplo en hospitales y clínicas. Para registrar un egreso por defunción es necesario que se presenten los documentos que avalen tal condición, emitidos por las instituciones pertinentes, dígase el Cuerpo de Investigaciones Científicas, Penales y Criminalísticas (CICPC), Hospitales, Registros Civiles y Alcaldías. En caso de que no exista certeza científica de la muerte del individuo, se declara presuntamente fallecido hasta que la documentación sea generada. En cualquier situación los familiares, tribunales que llevaban procesos al individuo, embajadas (en caso de extranjeros) y las oficinas civiles y del gobierno deben ser notificados oportunamente.

El egreso por decisión judicial, indica la salida de un individuo del penal que es ordenada por alguna de las instancias del poder judicial (tribunales de Control, Juicio, Ejecución, Apelación o el Tribunal Supremo de Justicia), algún Órgano de Seguridad Ciudadana, el CICPC, la Coordinación Regional o la Dirección General de Servicios Penitenciarios (DGSP).

Las decisiones pueden ser otorgadas como resultado de convenios internacionales en los casos de la repatriación y extradición, por leyes nacionales de excepción como la amnistía y el indulto presidencial, por cambio de régimen penitenciario en el otorgamiento y revocatoria de destacamento de trabajo o de una FACP de régimen abierto, por criterios humanitarios para enfermos con padecimientos múltiples en la libertad condicional por ayuda humanitaria, por suspensión condicional de la ejecución de la pena; traslado a otros centros, sobreseimiento o elementos no comprobatorios de culpabilidad o simplemente por cumplimiento de la pena impuesta. El expediente penitenciario de estos reclusos se cierra en la mayoría de los casos para las penas abiertas aunque pueden ser reabiertas las causas e implicar un re-ingreso, en dependencia del motivo por el cual se haya otorgado el egreso.

El egreso por decisión judicial se ejecuta en el centro si el individuo no tiene otros procesos pendientes o ejecuciones de procesos por cumplir. Se ejecute o no, se tiene que notificar a las instituciones judiciales involucradas con el individuo.

Por tanto dentro del proceso de Egresos, el egreso por decisión judicial no es más que la expresión del cumplimiento de la ley a través de una orden o decisión. Así el proceso de Decisiones desencadena el egreso por decisión judicial del proceso de Egresos.

1.6 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.

1.6.1 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 4 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 software. Además RUP identifica una serie de roles para los

trabajadores que realizan las actividades de cada flujo, dentro de los principales roles se encuentran: ingeniero de requisitos, diseñador, arquitecto, implementador, jefe de proyecto y probador.

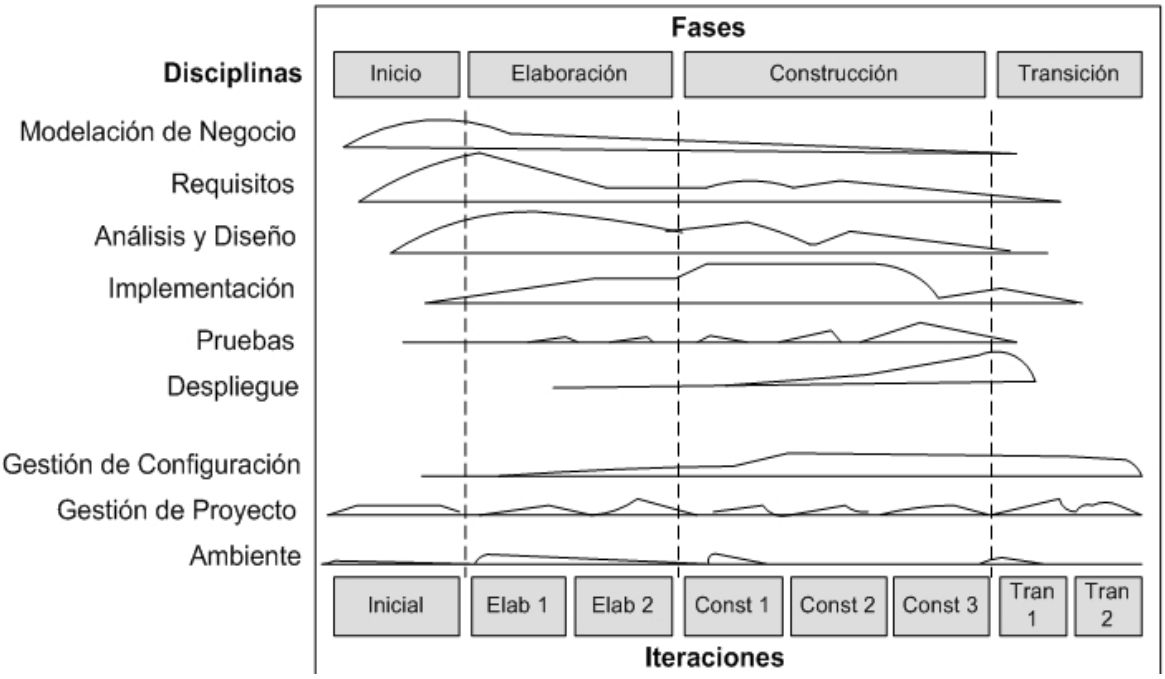


Figura 1.3 Fases y flujos de trabajo en RUP (RATIONAL SOFTWARE CORPORATION, 2003).

Los flujos de trabajo representados en la Figura 1.3 proponen la realización de un gran número de actividades y la elaboración de un amplio conjunto de artefactos, que usualmente los proyectos de desarrollo de software se comprometen a desarrollar pero que no realizan en su totalidad debido a la carencia de tiempo o a que descubren que no eran necesarios. RUP indica que al inicio del proyecto se realice una adecuación de cada flujo de trabajo de manera que se produzcan solo los artefactos y se realicen las actividades que tienen un propósito dentro del proyecto. La adecuación de RUP para el SIGEP se especifica en (ARIAS, 2006).

A continuación se describen las disciplinas de mayor interés de acuerdo a los roles desempeñados por los autores 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 obtenidas en la disciplina Requisitos. Los trabajadores más relevantes involucrados en este flujo son el diseñador y el arquitecto.

La adecuación del SIGEP para la disciplina contempla como artefactos a generar el Modelo de Diseño y el Modelo de Datos y como trabajador principal al diseñador.

El Modelo de Diseño es detallado en cuanto a la arquitectura porque debe ser implementado sin ambigüedad. Este modelo es esencial para las actividades de implementación y pruebas. Contiene subsistemas, paquetes, clases de diseño y diagramas de clases e interacción.

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 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.

Como resultado de las actividades realizadas por el implementador se obtiene el software listo para ser probado por terceros, con todas las funcionalidades implementadas y si fuese necesario diagramas de componentes. Todo ello conforma el Modelo de Implementación.

1.6.2 Patrones de diseño de software

Los patrones de diseño 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 a los autores 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 del 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 use, debe ser implementado nuevamente.

De modo general, la descripción de un patrón de diseño incluye las siguientes partes:

Nombre: frase que encierra la característica fundamental del patrón.

Problema: descripción del problema típico que el patrón resuelve.

Solución: especificación de cómo resolver el problema. Incluye la descripción de los elementos que componen el patrón y cómo se relacionan entre sí.

Implicaciones: define cuándo y cómo el patrón debe ser usado y los beneficios y problemas que implican su uso.

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 real de la comunicación con la fuente de datos y que permite una migración más fácil de 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 menor acoplamiento y facilitando el cambio de componentes sin afectar a sus clientes.
- **Strategy (Estrategia):** Se utiliza cuando se necesitan diferentes variantes de un algoritmo o cuando una clase define muchos comportamientos los cuales se manifiestan como definiciones condicionales múltiples de sus operaciones. Este patrón propone encapsular los algoritmos en diferentes clases, eliminando las costosas definiciones de comportamiento multicondicionales. Brinda gran flexibilidad para futuras extensiones y tiene como desventaja, que puede producir una gran explosión en el número de objetos del sistema.
- **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.
- **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.

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

1.6.3 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, cómo se deben nombrar las variables, clases y ficheros y cómo se deben escribir instrucciones específicas del lenguaje de programación.

Las convenciones 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 esté bien o mal escrito sin embargo uno mal escrito tiene una probabilidad mayor de no funcionar correctamente además de que es casi imposible de depurar o mantener.

Las convenciones 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 convenciones de *Javadoc*[2] adaptadas para el SIGEP.

1.6.4 Programación orientada a objetos.

La Programación Orientada a Objetos (POO) es un paradigma de programación que expresa un programa como un conjunto de objetos que colaboran entre ellos para realizar tareas. Un objeto es una entidad provista de un conjunto de propiedades o atributos y de comportamiento o funcionalidad que representa a los objetos, procesos o conceptos del mundo real. Junto a su definición se relacionan un conjunto de claves dentro de su significado como lo son:

[2] 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.

- Clase es la definición de las propiedades y comportamiento de un conjunto de objetos concretos.
- Instancia es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

La POO no rompe con el paradigma secuencial, estructural y otros anteriores, sino que contiene elementos de las mejores prácticas de estos y agrega nuevos beneficios. Ejemplo de ello, utilizado con frecuencia en el SIGEP es la reutilización y la extensión del código. Además, permite agilizar el desarrollo de software y facilitar el trabajo en equipo y el mantenimiento del software.

En sus pocas décadas de desarrollo, la POO ha demostrado ser imprescindible para el manejo de la complejidad creciente de los sistemas informáticos actuales. En el SIGEP se utiliza como lenguaje Java, exponente puro de este paradigma.

1.6.5 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 máquina 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 bases 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.

[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.

• **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.

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.

[4] HTTP, del inglés *HyperText 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.

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.

1.6.6 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*[5] 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.

[5] 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 dominio:** Proporciona funcionalidades para un dominio específico de aplicación.
- **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.

1.6.6.1 Framework Spring

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

Spring es un framework basado en la *Inversión de Control*[6] y en la *Programación Orientada a Aspectos*[7]. 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.

---

[6] 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.

[7] 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.

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 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.

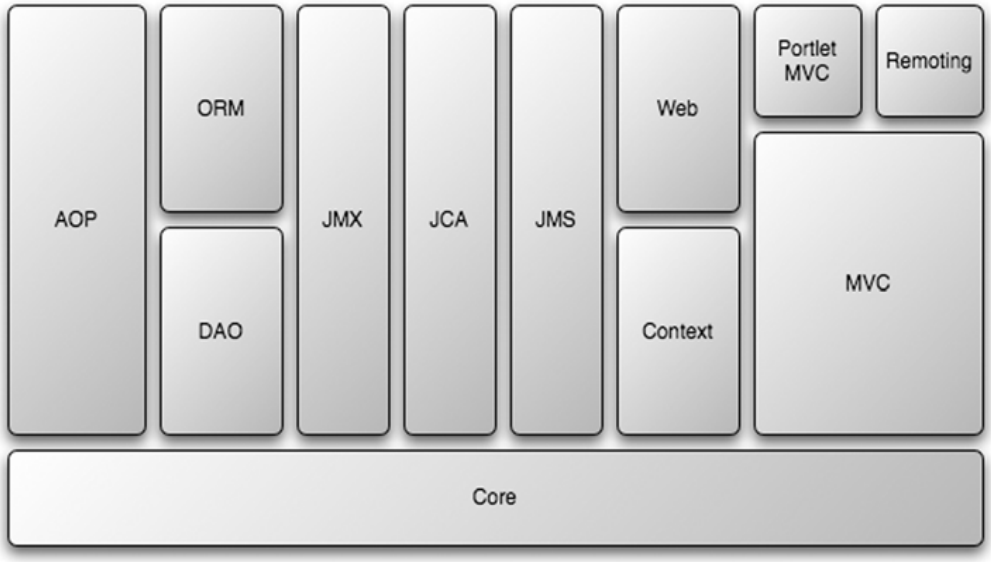


Figura 1.4 Módulos del Framework Spring (WALLS, 2007)

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*[8] 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.

[8] 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.

- **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.

1.6.6.2 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.

1.6.7 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.

1.6.8 Entorno integrado de desarrollo (*IDE*[9])

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 *pluig-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:

[9] 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*[10] 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/)

1.6.9 Bibliotecas

Jasper Reports

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

Librería gratis y de código abierto que permite la generación de reportes en varios formatos como PDF, RTF, HTML, XML y XLS, a partir de una plantilla XML. Jasper Reports permite representar información diversa de distintas maneras, pudiéndose incluir en los reportes texto, tablas, imágenes y gráficos.

Esta librería se utiliza en el SIGEP para mostrar información estadística que apoye la toma de decisiones de la DGCRR y para la generación de documentos oficiales que emiten los centros penitenciarios.

[10] 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.

Dojo Toolkit

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

Es una biblioteca *JavaScript*[11]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.

1.6.10 Herramienta de modelado

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

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

De los diagramas mencionados se utilizarán en este trabajo los diagramas de clases, interacción del Modelo de Diseño y de entidad relación para el Diseño de la Base de Datos.

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.

1.7 Conclusiones

Para el desarrollo de los procesos Decisiones y Egresos se tuvo en cuenta sistemas similares en el área geográfica en el que se va a implantar finalmente el SIGEP. A partir del 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.

Las características de los procesos Decisiones y Egresos, como partes del núcleo del SIGEP, reflejaron la importancia de la informatización de estos procesos para el Sistema 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 sofware una adecuación para el proyecto de la metodología RUP. Los flujos de trabajo sobre 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 del trabajo de implementación con estos frameworks y otros abordados en el capítulo.

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

CAPÍTULO 2: ARQUITECTURA DEL SISTEMA

2.1 Introducción

En el siguiente capítulo se describe la arquitectura que utiliza el SIGEP, en consecuencia la arquitectura sobre la cual se desarrollan los módulos Decisiones y Egresos. 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.

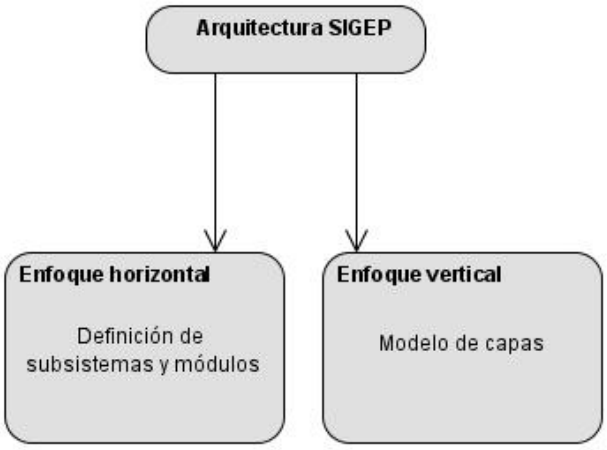


Figura 2.1 Enfoques de la arquitectura del SIGEP

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 el subsistema común, subsistemas de negocio y de soporte como se muestra en la Figura 2.2.

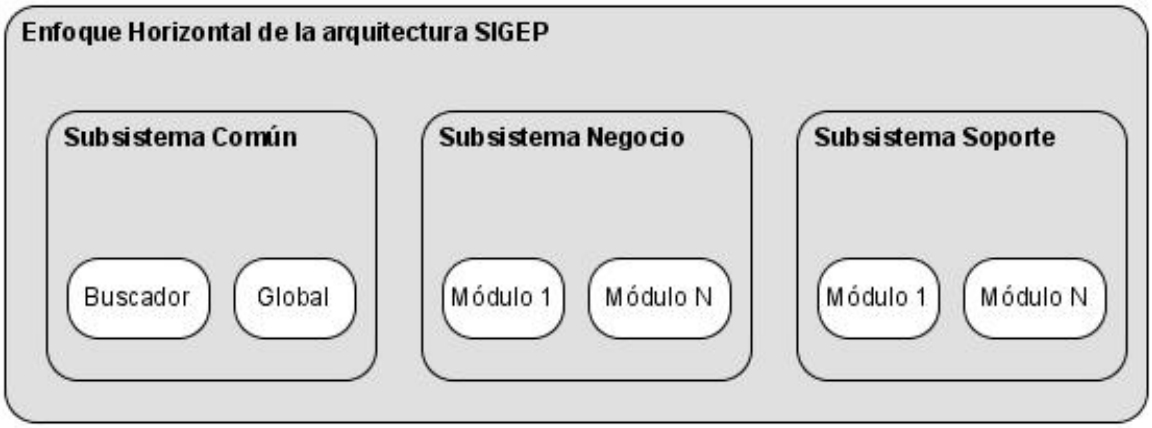


Figura 2.2 Enfoque Horizontal de la arquitectura del SIGEP

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 Control Penal, Seguridad y Custodia, Tratamiento y Salud Integral.

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

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 Decisiones, Egresos, Ingresos, Datos Personales y Situación Jurídica que se encuentran dentro del subsistema Control Penal.

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.

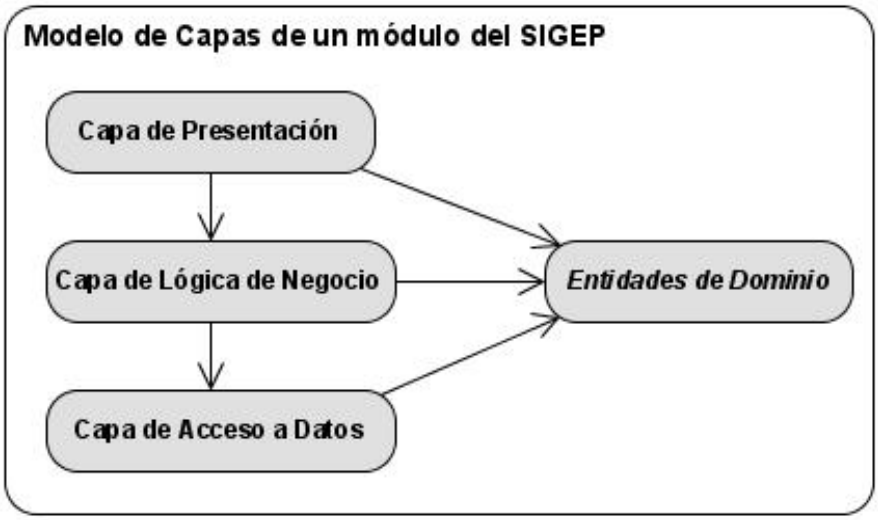


Figura 2.3 Modelo de capas de un módulo en el SIGEP

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 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.

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 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.

2.3 Actividades 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*[12] y el *Proceso Elemental de Negocio*[13] correspondientes. Esta documentación es la

[12] 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.

especificación de lo que el módulo debe hacer, establecido en acuerdo mutuo y en forma de contrato entre el cliente y el equipo de desarrollo durante la captura de requisitos. Tomando como partida esta documentación las actividades a realizar, por el(los) diseñador(es) e implementador(es) son:

1. Análisis de las funcionalidades
2. Diseño del 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.

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

2. Diseño del dominio.

El diseño del dominio constituye una entrada principal a las restantes actividades de diseño. En este punto se identifican las entidades que serán gestionadas por la capa de negocio, persistidas o recuperadas por la capa de acceso a datos y mostradas por la capa de presentación. 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 persistentes del módulo Egresos.

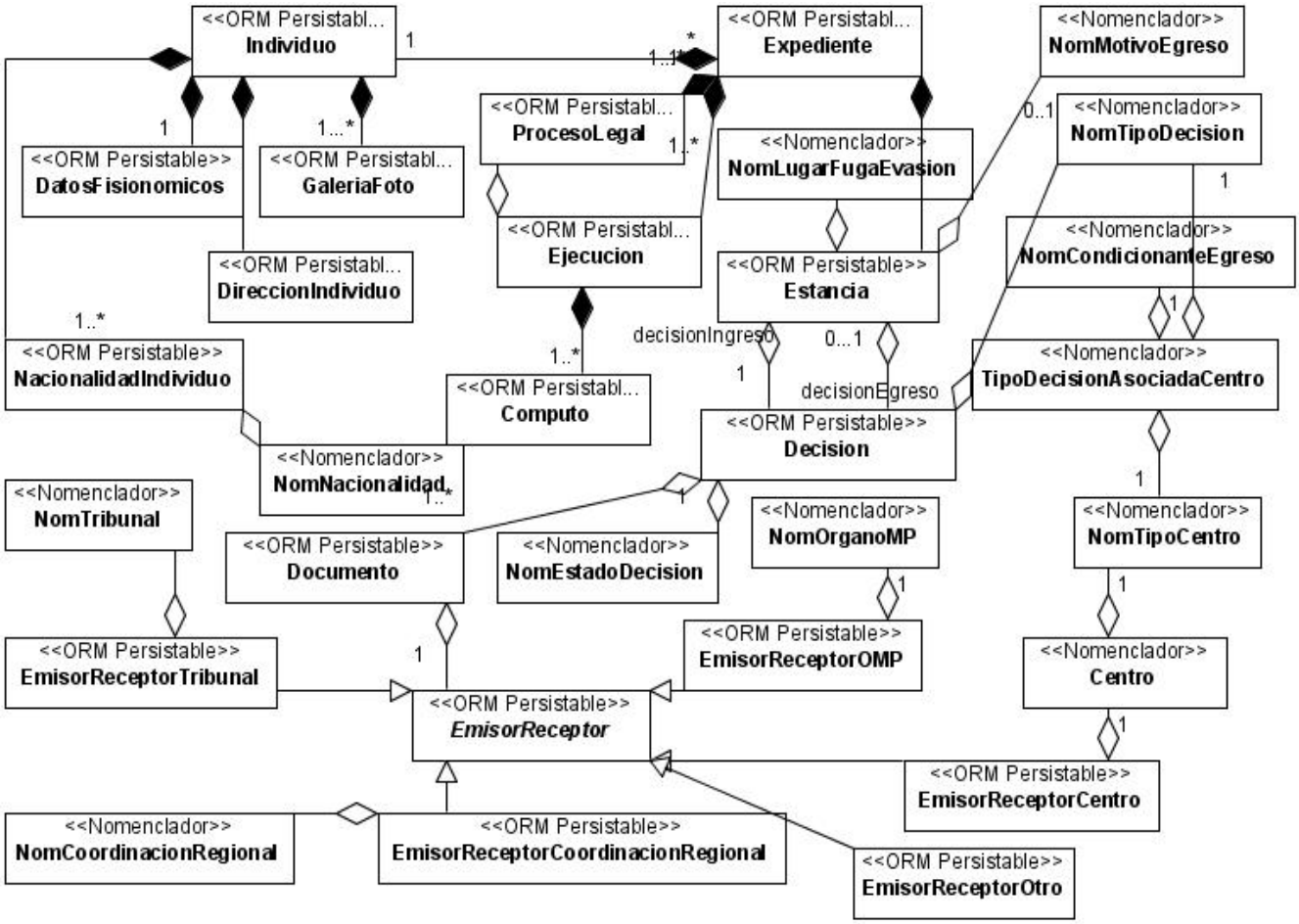


Figura 2.4 Diagrama de clases persistentes del módulo Egresos

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 persistentes aporta al modelo de datos una buena aproximación de la estructura estática de la base de datos. Esta estructura tiene que garantizar que el almacenamiento y recuperación de la

información ocurra de manera adecuada y que se cumplan las restricciones identificadas, a través de la integridad referencial. En la Figura 2.5 se muestra el diagrama lógico de la base de datos correspondiente al módulo Egresos, generado a partir de la definición de las entidades persistentes.

El modelo de datos es refinado en la medida en que se avanza en las actividades de diseño porque sea necesario persistir nuevos elementos o se decida la utilización de procedimientos almacenados, vistas, restricciones y/o funciones definidas en el gestor de base de datos. 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.

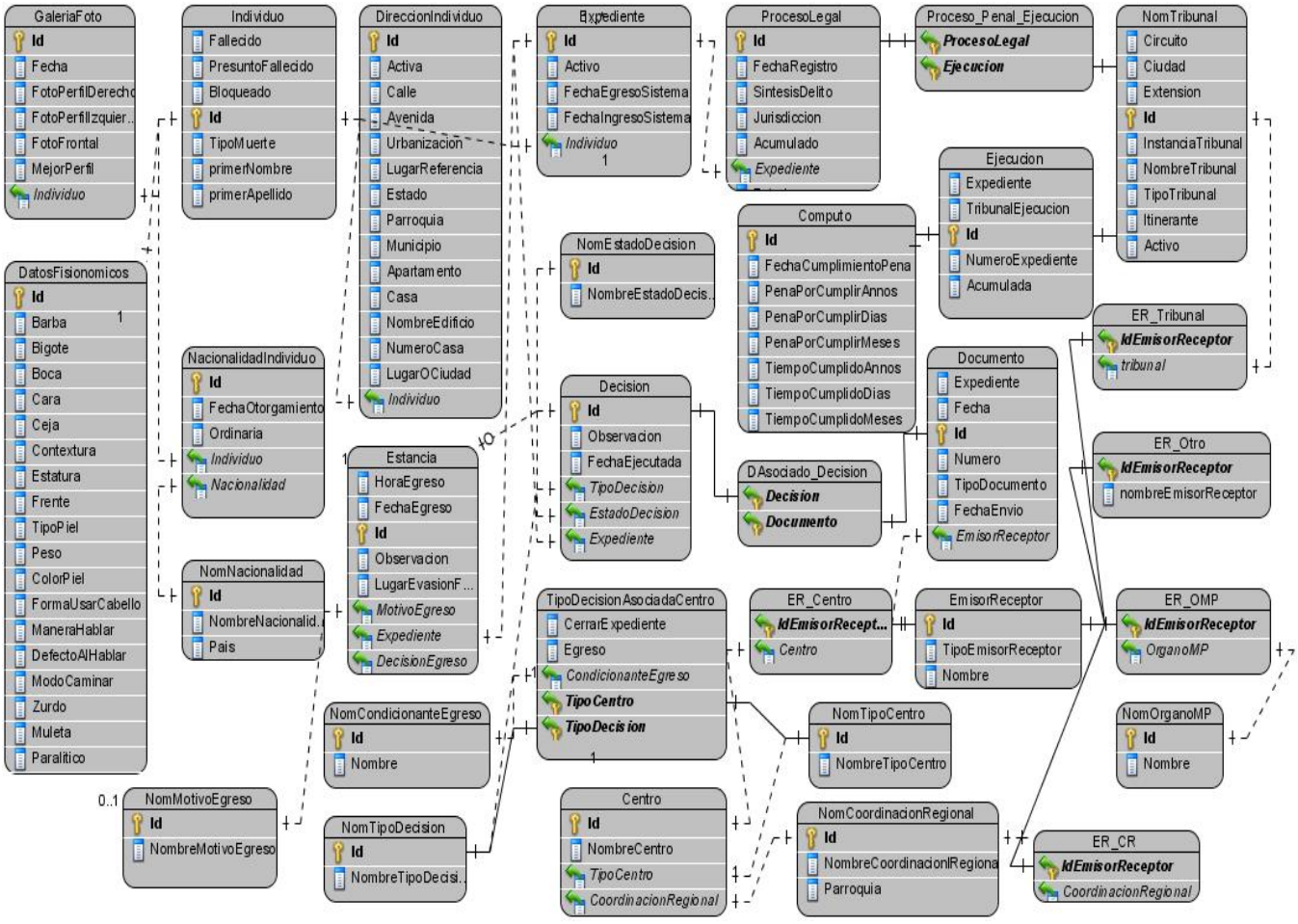


Figura 2.5 Diagrama entidad relación del módulo Egresos

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, las excepciones a lanzar y la manera en que van a ser tratadas y mostradas al usuario. En la Figura 2.6 se muestra el diagrama de clases correspondiente al diseño de la capa de negocio del módulo Egresos a manera de ejemplo de lo explicado. Aquí se muestran las interfaces e implementaciones de los managers definidos, la fachada del módulo Egresos, las excepciones y la definición del evento que tiene que ser lanzado cuando un individuo es egresado del sistema penitenciario para el escenario de Egreso por decisión judicial.

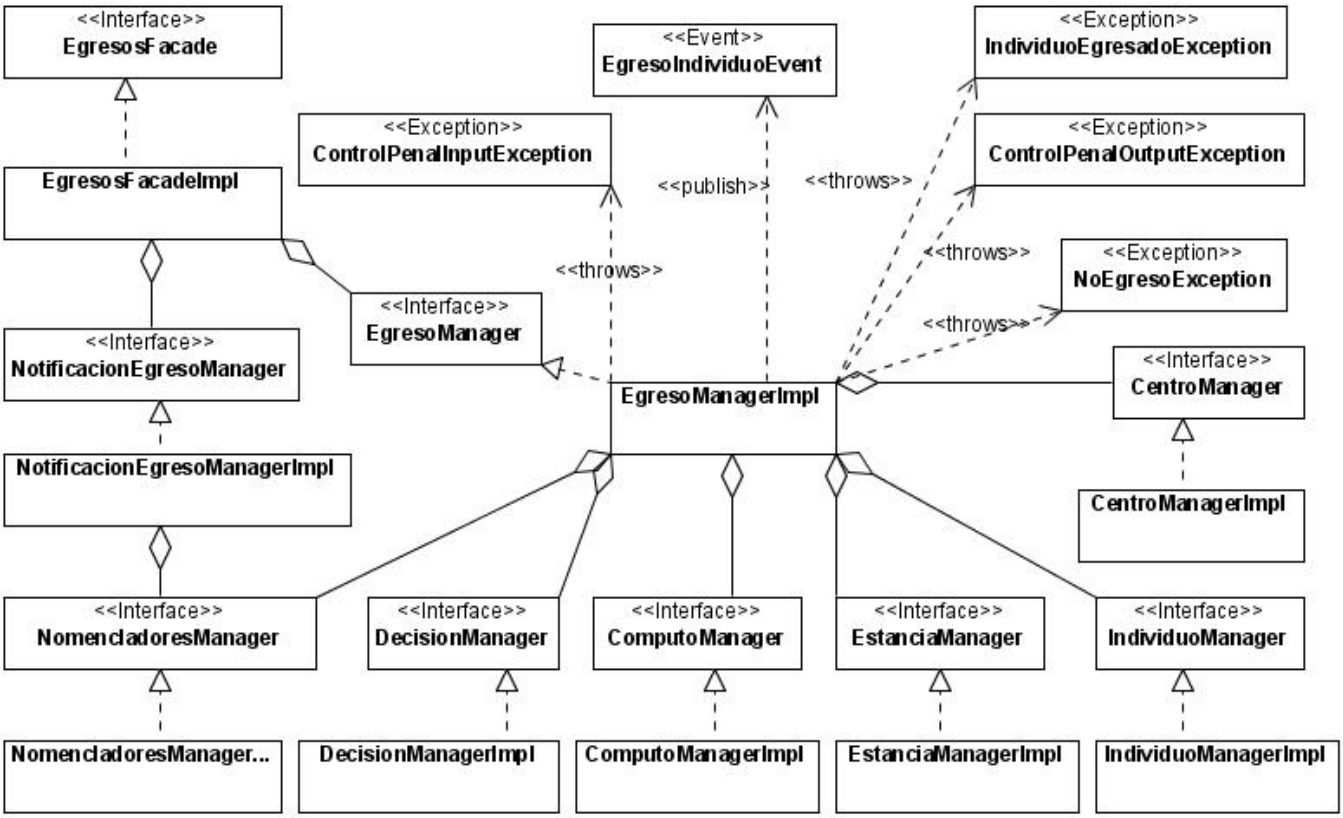


Figura 2.6 Diseño de la capa de negocio del escenario Egreso por decisión judicial

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`, `findByCriteria` y `findByExample`.

En la Figura 2.7 se muestra el resultado del diseño de la capa de acceso a datos del módulo Egresos y la comunicación directa entre un objeto de la capa de acceso a datos del módulo (`IndividuoDAOImpl`) con funciones definidas en un paquete del gestor de base de datos. En este caso en particular la función *cumplimientos_decisiones* recupera todos los individuos que pueden ser egresados ya sea porque tienen decisión judicial pendiente (función: *decision_judicial_pendiente*) o porque tienen un cumplimiento de pena para esa fecha (función: *cumplimientos_de_pena*).

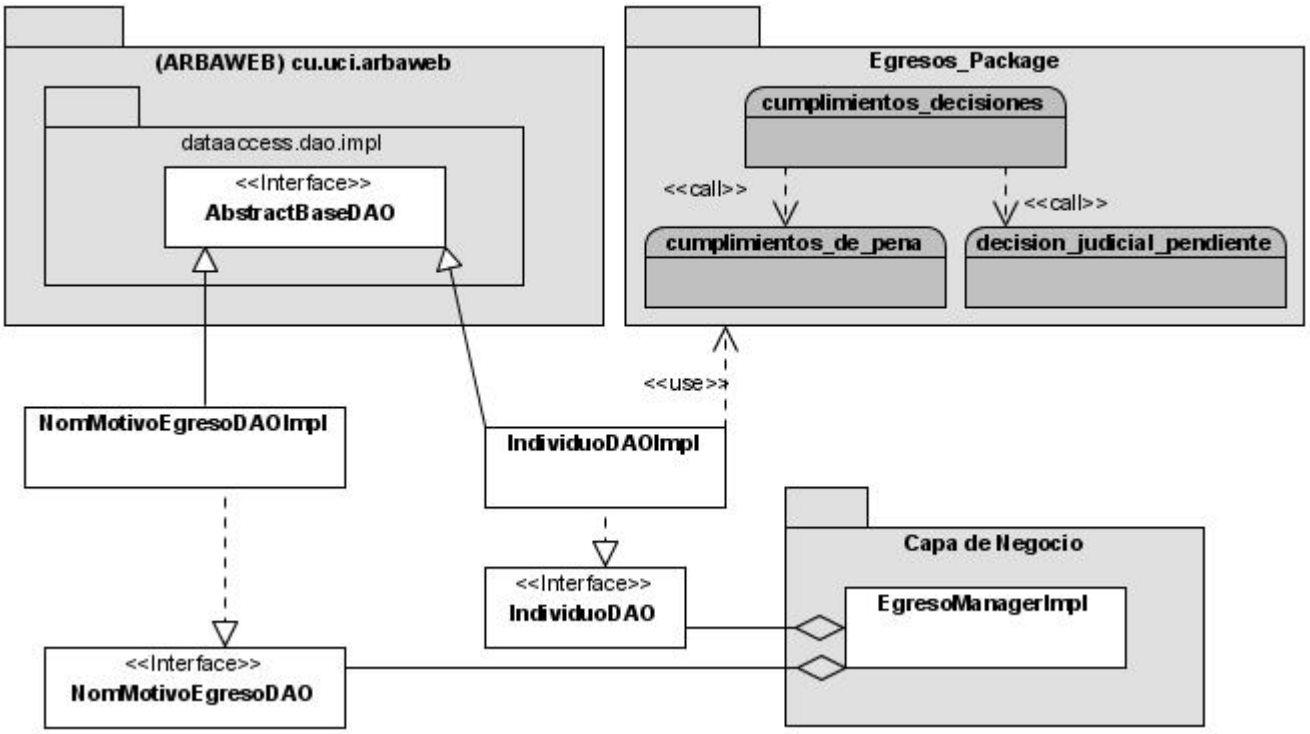


Figura 2.7 Diagrama de la capa de acceso a datos del módulo Egresos

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 `registrarEgresosPorDecisionJudicialIndividuo` perteneciente al manager `EgresoManagerImpl` 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*[14].

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 `EgresosFacadeImpl`, que refleja la ausencia de lógica interna en la implementación del método `registrarEgresoPorDecisionJudicialIndividuo`.

```
public class EgresosFacadeImpl implements EgresosFacade {

      private EgresoManager egresoManager;

      public void setEgresoManager(EgresoManager egresoManager) {
            this.egresoManager = egresoManager;

      }

      ...
      public void registrarEgresoPorDecisionJudicialIndividuo(
         DatosEgresoDecisionJudicial datosEgresoDecisionJudicial) throws Exception {
            egresoManager.registrarEgresoPorDecisionJudicialIndividuo (
                                                datosEgresoDecisionJudicial);

      }}
```

[14] 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*]-dataaccess-context.xml**. En la configuración de estos objetos se inyecta el *sessionFactory* de Hibernate como se muestra en el siguiente ejemplo:

```
<bean id="decisionDAO"
            class="vnz.sigep.controlepnal.decisiones.dao.impl.DecisionDAOImpl">
      <property name="sessionFactory">
            <ref bean="hibernateSessionFactory" />
      </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 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, `EgresoFugaWizardController` que gestiona el flujo del asistente para el registro de Egresos por fuga.

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*[15] 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 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.

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

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: Decisiones y Egresos.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

3.1 Introducción

En este capítulo se soluciona los módulos de Decisiones y Egresos de acuerdo a su Diseño e Implementación. Se utiliza como lógica de presentación en el capítulo la secuencia de actividades realizadas, en correspondencia con las definidas para la realización de un módulo dentro del SIGEP aunque no se representan todos los diagramas e implementaciones hechas sino solo una muestra representativa de las funcionalidades distintivas de ambos módulos.

Se muestran 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 Actividades y aspectos relevantes en el desarrollo de los módulos Decisiones y Egresos

El diseño e implementación de los módulos Decisiones y Egresos se realizó tomando como referencia la secuencia de actividades definidas para el desarrollo de un módulo del SIGEP explicada en el Capítulo 2 de este documento. Los artefactos generados como parte de estas actividades fueron los Modelos de Diseño e Implementación adaptados para el proyecto SIGEP a partir de la definición de los artefactos homónimos de la metodología RUP, analizada en el Capítulo 1.

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.

3.3 Análisis de las funcionalidades

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

Módulo Decisiones:

- registrar decisión,
- registrar documentos de la decisión,
- anular decisión,
- ratificar decisión,
- denegar decisión.

Módulo Egresos:

- registrar egreso por decisión judicial,
- registrar egreso por evasión,
- registrar egreso por defunción,
- registrar documentos que avalen la defunción del individuo,
- registrar presunta defunción del individuo,
- notificar egreso,
- notificar imposibilidad de la ejecución del egreso ordenado.

Para facilitar el trabajo de los usuarios de la aplicación, se decide utilizar asistentes que guíen paso a paso en las funcionalidades donde se ejecutan registros.

Por la implicación que genera un egreso dentro del sistema penitenciario venezolano se decide que el módulo Egresos notifique a todos los módulos del SIGEP la ejecución de un egreso independientemente del tipo que sea. En este proceder se utiliza el modelo de eventos del framework Spring.

3.4 Diseño del dominio

El modelo de dominio del módulo Decisiones se muestra en el Anexo 3 y el correspondiente a Egresos en el Anexo 4. En ambos se representa el diagrama del dominio y la descripción de las entidades y los nomencladores más significativos.

El módulo Decisiones tiene como principal elemento la jerarquía de decisiones que se pueden registrar en el sistema y toda la estructura de los documentos que incluye los tipos de remitentes y los tipos de documento. En la Figura 3.1 se muestra la jerarquía de decisiones.

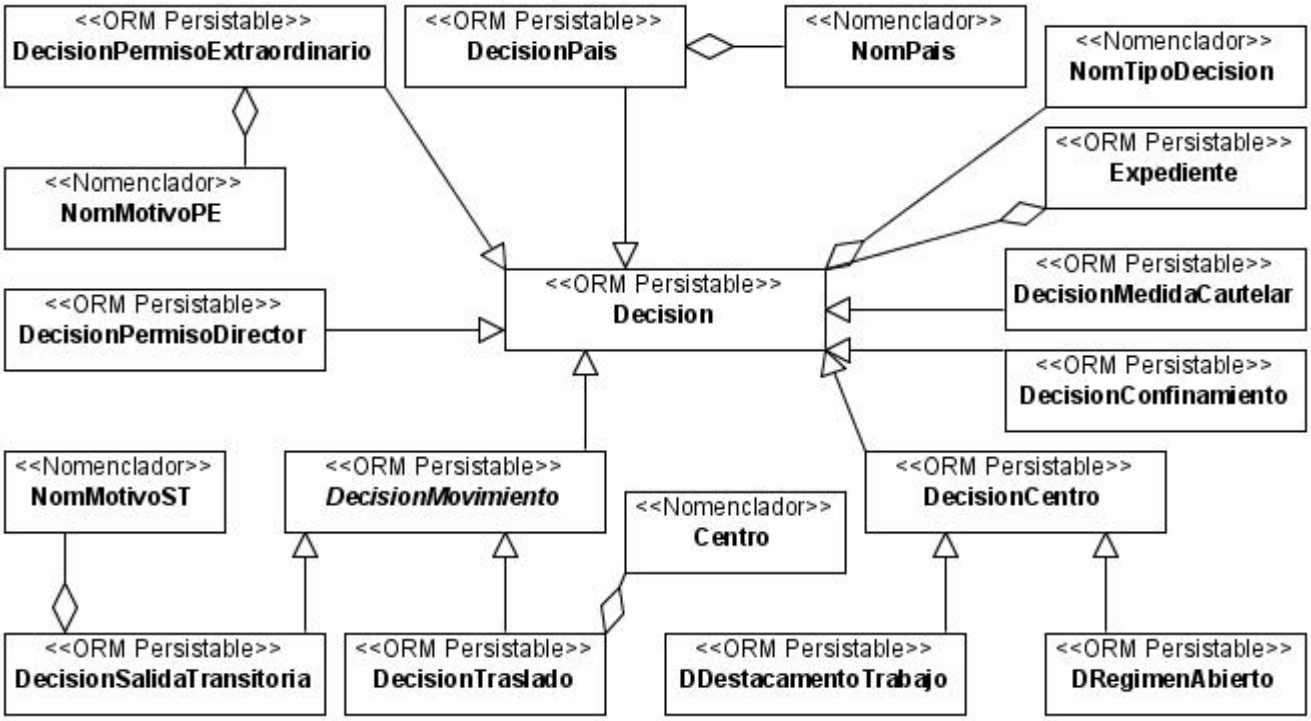


Figura 3.1 Jerarquía de Decisiones

El módulo Egresos se nutre de un dominio muy amplio. Algunas de estas entidades son gestionadas por los módulos Decisiones, Datos Personales y Situación Jurídica por lo que se le suman atributos necesarios al módulo en particular. Además, aporta nomencladores útiles para la programación de la interfaz como `NomLugarFugaEvasion` que contiene los posibles lugares de los cuales puede realizar la fuga un recluso. En la Figura 2.4 “Diagrama de clases persistentes del módulo Egresos” de la página 35 se muestra el dominio sobre el cuál interactúa el módulo.

3.5 Diseño del modelo de datos

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

Analizando el modelo de datos de Decisiones se observa la representación de las entidades persistentes que responden a un caso distintivo de ese módulo que es el registro de una decisión en el expediente de un individuo. Para que este registro sea efectivo, se debe realizar una secuencia de validaciones relacionadas con el tipo de centro en el cual se encuentra el individuo, su estado legal, el tipo de documento adjunto a la decisión y el tipo de remitente por cada uno de estos.

Para ilustrar la validación a ejecutar tomemos como referencia la decisión de *Traslado* Interpenal (clase `DecisionTraslado`). Una decisión cuyo tipo sea *Traslado Interpenal* puede ser otorgada a un

individuo que tenga cualquier *estado legal*[16] y que se encuentre en un Establecimiento Penitenciario o Régimen intramuros (clase `NomTipoCentro`). El único documento que puede traer asociado la orden, que además es requerido y *rector*[17], tiene que ser una Orden de Traslado (clase `NomTipoDocumento`) que solo puede ser emitida por un Tribunal de Ejecución, la DGSP o la Coordinación Regional (clase `EmisorReceptor`).

Tómese en cuenta que actualmente en el sistema venezolano se emiten alrededor de cuarenta tipos de decisiones que como en el caso del Traslado Interpenal requieren la verificación del estado legal del individuo, el tipo de centro y los tipos de documento con el tipo de remitente para poder ser registrada. La definición de estas restricciones puede estar sometida a modificaciones por la ley venezolana que en medio de profundos cambios sociales tiende a la variación. Para contrarrestar esta situación se decidió mantener la especificación de estas restricciones alejadas de la implementación del negocio y se definió una estructura en la base de datos que puede ser gestionada desde la aplicación en caso de modificación de la ley, gestionando de esta manera el riesgo de que se generen, modifiquen o eliminen restricciones del sistema penitenciario. Así el módulo Decisiones aporta al modelo de datos del SIGEP toda la jerarquía de decisiones, la estructura de los documentos y las tablas que nomenclan por cada tipo de decisión su relación con el estado legal del individuo, el tipo de centro y los tipos de documentos y de estos últimos con los tipos de remitentes como lo refleja la Figura 3.2.

[16] Estado Legal: Estado que tiene un individuo dentro del sistema penitenciario de acuerdo a sus procesos abiertos (procesado), ejecuciones por cumplir (penado) o ambos.
[17] Documento rector: Documento que marca la fecha de registro de la decisión. Es un documento requerido para certificar la validez de una decisión.

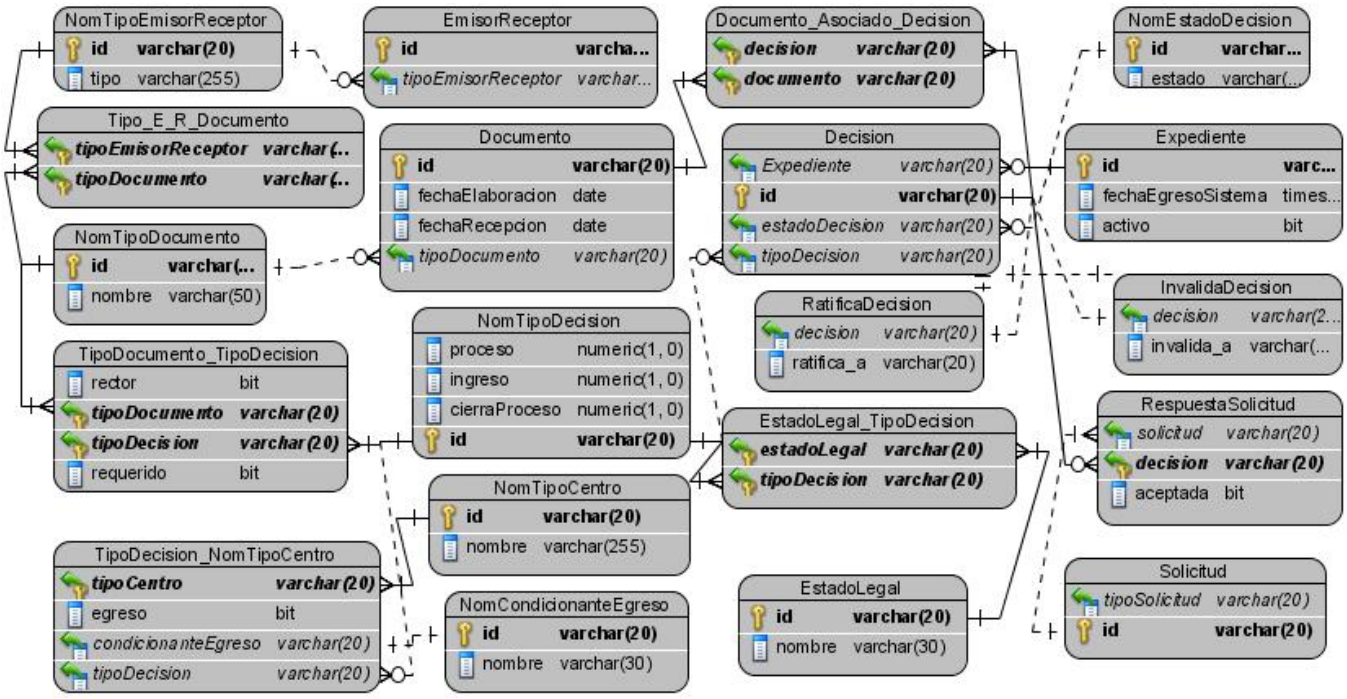


Figura 3.2 Fragmento del modelo de datos Decisiones. Relación: tipo de centro - tipo de documento - estado legal - tipo de decisión.

El módulo Egresos, por su parte, aporta pocas tablas al modelo físico del SIGEP, más bien agrega una serie de campos en tablas ya definidas por los módulos Situación Jurídica, Decisiones y Datos Personales para garantizar el cierre de la estancia, expediente y marcar la defunción del individuo (Figura 3.3). Sin embargo, para su funcionamiento, el módulo necesita recuperar información de la mayoría de las tablas de los módulos del subsistema de Control Penal como lo representa la Figura 2.5 referenciada al inicio de este epígrafe.

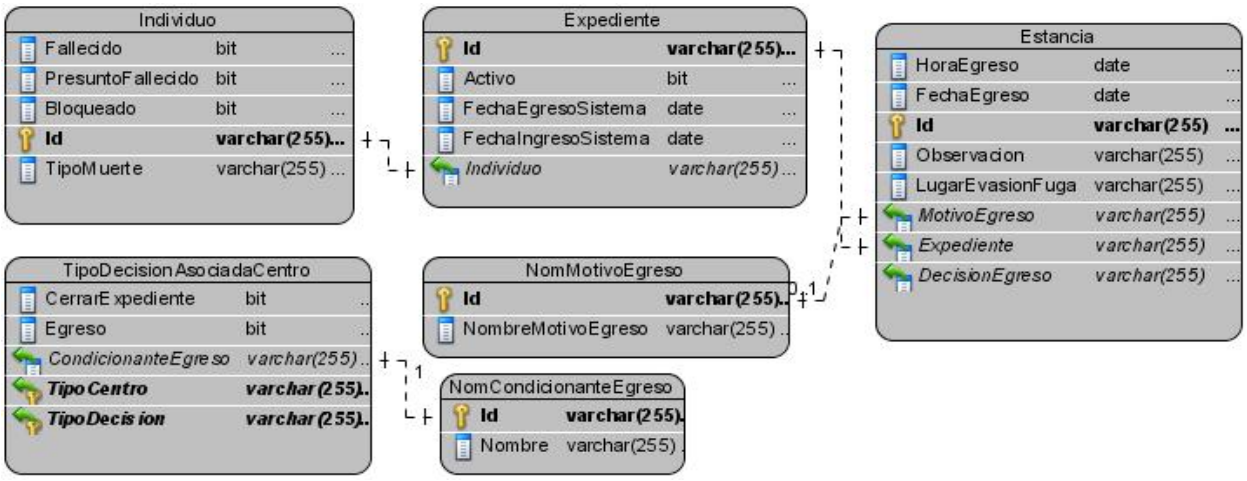


Figura 3.3 Tablas y campos que adiciona Egresos al modelo de datos del SIGEP

3.6 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 el resto de los módulos de Control Penal, 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 Decisiones y Egresos son: el manejo de excepciones, las transacciones y el manejo de eventos.

Manejo de excepciones

La capa de negocio como capa media y donde reside la implementación de los requisitos del negocio tiene que verificar la integridad de los datos que a ella llegan y que de ella salen. Así en el caso de eventualidades a notificar a otras capas lo más recomendado es el lanzamiento de excepciones que puedan ser capturadas por la capa de presentación y mostradas al usuario de una forma adecuada.

El SIGEP tiene un grupo de excepciones generales definidas para problemas en entradas y salidas de datos, sin embargo, en cada módulo se debe definir un conjunto de excepciones que puedan ser mostradas al usuario de forma particular. Ejemplo de ello son las excepciones definidas para el módulo Egresos: *IndividuoEgresadoException* y *NoEgresoException* y la excepción *DocumentoRequeridoException* del módulo Decisiones que son lanzadas cuando un individuo ya ha sido egresado del sistema, existe un motivo para que el egreso no se realice o falta algún documento requerido para el registro de una decisión, respectivamente.

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 Decisiones y Egresos: `DocumentoManagerImpl, DecisionManagerImpl y EgresoManagerImpl` 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`*[18] 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(* *..*DecisionManagerImpl.*(..))"
            advice-ref="decisionMgrAdvice" />
</aop:config>
<tx:advice id="decisionMgrAdvice">
   <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 `DecisionManagerImpl`. Con esta configuración los métodos de la clase `DecisionManagerImpl` se ejecutarán en contextos transaccionales sin siquiera ser conscientes de ello.

[18] 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.

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: `* *..*DecisionManagerImpl.*(..)`. El primer `*` significa *cualquier tipo de retorno*; la expresión `*..*DecisionManagerImpl` significa *cualquier clase cuyo nombre termine en* `DecisionManagerImpl` 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. Como se ha mencionado en varias ocasiones, los módulos Decisiones y Egresos, inician otros procesos cuando se hayan ejecutado las funcionalidades de registro de una decisión y el egreso a un individuo. Cuando en el sistema se ejecuta el registro de una *decisión de traslado* o de *salida transitoria* por la interfaz del módulo Decisiones, los módulos Traslados y Salidas del subsistema Seguridad y Custodia inmediatamente establecen una planificación para la ejecución de estas órdenes. El registro del egreso de un individuo en el módulo Egresos implica que los módulos Educación, Trabajo, Cultura y Deporte del subsistema Tratamiento ejecuten acciones de desincorporación del individuo de las actividades docentes, unidades productivas, actividades deportivas y culturales.

En consecuencia, es necesario que los componentes de la capa de negocio lancen eventos que otros componentes deberán escuchar para ejecutar acciones. Para el caso del módulo Decisiones el componente que lanza el evento `RegistroDecisionEvent` es el objeto de negocio `DecisionManagerImpl`, para Egresos el evento `EgresoIndividuoEvent` lo publica el objeto `EgresoManagerImpl`.

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

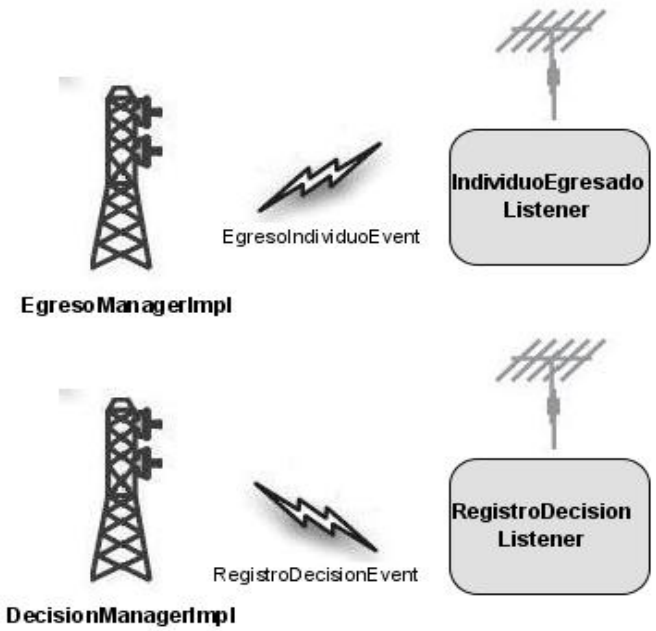


Figura 3.4 Utilización del modelo de eventos de Spring en los módulos Decisiones y Egresos.

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.

A continuación se describe el procedimiento para el manejo de eventos que se siguió en los módulos Decisiones y Egresos.

Crear el 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.events;
...
public class EgresoIndividuoEvent extends ApplicationEvent {
	/**
	 * Individuo egresado.
	 */
	private Individuo individuo;
	/**
	 * Fecha del egreso.
	 */
	private Date fechaEgreso;
	/**
	 * Motivo del egreso.
	 */
	private NomMotivoEgreso motivoEgreso;
	...
	public EgresoIndividuoEvent(Object fuente, Individuo individuo, ...){
	super(fuente);
		this.individuo = individuo;
		...
	}
	...
}
```

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 {
	...
	/**
	 * Referencia al contenedor de Spring.
	 */
	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(Individuo individuo, Date fecha, ...) {
	ApplicationEvent evento =
				new EgresoIndividuoEvent(this, individuo, 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 una clase interesada en escuchar el evento `EgresoIndividuoEvent` pudiera ser como sigue:

```
package vnz.sigep.subsistema.modulo.listener;
...
public class EventListener implements ApplicationListener {
      private ModuloFacade moduloFacade;
      public void onApplicationEvent(ApplicationEvent event) {
            if (event instanceof EgresoIndividuoEvent){
                  // Implementación del manejo del evento.
            }
      }
      ...
}
```

3.7 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.

En el escenario Registrar egreso por decisión judicial del módulo Egresos es necesario mostrar al usuario los individuos que pueden ser egresados en una fecha determinada porque tuviesen registrada una decisión pendiente de ejecución que implica egreso en ese tipo de centro o porque tuviera algún cómputo de la pena que marcara como fecha de cumplimiento esa fecha u otra anterior.

Decidir cuales individuos pueden egresar en una fecha determinada constituye un aporte esencial del SIGEP de ayuda a sus usuarios pues sustituye un tedioso proceso manual de revisión de expedientes. Para obtener estos datos se determinó la implementación de funciones, que radicaran en el gestor, de manera que se pudiese garantizar un acceso rápido a esa información, teniéndose en cuenta que son consultadas más de quince 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 Egresos.

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.

3.8 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, Spring-MVC y Jasper Reports. Para la implementación de esta capa en los módulos Decisiones y Egresos 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.

3.8.1 Diseño de la funcionalidad Registrar decisión

Para el diseño de la funcionalidad *Registrar Decisión* (Ver Anexo 8.1.1) se encuentran características especiales. Se debe brindar la posibilidad al usuario de registrar decisiones de varios tipos, con datos específicos para cada una, en relación al tipo de decisión. Por ejemplo, de la decisión *Permiso otorgado por el director* interesa registrar las fechas de inicio y fin del permiso, el motivo del permiso y la dirección en que se encontrará el individuo; de la decisión de *Extradición* se necesita registrar el país al que se extraditará el individuo y la fecha de la extradición y así por cada una de las restantes.

En el momento en que se diseña el módulo Decisiones no se tiene conocimiento de todos los posibles tipos de decisiones y los datos que se necesitan registrar de cada uno de ellos y, aunque no es un requisito permitir crear nuevos tipos de decisiones en tiempo de ejecución, sí se necesita hacer un diseño lo suficientemente flexible que permitiera incluir, en futuras iteraciones, nuevos tipos de decisión sin modificar el código escrito anteriormente.

La aparición de nuevos tipos de decisión 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 decisiones y que se crearán nuevos en el futuro pues esta capa solo conoce la base de la jerarquía de decisiones 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 “Asistente para el registro de decisiones”, en efecto la utilización de *asistentes* es muy común en aplicaciones web. En la primera página del asistente se registrarán datos generales de las decisiones como el tipo de decisión, la fecha en que fue emitida y los documentos que la respaldan; en la segunda página del asistente se registrarán los datos específicos de acuerdo al tipo de decisión.

Para la gestión del asistente 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 cada una de las páginas del asistente con los datos que estas necesiten, de manejar el flujo de navegación y de recolectar los datos introducidos por el usuario y con ellos crear una instancia de la decisión correspondiente para ser enviada a la capa de negocio.

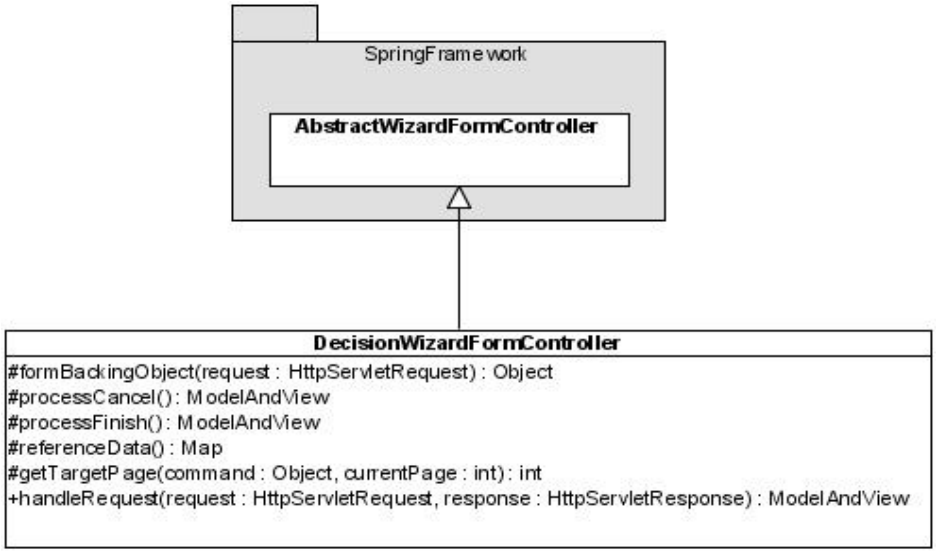


Figura 3.5 Uso de la clase AbstractWizardFormController de Spring-MVC en la creación de asistentes

En el método `getTargetPage`, el controlador debe determinar la página del asistente que se va a mostrar en dependencia del tipo de decisión que se esté registrando. 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 cada página del asistente. Estos datos dependen del tipo de decisión que se esté registrando. En el método `referenceData`, el controlador debe devolver datos adicionales que se necesitan para la construcción de la página a mostrar. Como se observa, estos tres métodos tendrían que modificarse en iteraciones posteriores al introducir nuevos tipos de decisiones si no se realiza un diseño suficientemente flexible. La clase `DecisionWizardFromController` debe delegar parte de la implementación de estos tres métodos en clases específicas para cada tipo de decisión.

En el siguiente diagrama de secuencia se muestra la solución propuesta para el método `formBackingObject`.

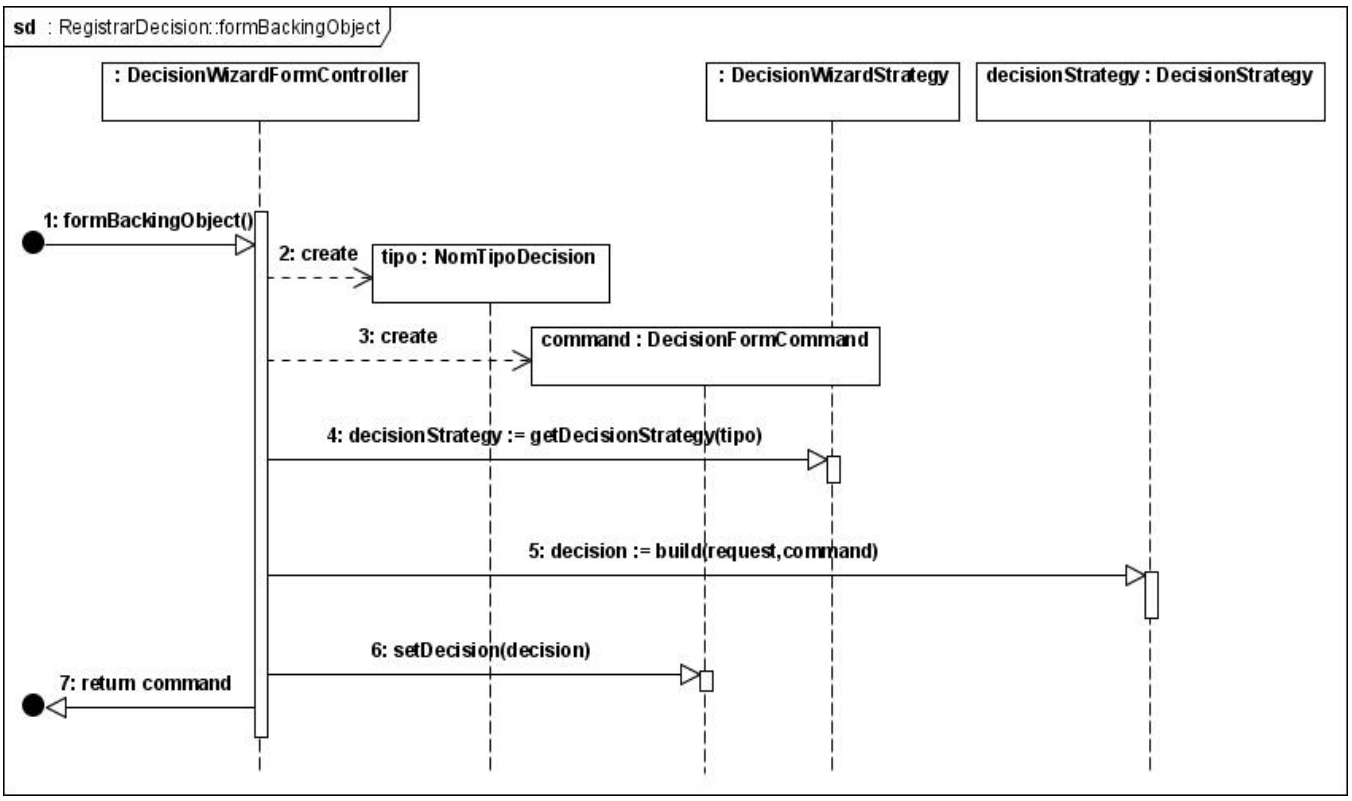


Figura 3.6 Diagrama de secuencia del método FormBackingObject

Apoyándonos en el patrón Estrategia se obtiene una instancia de una implementación de la interfaz `DecisionStrategy` dado el tipo de decisión. La interfaz `DecisionStrategy` tiene el método `build` que construye una decisión específica dada la petición del usuario y con ella se construye finalmente el command. De forma similar se procede para los métodos `getTargetPage` y `referenceData` agregando métodos `getTargetPage` y `referenceData` a la interfaz `DecisionStrategy`. De esta manera, el controlador no necesita conocer el tipo de decisión que se está registrando sino que, para cada tipo de decisión, obtiene una instancia de `DecisionStrategy` en la que delegará las tareas específicas. Finalmente se debe señalar que no es necesario programar la clase DecisionWizardStrategy mostrada en el diagrama de secuencia de la Figura 3.6, porque gracias a la técnica de inyección de dependencias que brinda Spring, la propia clase `DecisionWizardFromController` puede cumplir esta función de forma transparente. Para ello define un atributo **`Map<NomTipoDecision, DecisionStrategy> estrategias`**, a través del cual podrá determinar, dado un tipo de decisión, la instancia de `DecisionStrategy` correspondiente.

Dado el diseño propuesto, agregar nuevos tipos de decisiones se reduce a crear la nueva página del asistente e implementar la interfaz `DecisionStrategy`. A continuación se muestra cómo sería este proceso tomando como referencia la decisión de *Traslado Interpenal*.

Crear la nueva página del asistente

Se crea la página `decision-traslado.jsp` y se inyecta el nombre lógico de la vista al controlador de la siguiente forma:

```
<property name="pages">
   <list>
      <value>decision-form</value>
      <value>decision-traslado</value>
      ...
   </list>
</property>
```

Implementar la interfaz DecisionStrategy

Ejemplo de la implementación de la interfaz `DecisionStrategy`

```
package vnz.sigep.controlpenal.decisiones.web.decisionwizard;
...
public class DecisionTrasladoStrategy implements DecisionStrategy {
      /**
       * Fachada del módulo.
       */
      private DecisionesFacade decisionesFacade;
      ...
      /**
       * Crea la decisión de traslado dada la petición del usuario.
       */
      public Decision build(HttpServletRequest request, DecisionFormCommand command)
{
            DecisionTrasladoInterpenal decision = new DecisionTrasladoInterpenal();
            ...
            if (command.getDecision() instanceof DOrdenTrasladoInterpenal) {
                  DOrdenTrasladoInterpenal decisionCommand =
                        (DOrdenTrasladoInterpenal) command.getDecision();
                  String idCentro = "-1";
                  if (decisionCommand.getDestino() != null)
                        idCentro = decisionCommand.getDestino().getId();
                  decision.setDestino(new Centro(idCentro));
                  decision.setFechaTraslado(decisionCommand.getFechaTraslado());
                  decision.setMotivo(decisionCommand.getMotivo());
            } else {
                  decision.setDestino(new Centro("-1"));
            }
            ...
            return decision;
      }
```

```
	/**
	 * Devuelve el índice de la página que contiene el formulario de la decisión
	 * de traslado en el arreglo de páginas.
	 */
	public int getTargetPage() {
		return 1;
	}
	/**
	 * Devuelve los datos adicionales necesarios para crear la página del
formulario
	 * de decisión de traslado.
	 */
	public Map<String, Object> referenceData(HttpServletRequest request,
			DecisionCommand command) throws Exception {
		// obtener posibles centros a trasladar
		DecisionTrasladoInterpenal decisionTraslado =
					(DecisionTrasladoInterpenal)
					command.getDecision();
		Centro centro = decisionTraslado.getCentro();
		List<Centro> centros =
				decisionesFacade.listarPosiblesCentrosATrasladar(centr
			o);
		// devolver datos necesarios para crear la página
		Map<String, Object> data = new HashMap<String, Object>();
		data.put("centrosDestino", centros);
		data.put("motivosTraslado", decisionesFacade.listarMotivosTraslado());
		return data;
	}
}
```

Luego se inyecta una instancia de la clase creada al atributo `estrategias` del controlador:

```
<property name="estrategias">
   <map>
      ...
      <entry key-ref="tipoDecisionTraslado" >
         <bean class="vnz.sigep.controlpenal.decisiones.web.

                                        decisionwizard.DecisionTrasladoStrategy">
            ...
         </bean>
      </entry>
      ...
   </map>
</property>
```

El bean `tipoDecisionTraslado` puede ser declarado en el fichero de configuración del contexto de Spring para la capa de negocio de la siguiente forma:

```
<bean id="tipoDecisionTraslado"
   class="vnz.sigep.common.global.domain.NomTipoDecision">
      <constructor-arg index="0">
         <value>${tipo.decision.traslado.id}</value>
```

```
      </constructor-arg>
</bean>
```

Como se observa, para crear la instancia de `NomTipoDecision` es necesario el identificador del tipo de decisión de traslado en la base de datos. Este dato se declara en un fichero `.properties` bajo la clave `tipo.decision.tralado.id` para luego ser referenciado como se muestra en el ejemplo.

3.8.2 Notificaciones de egreso

Una funcionalidad del módulo Egresos es la notificación de la realización de un egreso o de la imposibilidad de su ejecución desde el centro penitenciario a cada una de las entidades relacionadas del poder judicial, civil o familiares. El envío de las notificaciones no es un requisito definido para el SIGEP, al menos en su primera versión, debido esencialmente a que no existe una infraestructura adecuada para ello.

Como solución al requisito de la notificación se decidió que el módulo Egresos generara la información necesaria para la notificación, permitiendo su impresión por el usuario del sistema o su descarga en formato PDF. La información de la notificación y los posibles destinatarios es dependiente del tipo de egreso realizado y, aunque en la mayoría de los casos esta información es muy básica, en otros implica el resumen de toda la estancia del individuo.

En la Figura 3.7 se muestra la interfaz gráfica desde la cual se genera a cada uno de los posibles destinatarios, en dependencia del tipo de egreso, las notificaciones necesarias.

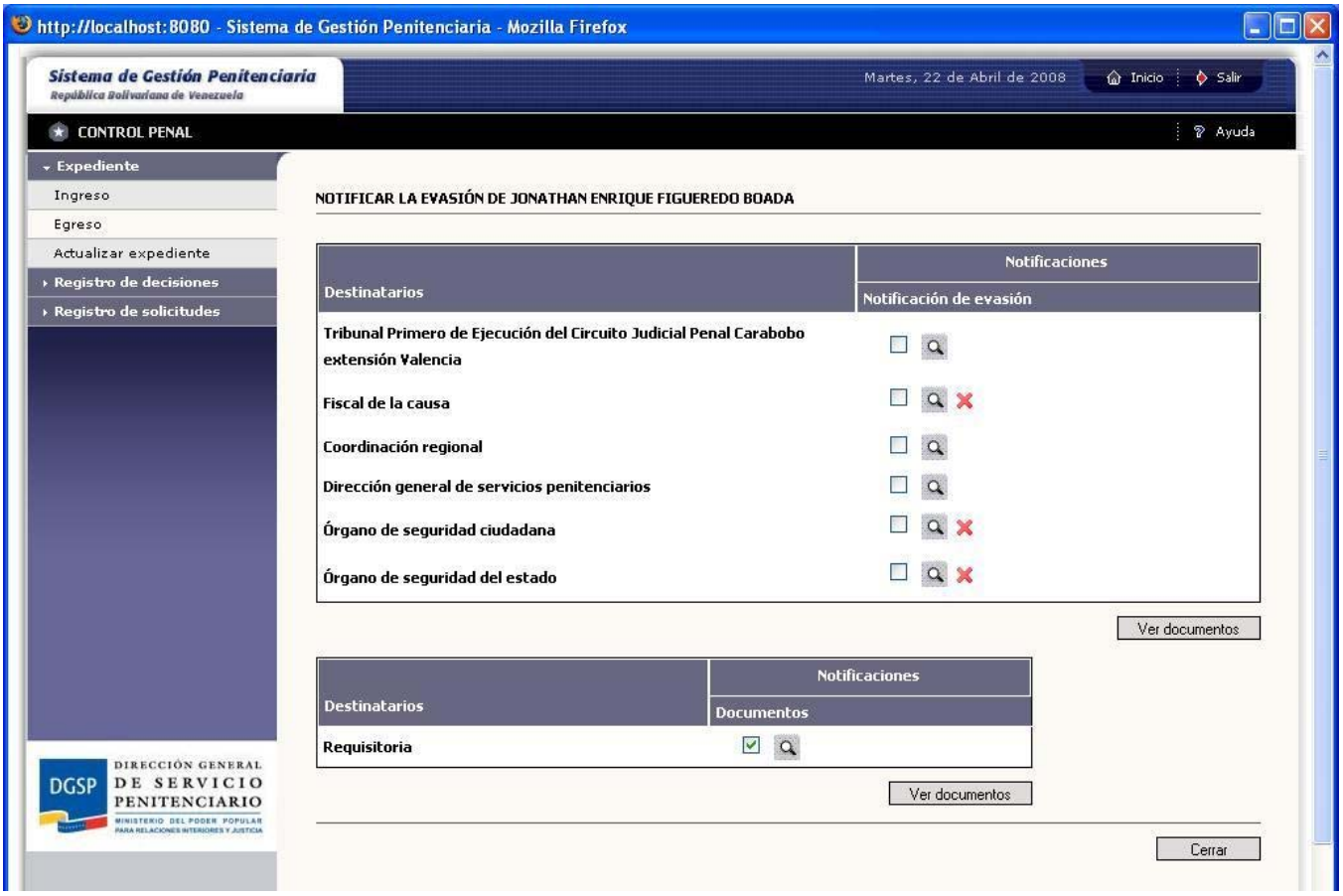


Figura 3.7 Interfaz gráfica de la notificación evasión de un individuo de Egresos

Para la generación de las notificaciones en formato PDF se utilizó la biblioteca de clases JasperReports especificada en el Capítulo 1. Spring-MVC provee integración con *JasperReports* a través de las vistas que define en el paquete `org.springframework.web.servlet.view.jasperreports`, su utilización desde un controlador del módulo Egresos se muestra en la Figura 3.8.

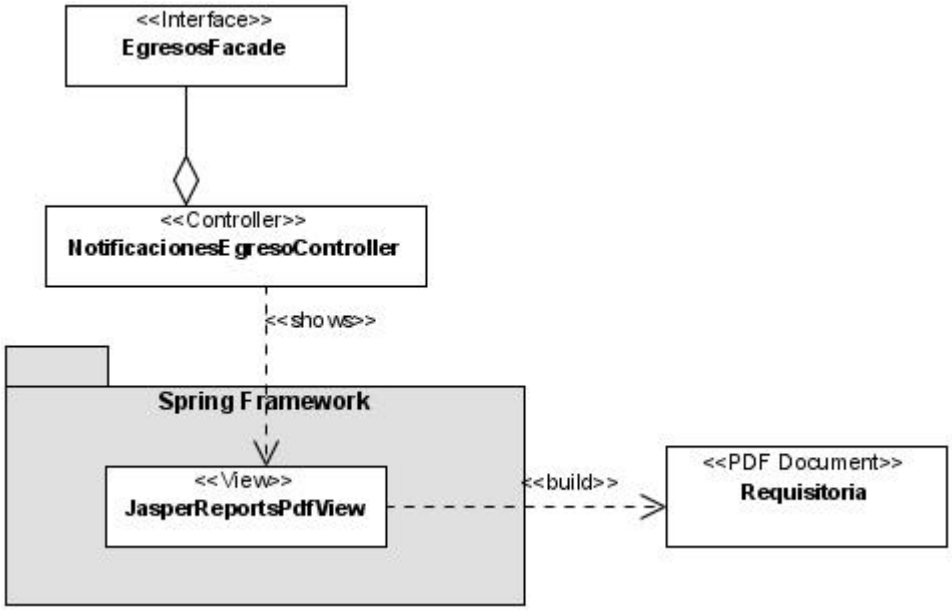


Figura 3.8 Clases involucradas en la generación de una Requisitoria

La generación de esta información por el SIGEP sustituye un proceso manual de búsqueda de información y de la confección final del documento. Para ejemplificar con un caso real vamos a analizar el caso de la *Requisitoria*, documento que tiene que ser emitido por el centro penitenciario y enviado a todas las instituciones cuando un individuo se fuga o evade del mismo (egreso por evasión o fuga). La Requisitoria es un documento que resume la situación legal del individuo y expone sus principales rasgos físicos y su foto. En la Figura 3.9 se muestra una Requisitoria generada por el módulo Egresos lista para imprimir, en formato PDF.

DIRECCIÓN GENERAL DE SERVICIOS PENITENCIARIOS

OFICIO No: 1401 | FECHA: 22/04/2008

CENTRO: Centro Penitenciario Carabobo

REQUISITORIA

Requisitoria librada en contra del individuo: Jonathan Enrique Figueredo Boada, titular del documento de identidad Nro. V1719107522, quien se evadiera de Centro Penitenciario Carabobo, en fecha 22/04/2008.

DATOS FILIATORIOS

APELLIDOS Y NOMBRE: Jonathan Enrique Figueredo Boada
SOBRENOMBRE O APODO: no tiene
DOCUMENTO DE IDENTIDAD: V1719107522
EDAD: 24
NOMBRE DE LOS PADRES: Iraima Boada y Jose Figueredo
NOMBRE DE LA ESPOSA(O) O CONCUBINA(O): No registrado.
DIRECCION: Carabobo Valencia Santa Rosa Calle vereda 16 Avenida Sector 8 Urbanización La Isabelica
DELITO(S): Robo,

TRIBUNAL: Tribunal Primero de Ejecución del Circuito Judicial Penal Carabobo extensión Valencia,

DATOS FÍSICOS

BARBA: no tiene
CABELLO TIPO Y FORMA: chicharrón
COLOR DEL CABELLO: negro
CARA: grande
NARIZ: grande
OREJAS: pequeñas
OJOS TIPO Y DEFECTOS: achinados y no tiene
COLOR DE OJOS: pardo oscuro
PESO: 72.5 Kgs. ESTATURA: 172.0 m.
BIGOTE: escaso
BOCA: grande
CEJAS: pobladas
FRENTE: corta
PIEL COLOR: moreno

DESCRIPCIÓN DEL HECHO:

DGSP
DIRECCIÓN GENERAL DE SERVICIO PENITENCIARIO

Ave. Urdaneta, Esq
Bolero. Caracas
Dirección de Servicios Penitenciarios
Instituto Autónomo de Sistema Penitenciario
República Bolivariana de Venezuela

Se agradece a todas las autoridades civiles y militares del país su mayor colaboración en la captura del evadido a objeto de reintegrarlo a este establecimiento.

Figura 3.9 Requisitoria generada por el módulo Egresos

3.9 Integración de los módulos Decisiones y Egresos al SIGEP

Una vez concluida la implementación y prueba por el equipo de calidad de los módulos Decisiones y Egresos, 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 3.10 se muestra el diagrama de componentes de Control Penal donde los módulos Decisiones y Egresos ya empaquetados se comunican entre ellos a través de interfaces.

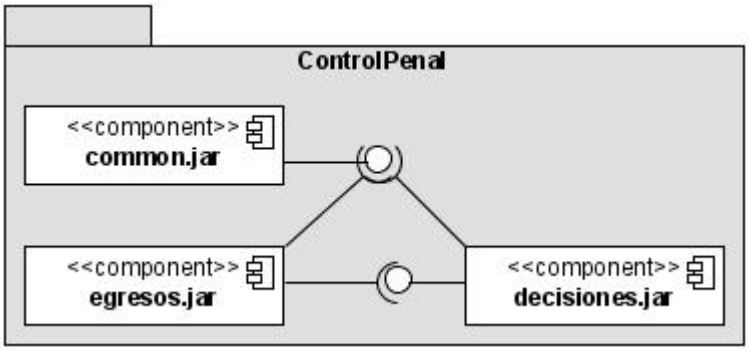


Figura 3.10 Decisiones y Egresos como componentes del SIGEP

3.10 Análisis de resultados de validaciones con el cliente

Los módulos Decisiones y Egresos 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 diciembre de 2007.

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 3.11. La mayoría de estas no conformidades han estado relacionadas con las interfaces gráficas de los módulos.

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

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

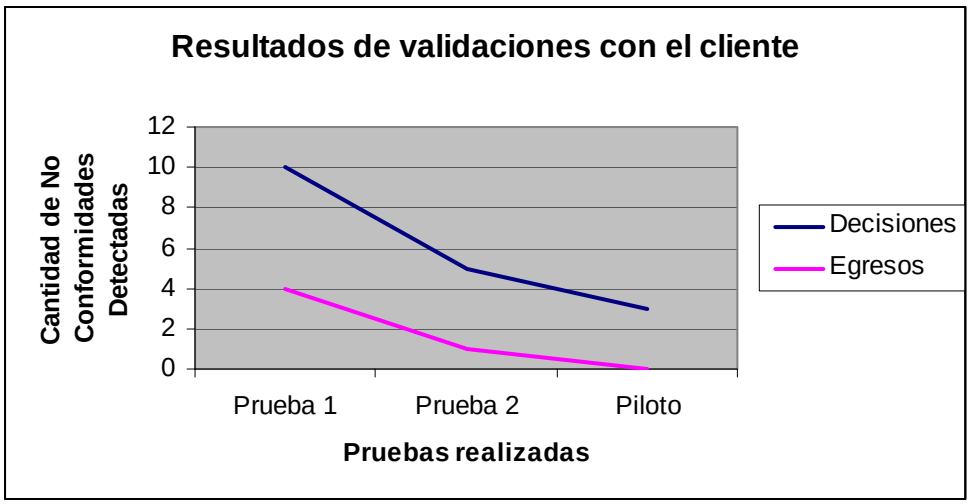


Figura 3.11 Resultados de validaciones con el cliente

3.11 Conclusiones

El análisis, diseño e implementación de los módulos Decisiones y Egresos 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 los individuos a egresar en una fecha determinada y la generación de documentos oficiales como las notificaciones de egreso y las Requisitorias.

CONCLUSIONES

Como resultado del presente trabajo, se realizó el diseño y la implementación de los módulos Decisiones y Egresos, 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 primera 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.

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

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.

KAISLER, STEPHEN. H. *Software Paradigms*. 2005.

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

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

PÉREZ, E. A. *Conceptualización del proceso Control Penal*, 2007.

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*, 2003a.

VEGA, YANET y otros. *Control Penal: Procesos Elementales del negocio*, ALBET, 2007.

VEGA, YANET. *Descripción del prototipo de Interfaz de usuario. Subsistema control penal. Módulo Decisiones*, ALBET SIGEP-SW-6, 2007.

VEGA, YANET. *Descripción del prototipo de Interfaz de usuario. Subsistema control penal. Módulo Egreso,* ALBET SIGEP-SW-24, 2007.

VEGA, YANET y BETANCOURT, ARACELIS *Procedimiento para desarrollar la Ingeniería de Requisitos en el proyecto Sistema de Gestión Penitenciaria*, UCI, 2007.

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

ANEXOS

ANEXO 1. Implementación del método *registrarEgresosPorDecisionJudicialIndividuo* del manager EgresoManagerImpl del módulo Egresos.

```
public void registrarEgresoPorDecisionJudicialIndividuo(
            DatosEgresoDecisionJudicial datosEgresoDecisionJudicial)
                  throws Exception {

      InputAssert.notNull(datosEgresoDecisionJudicial,
                                    "Los datos del egreso no pueden ser nulos");

      // individuo al cual se le otorga el egreso
     Individuo individuo = datosEgresoDecisionJudicial.getIndividuo();

     InputAssert.notNull(individuo,
                  "El egreso tiene que estar asociado a un individuo.");
     InputAssert.notNull(individuo.getId(),
      "El individuo al cual se le otorga el egreso tiene que estar registrado.");

      // expediente activo del individuo
      Expediente expedienteActivo =
      individuoManager.obtenerExpedienteActivo(individuo);

      if (expedienteActivo == null)
            throw new NoEgresoException(". El individuo ya ha sido egresado del
sistema. No tiene expediente abierto en el sistema.");

      // estancia abierta en el expediente activo del individuo
      Individuo individuoBD = expedienteActivo.getIndividuo();

      if (individuoBD == null)
            throw new IndividuoEgresadoException();

      if (individuoBD.getPresuntoFallecido())
            throw new NoEgresoException(". El individuo aparece registrado como
presunto fallecido.");

      Date fechaEgreso = datosEgresoDecisionJudicial.getFechaEgreso();
      Date horaEgreso = datosEgresoDecisionJudicial.getHoraEgreso();

      // id de la decisión por la que se otorgó el egreso
      String idDecision = datosEgresoDecisionJudicial.getIdDecision();

      Estancia estanciaAbierta =
            estanciaManager.buscarEstanciaAbiertaPorExpediente(expedienteActivo);

      if (estanciaAbierta == null)
            throw new NoEgresoException(". El individuo ya ha sido egresado del
sistema. No tiene estancia abierta en el expediente activo.");

      // si el egreso es por cumplimiento de pena
      if (datosEgresoDecisionJudicial.getCumplimientoPena()) {
          // poniendo como motivo de egreso , egreso por cumplimiento de la pena
```

```
            estanciaAbierta.setMotivoEgreso(new  NomMotivoEgreso(
                  nomencladoresManager.buscarValorDeProperty("motivoegreso.cumplimie
              ntopena.id")));
            expedienteActivo.setActivo(false);
            expedienteActivo.setFechaEgresoSistema(fechaEgreso);

            // persistir cambios al expediente
            expedienteDAO.persist(expedienteActivo);

      }
      else { // el egreso es por decisión judicial
            OutputAssert.hasText(idDecision,"Tiene que existir una decisión para dar
un egreso ordenado.");

            // la decisión por la que se va a dar el egreso
            Decision decision =  ordenManager.buscarDecision(idDecision);

            if (decision == null || !decision.getEstadoDecision().getId().equals(
            nomencladoresManager.buscarValorDeProperty("decision.estado.pendiente.id
      ")))
                  throw new NoEgresoException(". La decisión seleccionada no está
registrada como pendiente de ejecución.");

            // ejecutando la decisión
            if(decision.getTipoDecision().getId().equals(nomencladoresManager.
                  buscarValorDeProperty("tipodecision.confinamiento"))){
                  NomEstadoProgresividad estadoProgresividad = new
                        NomEstadoProgresividad(nomencladoresManager.
                        buscarValorDeProperty("estadoprogresividad.confinamiento"));

                  Progresividad progresividad = new Progresividad();
                  progresividad.setEstadoProgresividad(estadoProgresividad);
                  progresividad.setFecha(fechaEgreso);
                  progresividad.setExpediente(expedienteActivo);

                  progresividadDAO.persist(progresividad);
            }

            ordenManager.setEstadoDecision(decision,EstadoDecisionEnum.EJECUTADA);

            // la estancia siempre se cierra, el expediente no
            estanciaAbierta.setDecisionEgreso(decision);

            // poniendo como motivo de egreso, egreso por decisión judicial
            estanciaAbierta.setMotivoEgreso(new
NomMotivoEgreso(nomencladoresManager.
                  buscarValorDeProperty("motivoegreso.decisionjudicial.id")));

            NomTipoCentro tipoCentroActual = centroManager.
                                          obtenerCentro().getTipoCentro();

            Boolean accionExpediente = tipoDecisionAsociadaCentroDAO.
                        findAccionExpedienteByTipoDecisionTipoCentro(
                              decision.getTipoDecision(), tipoCentroActual);

            // acción sobre el expediente, true hay que cerrar el expediente
```

```
            if (accionExpediente != null && accionExpediente) {
                  expedienteActivo.setActivo(false);
                  expedienteActivo.setFechaEgresoSistema(fechaEgreso);

                  // cambios al expediente
                  expedienteDAO.persist(expedienteActivo);
            }
      }

      // cierre de la estancia
      estanciaAbierta.setFechaEgreso(fechaEgreso);
      estanciaAbierta.setHoraEgreso(horaEgreso);

      // bloquear al individuo
      individuoBD.setBloqueado(false);
      individuoManager.actualizarIndividuo(individuoBD);

      // persistir los cambios a la estancia
      estanciaManager.actualizarEstancia(estanciaAbierta);

      //evento que notifica la ocurrencia del egreso de un individuo
      this.notificarEgresoIndividuo(individuoBD);

}
```

ANEXO 2. Implementación del controlador *EgresoFugaWizardController*

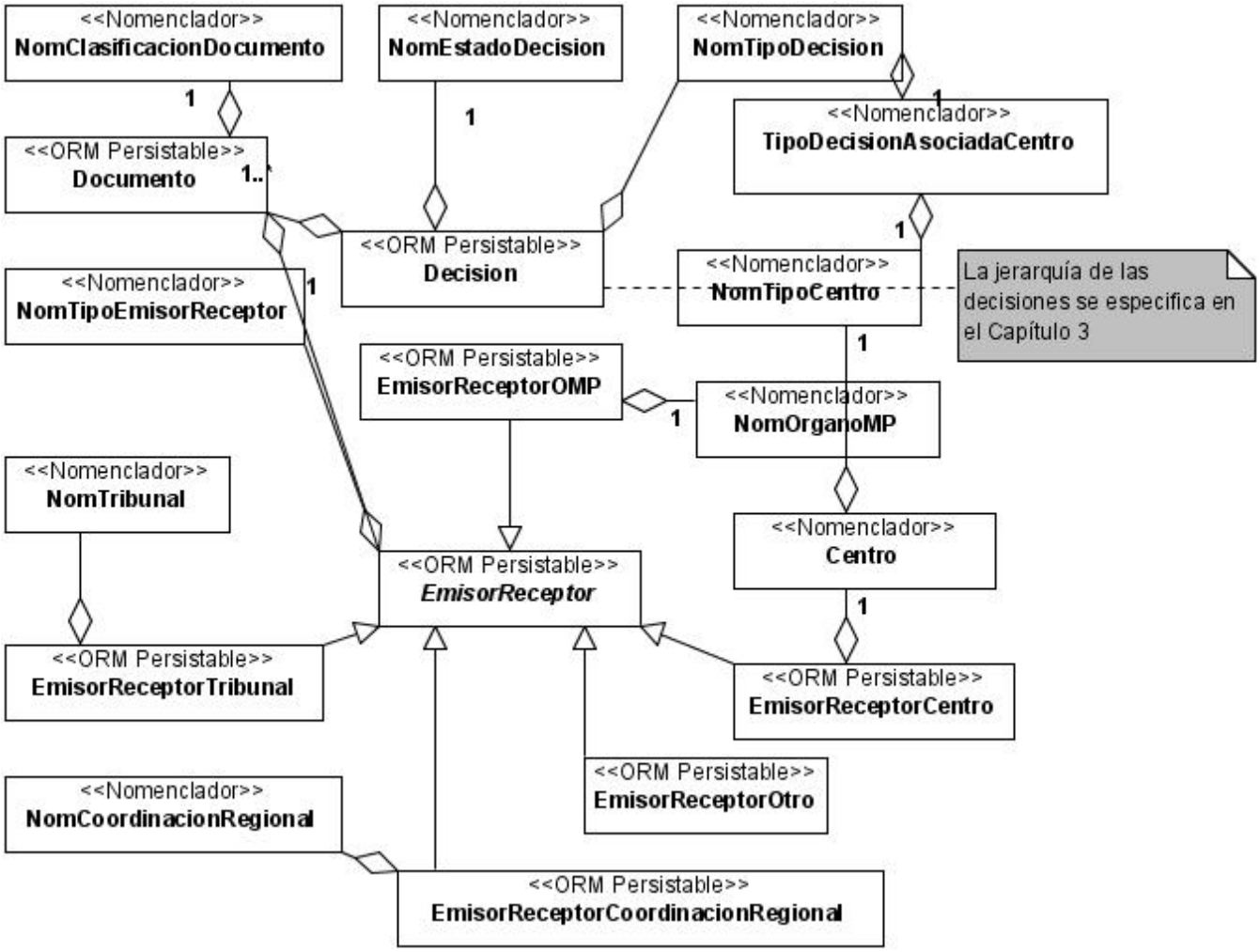
```
public class EgresoFugaWizardController extends AbstractWizardFormController {
      private EgresosFacade egresosFacade;
      ...
      public EgresoFugaWizardController() {
            super();
            setCommandClass(EgresoCommand.class);
            setCommandName("fugaCommand");
      }

      protected ModelAndView processFinish(HttpServletRequest request,
                  HttpServletResponse response, Object command, BindException
errors)
                  throws Exception {

            EgresoCommand commandE = (EgresoCommand) command;
            String fecha = commandE.getFecha();
            String hora = commandE.getHora();
            DateFormat df = new SimpleDateFormat("dd/MM/yyyy");
            DateFormat hf = new SimpleDateFormat("HH:mm:ss");
            Date fechaEgreso = null;
            Date horaEgreso = null;

            try {
                  fechaEgreso = df.parse(fecha);
                  horaEgreso = hf.parse(hora);
            } catch (Exception e) {}

            Individuo individuo = new Individuo(commandE.getIdIndividuo());
```

```
		// datos del egreso por evasión
		DatosEgresoEvasionFuga datosEgreso = new DatosEgresoEvasionFuga();
		datosEgreso.setDescripcionSuceso(commandE.getDescripcion());
		datosEgreso.setFechaEgreso(fechaEgreso);
		datosEgreso.setHoraEgreso(horaEgreso);
		datosEgreso.setIndividuo(individuo);
		if (StringUtils.hasText(commandE.getLugarFugaEvasion())
				&& !commandE.getLugarFugaEvasion().equals("-1"))
			datosEgreso.setLugarEvasionFuga(new NomLugarFugaEvasion(commandE
					.getLugarFugaEvasion()));
		else
			datosEgreso.setLugarEvasionFuga(null);
		try {
			egresosFacade.registrarEgresoPorFugaIndividuo(datosEgreso);
		} catch (NoEgresoException e) {
		return new ModelAndView("egreso/failed", "mensaje", e.getMessage());
		}

		// todo lo necesario para las notificaciones del egreso por fuga
		Map datosNotificacion = new HashMap();
		// en este caso es notificación de fuga y requisitoria
		datosNotificacion.put("nombreIndividuo", commandE.getNombreIndividuo());
		datosNotificacion.put("tribunales", egresosFacade
				.listarTribunalesCausa(new Expediente(commandE
						.getIdExpediente())));
		datosNotificacion.put("fiscalias", egresosFacade.listarFiscalias(new
		Expediente(commandE.getIdExpediente())));
		datosNotificacion.put("individuo", commandE.getIdIndividuo());
		datosNotificacion.put("expediente", commandE.getIdExpediente());
		datosNotificacion.put("motivoEgreso", commandE.getMotivoEgreso());

		return new ModelAndView("egreso/notificacionesfuga", "egreso",
				datosNotificacion);
	}

	protected Map referenceData(HttpServletRequest request, Object command,
			Errors errors, int page) throws Exception {

		Map<String, Object> map = new HashMap<String, Object>();
		switch (page) {
		case 0:
			if (request.getParameter("targets") == null)
				break;

			String idIndividuo = request.getParameter("targets");
			EgresoCommand commandE = (EgresoCommand) command;
			commandE.setIdIndividuo(idIndividuo);
			Individuo individuo = new Individuo(idIndividuo);
			commandE.setNombreIndividuo(egresosFacade
					.obtenerNombreCompletoIndividuo(individuo));

			// asociando el tipo de egreso que en este caso se corresponde con
			// fuga
			String motivoEgreso = request.getParameter("motivoegreso");
			commandE.setMotivoEgreso(motivoEgreso);
```

```
                // si el individuo está presuntamente fallecido se le ponen los
                // datos del presunto fallecimiento
                Boolean pF = egresosFacade
                            .verificarPresuntoFallecimientoIndividuo(individuo);

                if (pF) {
                Map datos =
          egresosFacade.obtenerDatosPresuntoFallecimientoIndividuo(individuo);
                      Date fecha = (Date) datos.get("fechaPresuntoFallecimiento");
                      Date hora = (Date) datos.get("horaPresuntoFallecimiento");
                      String descripcion = (String) datos
                                  .get("descripcionPresuntoFallecimiento");

                      commandE.setFecha((new SimpleDateFormat("dd/MM/yyyy")
                                  .format(fecha)));
                      commandE.setHora((new
                SimpleDateFormat("HH:mm:ss").format(hora)));
                      commandE.setDescripcion(descripcion);
                }

                commandE.setPresuntamenteMuerto(pF ? "1" : "0");
                Boolean tieneEstancia = egresosFacade
                            .verificarIndividuoTieneEstanciaAbierta(individuo);
                commandE.setTieneEstancia(tieneEstancia ? "1" : "0");
                Expediente expediente = egresosFacade
                            .buscarExpedienteActivoPorIndividuo(new Individuo(
                                        idIndividuo));
                if (tieneEstancia && expediente != null)
                            commandE.setIdExpediente(expediente.getId());

                SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
                Date fecha = globalFacade
                      .buscarFechaIngresoExpedienteActivoPorIndividuo(individuo);

                String fechaExpediente = null;
                if (fecha != null)
                      fechaExpediente = sdf.format(fecha);
                map.put("fechaExpediente", fechaExpediente);
                map.put("lugaresFuga", egresosFacade.listarLugarFugaEvasion());
                break;
          }
          return map;
     }
}
```

ANEXO 3. Diseño del dominio del módulo Decisiones

3.1 DCD: Dominio Decisiones



3.2 Descripción de las entidades de dominio significativas

Clase: Decision

Estereotipo(s) :	<<ORM Persistable>>
Descripción:	Entidad Orden contiene los datos generales de una decisión del sistema penitenciario
Tabla correspondiente:	Decision

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*

id	private	String	
tipoDecision	private	NomTipoDecision	
expediente	private	Expediente	
observacion	private	String	
estadoDecision	private	NomEstadoDecision	
documentosAsociados	private	List	
fechaEjecutada	private	Date	null

Clase: Documento

Estereotipo(s) :	<<ORM Persistable>>
Descripción	Entidad que refleja el concepto de un documento.Contiene los fecha de emisión, recepción y los remitentes del mismo
Tabla correspondiente:	Documento

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
expediente	private	Expediente	null
fecha	private	Date	null
id	private	String	null
numeroDocumento	private	String	null
tipoDocumento	private	NomTipoDocumento	null
otroRemitente	private	String	null
otroDestinatario	private	String	null
fechaEnvio	private	Date	null
emisoresReceptores	private	List	

Clase: EmisorReceptor

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Entidad padre de la jerarquía de los Emisores- Receptores de un documento
Tabla correspondiente:	EmisorReceptor
Abstracta :	true

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
id	private	String	
tipoEmisorReceptor	private	NomTipoEmisorReceptor	

Clase: EmisorReceptorCentro

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Emisor o receptor del documento, un centro penitenciario
Tabla correspondiente:	Emisor_Receptor_Centro

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
centro	private	Centro	

Clase: EmisorReceptorCoordinacionRegional

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Emisor o receptor del documento, la coordinación regional
Tabla correspondiente:	Emisor_Receptor_CR

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
coordinacionRegional	private	NomCoordinacionRegional	

Clase: EmisorReceptorOMP

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Emisor o receptor del documento, un órgano del Ministerio Público o Fiscalía
Tabla correspondiente:	EmisorReceptor_OMP

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
organoMP	private	NomOrganoMP	

Clase: EmisorReceptorOtro

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Emisor o receptor del documento, otro emisor o receptor no nomenclado como pueden ser alcaldías, óganos de seguridad ciudadana etcétera
Tabla correspondiente:	Emisor_Receptor_Otro

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
nombre	private	String	

Clase: EmisorReceptorTribunal

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Emisor o receptor del documento, un tribunal
Tabla correspondiente:	Emisor_Receptor_Tribunal

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
tribunal	private	NomTribunal	

Clase: DecisionCentro

Estereotipo(s) :	<<ORM Persistable>>
Documentación:	Decisión que implica la salida del penal actual a otro centro penitenciario
Tabla correspondiente:	Decision_Centro

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
centro	private	Centro	null

Clase: DecisionMedidaCautelar

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Decisión de medida cautelar.

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
medida	private	String	null

Clase: DecisionPais

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Entidad que representa una orden de repatriación o extradición. El tipo de orden depende del valor del atributo <code>Boolean extradicion</code>

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
fechaRE	private	Date	null
extradicion	private	Boolean	null
pais	private	NomPais	null

Clase: DecisionPermisoDirector

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Orden de permiso otorgado por el director.

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
motivo	private	String	null
direccion	private	String	null
fechaInicio	private	Date	null
fechaFin	private	Date	null
presuntaFechaRegreso	private	Date	null

Clase: DecisionPermisoExtraordinario

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Orden de permiso extraordinario de salida del centro

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
observaciones	private	String	null
fechaMinSal	private	Date	null
fechaMaxSal	private	Date	null
motivoPE	private	NomMotivoPE	null

Clase: DRegimenAbierto

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Órdenes que es necesario relacionar con un centro, ej: "Destacamento de Trabajo", "Otorgamiento de CTC".
Padre	DecisionCentro

Clase: DDestacamentoTrabajo

Estereotipos(s) :	<<ORM Persistable>>
Documentación	DDestacamentoTrabajo
Padre	DecisionCentro

Clase: DecisionMovimiento

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Orden de que implica movimiento de un individuo dentro del sistema penitenciario o como una salida a otro lugar
Abstract :	true

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
ejecutada	protected	Boolean	
planificable	protected	Boolean	

Clase: DecisionSalidaTransitoria

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Orden de salida transitoria del penal
Padre	DecisionMovimiento

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
destino	private	String	null
fechaSalidaTransitoria	private	Date	null
presuntaFechaRegreso	private	Date	null

observaciones	private	String	null
motivo	private	NomMotivoST	null

Clase: DecisionTraslado

Estereotipos(s) :	<<ORM Persistable>>
Documentación	Orden de traslado. @author Juan Carlos Gómez Correa
Padre	DecisionMovimiento

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
destino	private	Centro	null
fechaTraslado	private	Date	null
motivo	private	String	null
otroMotivo	private	String	null

Clase: EstadoLegal

Estereotipos(s) :	<<Nomenclador>>
Documentación	EstadoLegal. Define si un individuo está procesado(0), penado(1) o ambos(2).
Tabla correspondiente	Estado Legal

Atributos

Nombre	*Visibilidad*	*Tipo*	*Initial Value*
id	private	String	
nombre	private	String	
tiposSolicitud	private	NomTipoSolicitud	new ArrayList<NomTipoSolicitud>(0)
tiposDecision	private	NomTipoDecision	new ArrayList<NomTipoDecision>(0)

Clase: NomEstadoDecision

Estereotipos(s) :	<<Nomenclador>>
Documentación	Nomencla los estados que puede tener una decisión:pendiente, ejecutada, invalidada, ratificada etcétera
Tabla correspondiente	Nom_Estado_Decision

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
id	private	String	
nombreEstadoDecision	private	String	

Clase: NomTipoDecision

Estereotipos(s) :	<<Nomenclador>>
Documentación	Nomencla los tipos de decisión que se pueden registrar en el sistema .
Tabla correspondiente	Nom_Tipo_Decision

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
id	private	String	
proceso	private	Boolean	
cierraProceso	private	Boolean	
ingreso	private	Boolean	
computo	private	Boolean	
pena	private	Boolean	
denegacion	private	Boolean	
nombreTipoDecision	private	String	
abreviatura	private	String	
tiposFases	private	List	

estadoLegal	private	EstadoLegal	

Clase: NomTipoEmisorReceptor

Estereotipos(s) :	<<Nomenclador>>
Documentación	Nomencla los tipos de emisor- receptor de los documentos: tribunal, centro , coordinación regional etc .
Tabla correspondiente	Nom_Tipo_Emisor_Receptor

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
id	private	String	
nombreEmisorReceptor	private	String	
clasificacion	private	String	
nomTiposDocumento	private	List	
nomTiposSolicitud	private	List	

Clase: TipoDecisionAsociadaCentro

Estereotipos(s) :	<<Nomenclador>>
Documentación	Nomencla las decisiones que se pueden registrar por cada tipo de centro, y en el caso de que puedan implicar egreso las condiciones a verificar y si en ese centro esa decisión implica cierre del expediente penitenciario.
Tabla correspondiente	Tipo_Decision_Asoc_Centro

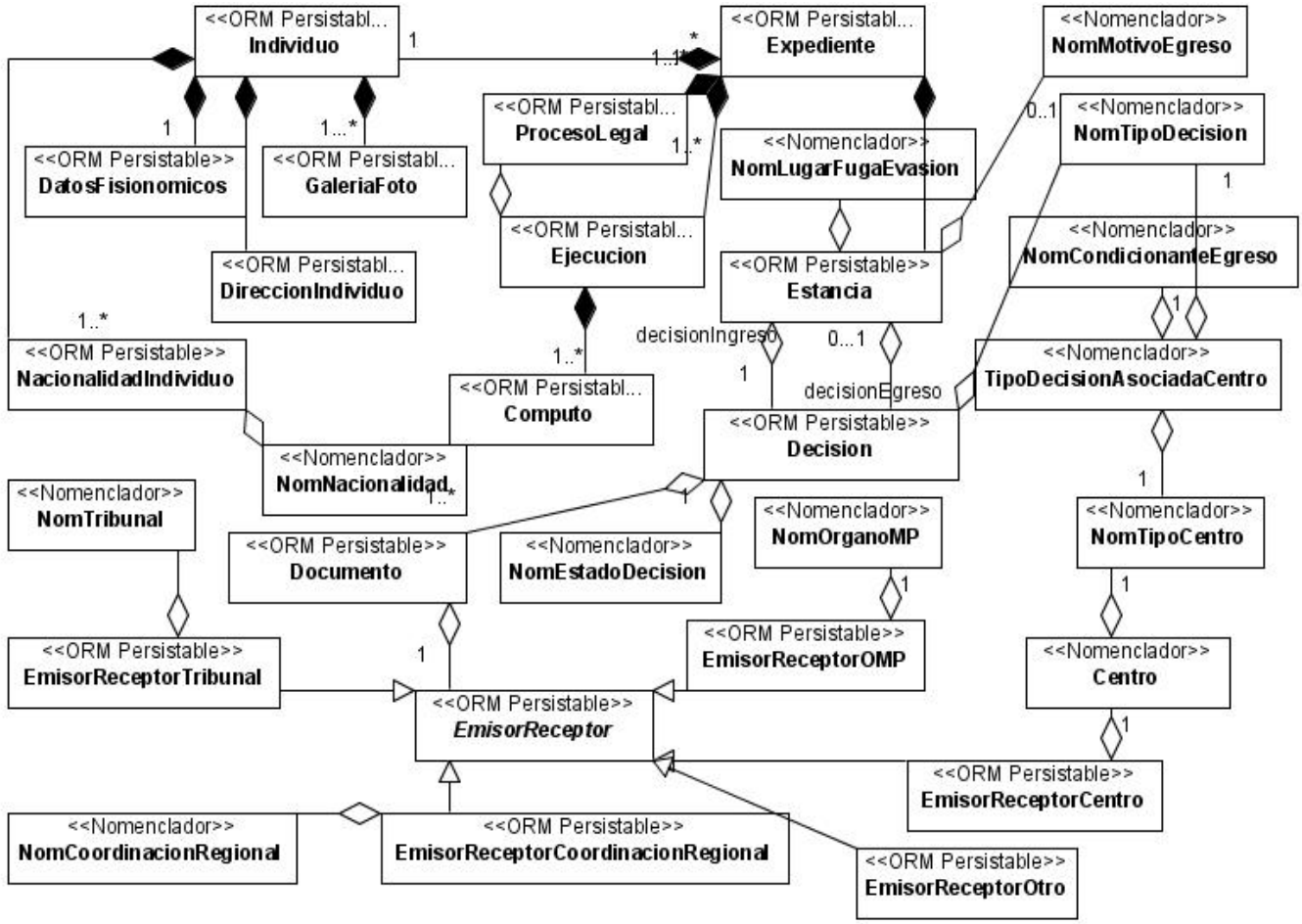
Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor Inicial*
tipoDecision	private	NomTipoDecision	null
tipoCentro	private	NomTipoCentro	null
id	private	TipoOrdenAsociadaCentroId	null
condicionanteEgreso	private	NomCondicionanteEgreso	

cerrarExpediente	private	Boolean	false
egreso	private	Boolean	

ANEXO 4. Diseño del dominio del módulo Egresos

4.1 DCD: Dominio Egreso



4.2 Descripción de las entidades más significativas

Clase: Expediente

Estereotipo(s) :	<<ORM Persistable>>
Documentación	Entidad de dominio que define los datos de un expediente de un individuo en el sistema penitenciario
Tabla correspondiente:	Expediente

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
id	private	String	
individuo	private	Individuo	
activo	private	Boolean	true
fechaDetencion	private	Date	
fechaEgresoSistema	private	Date	
fechaIngresoSistema	private	Date	
decisiones	private	List	
solicitudes	private	List	

Clase: Individuo

Estereotipo(s) :	<<ORM Persistable>>
Documentación	Entidad de dominio que define los datos de un individuo. Es una entidad única en el sistema.
Tabla correspondiente:	Individuo

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
fallecido	private	Boolean	false
presuntoFallecido	private	Boolean	false
bloqueado	private	Boolean	true
alias	private	String	
apellido1	private	String	
apellido2	private	String	
apodo	private	String	
banda	private	String	
cumplioSM	private	Boolean	
estadoCivil	private	NomEstadoCivil	
fechaNacimiento	private	Date	
gradoInstruccion	private	NomGradoInstruccion	

id	private	String	
indiceDelictual	private	NomIndiceDelictual	
indocumentado	private	Boolean	
nombre1	private	String	
nombre2	private	String	
numeroHijas	private	Integer	
numeroHijos	private	Integer	
numeroIdentidad	private	String	
paisNacimiento	private	NomPais	
paisResidencia	private	NomPais	
religion	private	NomReligion	
sexo	private	Boolean	
tipoDocumentoIdentificacion	private	NomTipoDocumentoIdentificacion	
vinculos	private	List	
tipoMuerte	private	String	

Clase: Estancia

Estereotipo(s) :	<<ORM Persistable>>
Documentación	Entidad que refleja el concepto de la estancia de un individuo en un centro penitenciario, de acuerdo a un expediente penitenciario.
Tabla correspondiente:	Estancia

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
centro	private	Centro	
decisionEgreso	private	Decision	
decisionIngreso	private	Decision	
horaIngreso	private	Date	
horaEgreso	private	Date	
expediente	private	Expediente	
fechaEgreso	private	Date	
fechaIngreso	private	Date	
id	private	String	
motivoEgreso	private	NomMotivoEgreso	

observacion	private	String	
lugarEvasionFuga	private	NomLugarFugaEvasion	

Clase: NomCondicionanteEgreso

Estereotipo(s) :	<<Nomenclador>>
Documentación	Nomencla los condicionantes para que en un centro se pueda o no efectuar un egreso
Tabla correspondiente:	NomCondicionanteEgreso

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
id	private	String	
nombre	private	String	

Clase: NomMotivoEgreso

Estereotipo(s) :	<<Nomenclador>>
Documentación	Nomencla los motivos de egreso de un individuo: fuga, evasión, decisión judicial etcétera
Tabla correspondiente:	NomMotivoEgreso

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
id	private	String	
nombreMotivoEgreso	private	String	
abreviatura	private	String	

Clase: NomLugarFugaEvasion

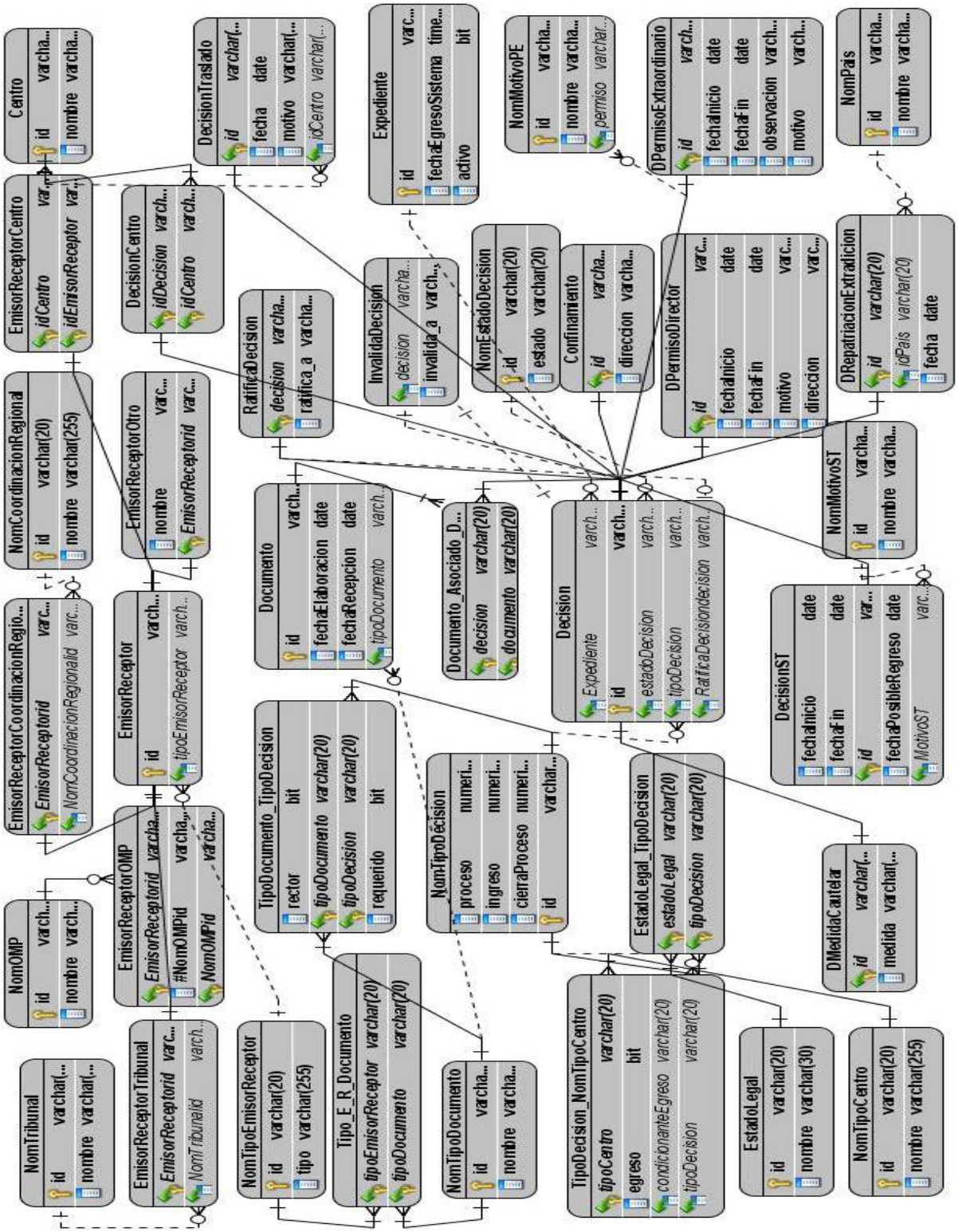
Estereotipo(s) :	<<Nomenclador>>
Documentación	Nomenclador que tiene los valores de los lugares de los cuales se puede fugar o evadir un individuo.
Tabla correspondiente:	Nom_Motivo_Egreso

Atributos

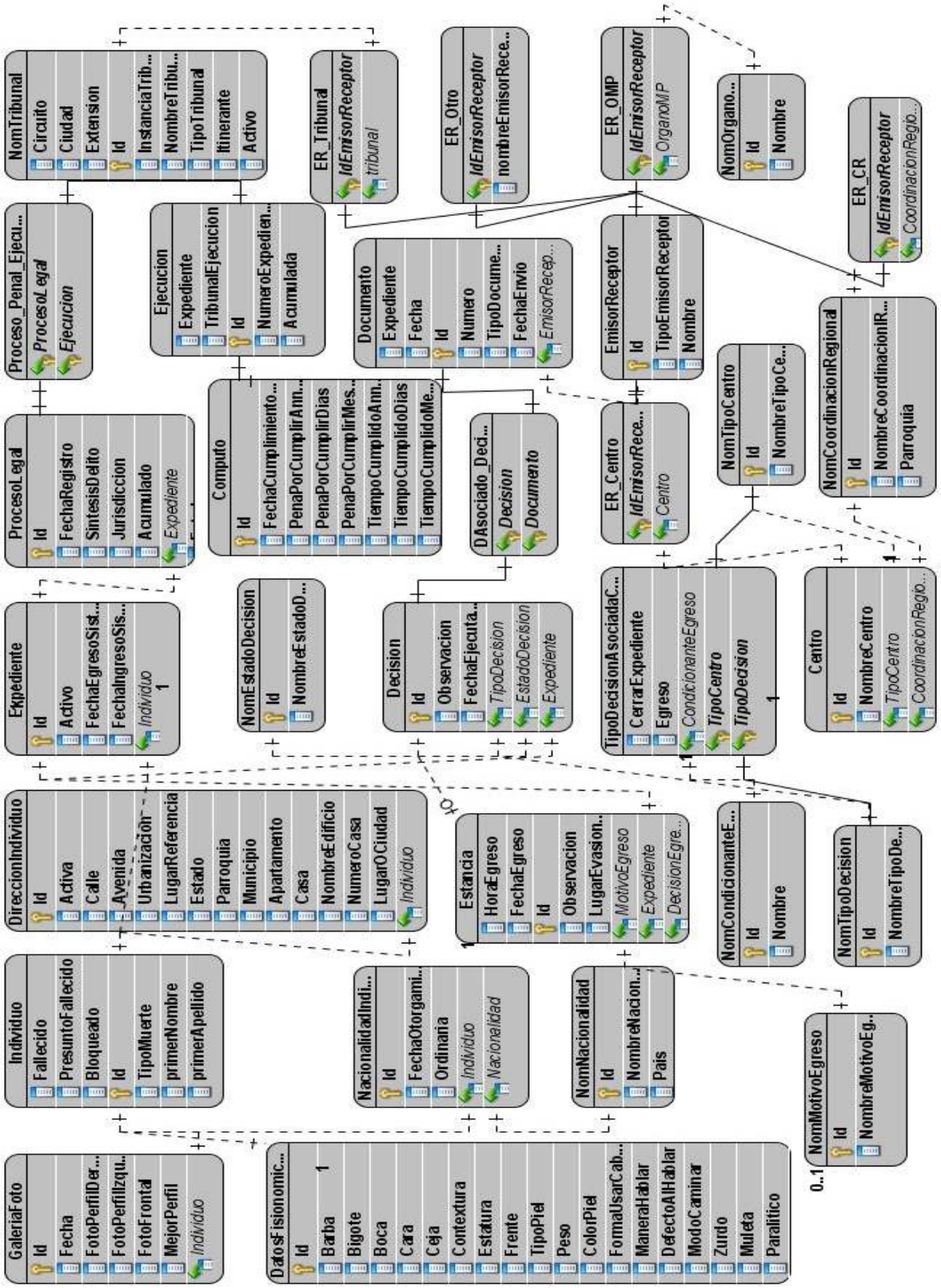
Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
id	private	String	
nombre	private	String	

ANEXO 5 Diseño del Modelo de Datos de los módulos Decisiones y Egresos

5.1 Modelo de Datos Decisiones

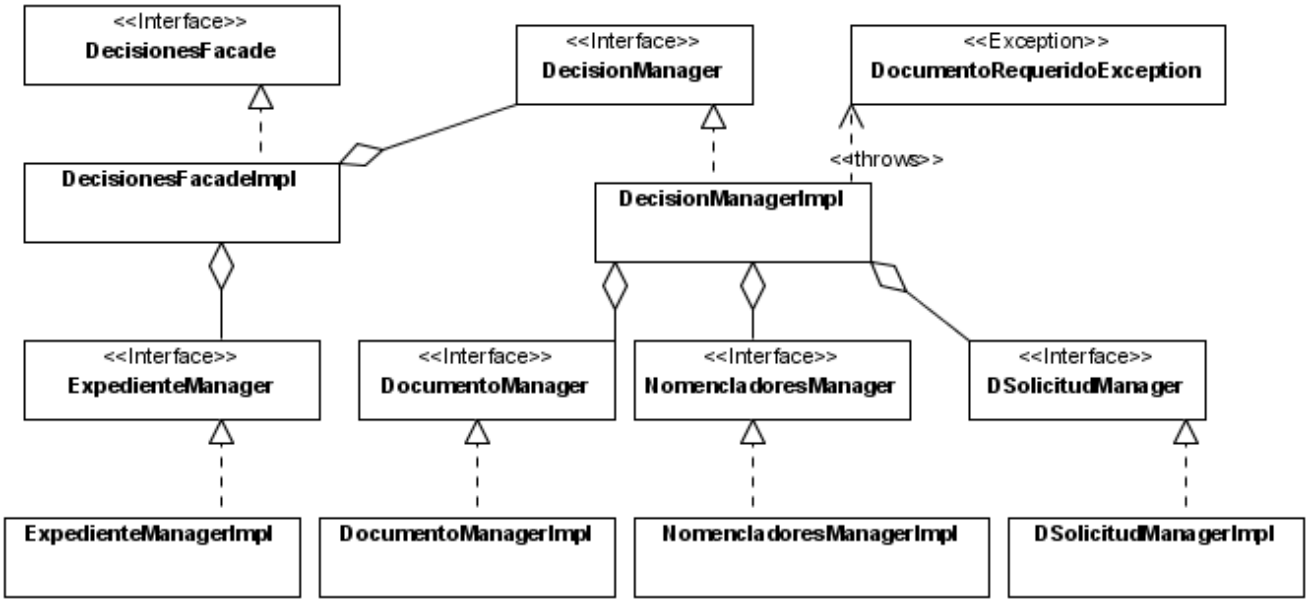


5.2 Modelo de Datos Egresos.



ANEXO 6. Diseño de la capa de Negocio de los Módulos Decisiones y Egresos

6.1 DCD: Capa de Negocio Decisiones



Excepción: DocumentoRequeridoException

Estereotipo :	<<Exception>>
Documentación:	Excepción que se lanza cuando al registrar la decision no se registrado un documento que es considerado requerido para la misma

Evento: RegistroDecisionEvent

Estereotipo :	<<Event>>
Documentación:	Evento que se publica cuando se registra o modifica una decisión , es de vital importancia para los módulos Traslados y Salidas Transitorias porque implican una nueva planificación o modificación de una respectivamente, para el caso de la decisiones de tipo Traslado Interpenal , Transferencia o Salida Transitoria.

Atributos

Nombre	*Visibilidad*	*Tipo*	*Valor inicial*
idDecision	private	String	null
idExpediente	private	String	null
idTipoDecision	private	String	null
idEstadoDecision	private	String	null

fechaRegistro	private	Long	null
eliminada	private	Boolean	false

Clase: DecisionManager

Estereotipo :	<<Interface>>
Documentación:	Interfaz del manager que gestiona los procesos de negocio relacionados con las decisiones (órdenes).

Operaciones

Nombre	*Parámetros*	*Visibilidad*	*Tipo retorno*
registrarDecision	expediente : Expediente decision : Decision documentosAsociadosNuevos : List documentosAsociadosEditados : List documentosAsociadosEliminados : List idsSolicitudesAsociadas : List idsSolicitudesDesasociadas : List	public	void
listarDecisionesPorExpediente	expediente : Expediente	public	List
obtenerIdsDecisiones		public	Map
listarDocumentosAsociados	decision : Decision	public	List
obtenerRangoDecisionesPorExpediente	expediente : Expediente inicio : Integer cantidadMaxima : Integer idsExcluidos : List	public	List
cantidadDecisionesPorExpediente	expediente : Expediente	public	Integer
cantidadDocumentosAsociados	decision : Decision	public	Integer
obtenerRangoDocumentosAsociados	decision : Decision inicio : Integer cantidadMaxima : Integer idsExcluidos : List	public	List
eliminarDecision	decision : Decision	public	void
obtenerDecision	decision : Decision	public	Decision
validarDocumentosRequeridos	decision : Decision documentosNuevos : List documentosEditados : List documentosEliminados : List	public	Boolean

buscarDecision	idDecision : String	public	Decision
actualizarDecision	decision : Decision	public	void
buscarDocumentoPorDecisionYTipoDocumento	decision : Decision tipoDocumento : NomTipoDocumento	public	Documento
obtenerDocumentoRector	decision : Decision	public	Documento
setEstadoDecision	decision : Decision estado : EstadoDecisionEnum	public	void
ejecutarDecision	decision : Decision fechaEjecucion : Date	public	void
getEstadoDecision	estado : EstadoDecisionEnum	public	NomEstadoDecision
buscarDecisionEgresoPorExpedienteEstadoYCentro	expediente : Expediente estadoDecision : NomEstadoDecision tipoCentro : NomTipoCentro	public	List
buscarDecisionPorEgresoTipoCentroFecha	egreso : Boolean tipoCentro : NomTipoCentro fecha : Date estanciaCerrada : Boolean estadoDecision : NomEstadoDecision	public	List
registrarDecision	individuos : List decision : Decision documentosAsociados : List idsSolicitudesAsociadas : List	public	void
registrarDecisionSinRestricciones	expediente : Expediente decision : Decision documentosNuevos : List documentosEditados : List documentosEliminados : List	public	void
validarInvalidarDecision	decision : Decision validar : Boolean	public	void

Clase: DecisionManagerImpl

Estereotipo :	Implementación
Documentación:	Implementación del Manager que gestiona los procesos asociados a las decisiones
Interfaz	DecisionManager

Clase: DocumentoManager

Estereotipos(s) :	<<Interface>>
Documentación:	Interfaz del manager que gestiona los procesos de negocio relacionados con la entidad Documento.

Operaciones

Nombre	*Parámetros*	*Visibilidad*	*Tipo retorno*
actualizarDocumento	documento : Documento	public	Documento
actualizarDocumentosPorComputo	computo : Computo documentosAdicionados : List documentosModificados : List documentosEliminados : List	public	void
actualizarDocumentosPorDecision	decision : Decision documentosAdicionados : List documentosModificados : List documentosEliminados : List	public	void
actualizarDocumentosPorPena	pena : Pena documentosAdicionados : List documentosModificados : List documentosEliminados : List	public	void
actualizarTipoFechasDocumentos	estancia : Estancia documentos : List idsEliminados : List	public	void
adicionarDocumento	documento : Documento	public	void
buscarDocumentoMasReciente	documentos : List	public	Documento
buscarDocumentoPorDecisionYTipoDocumento	decisionJudicial : tipoDocumento : NomTipoDocumento	public	Documento
buscarDocumentoPorEstanciaYTipoDocumento	estancia : Estancia tipoDocumento : NomTipoDocumento	public	List
buscarDocumentoPorId	documento : Documento	public	Documento
cambiarFechaDocumento	documento : Documento fecha : Date	public	void
cantidadDocumentosPorComputo	computo : Computo	public	Integer
cantidadDocumentosPorDecisionJudicial	decisionJudicial :	public	Integer

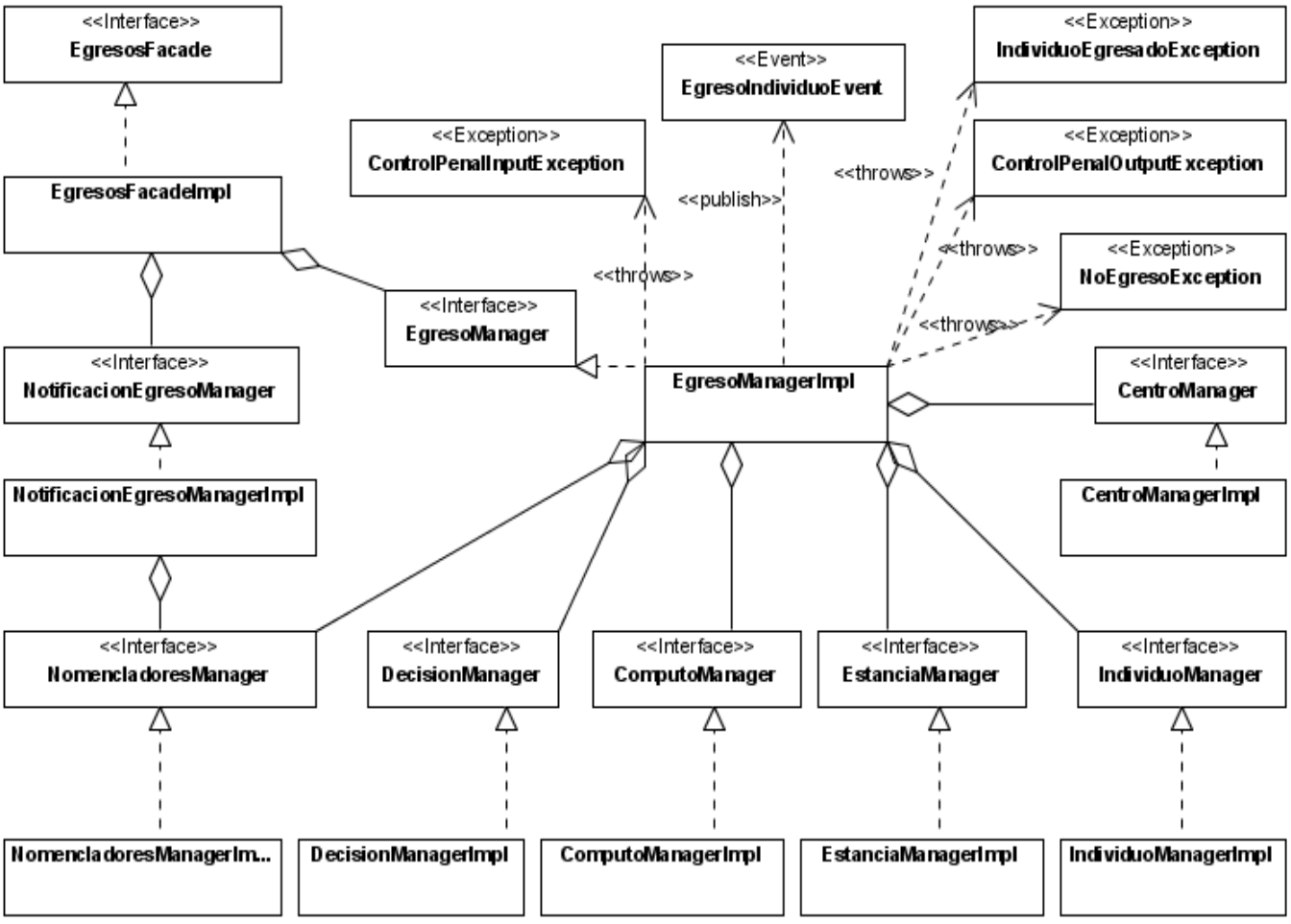
cantidadDocumentosPorPena	pena : Pena	public	Integer
eliminarDocumento	documento : Documento	public	void
generarNumeroDocumento	o : Object	public	String
modificarFechaEnvio	documentos : List	public	void
obtenerDocumentosPorComputo	computo : Computo comienzo : Integer cantidad : Integer excluir : List	public	List
obtenerDocumentosPorDecisionJudicial	decisionJudicial : comienzo : Integer cantidad : Integer excluir : List	public	List
obtenerDocumentosPorEstancia	estancia : Estancia comienzo : Integer cantidad : Integer	public	List
obtenerDocumentosIngresoPorEstancia	estancia : Estancia comienzo : Integer cantidad : Integer	public	List
obtenerDocumentosEgresoPorEstancia	estancia : Estancia comienzo : Integer cantidad : Integer	public	List
obtenerDocumentosPorPena	pena : Pena comienzo : Integer cantidad : Integer excluir : List	public	List
obtenerEmisor	documento : Documento	public	EmisorReceptor
obtenerReceptor	documento : Documento	public	EmisorReceptor
obtenerReceptores	documento : Documento	public	List
obtenerCantidadDocumentosIngresoPorEstancia	estancia : Estancia	public	Integer
obtenerCantidadDocumentosEgresoPorEstancia	estancia : Estancia	public	Integer
obtenerCantidadDocumentosPorEstancia	estancia : Estancia	public	Integer

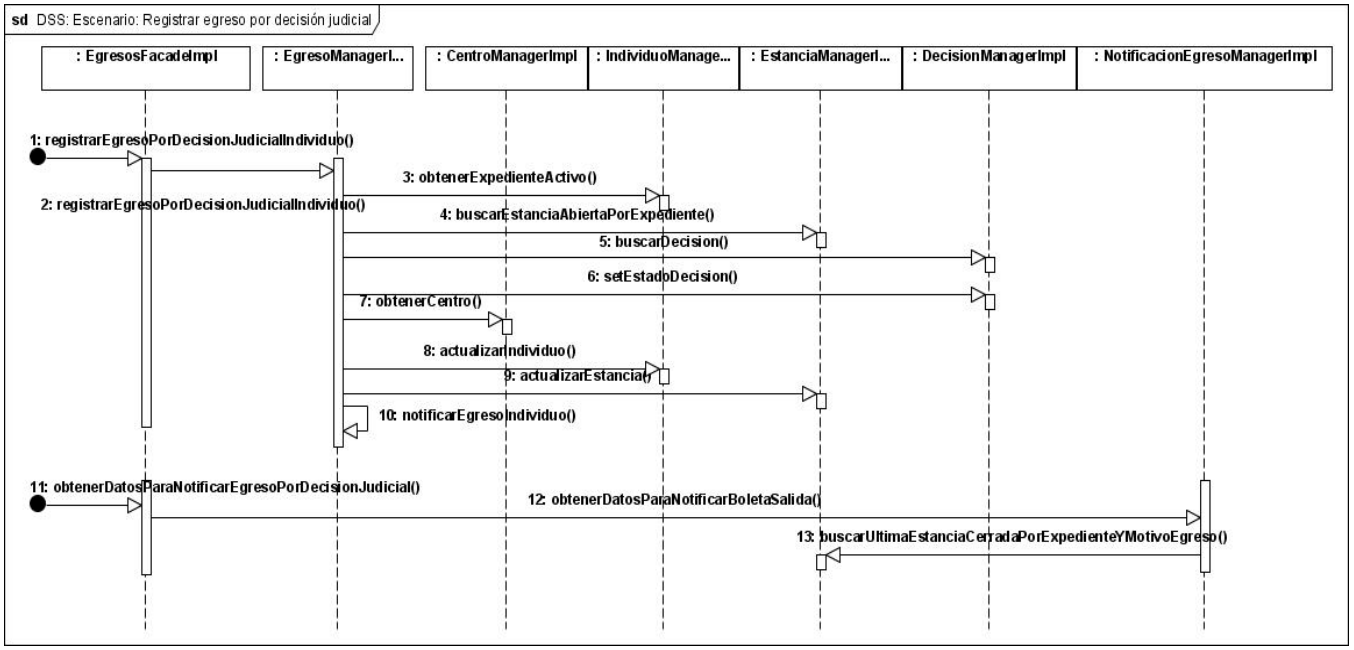
obtenerDocumentoGE	estancia : Estancia egreso : Boolean start : Integer len : Integer filter : List	public	List
obtenerCantidadDocumentosGEstancias	estancia : Estancia egreso : Boolean	public	Integer

Clase: DocumentoManagerImpl

Estereotipo :	Implementación
Documentación:	Implementación del manager que gestiona los procesos de negocio relacionados con la entidad Documento.
Interfaz	DocumentoManager

6.2 Capa de negocio Egresos





Evento: EgresoIndividuoEvent

Estereotipo :	<<Event>>
Documentación:	Evento que se publica cuando un individuo es egresado del sistema penitenciario. Contiene los datos más significativos de la realización del egreso y es de interés para todos los módulos que se relacionen con el individuo en el sistema penitenciario.

Atributos

Nombre	*Visibilidad*	*Tipo*
idIndividuo	private	String
fechaEgreso	private	Date
horaEgreso	private	Date
nomMotivoEgreso	private	NomMotivoEgreso
idEstanciaCerrada	private	String
idExpedienteCerrado	private	String
idDecision	private	String

Excepción: IndividuoEgresadoException

Estereotipo :	<<Exception>>
Documentación:	Excepcion que se lanza cuando el individuo ya se encuentra egresado del sistema y se está intentando la realización del egreso nuevamente o porque simplemente no tiene una estancia abierta o un expediente activo.

Excepción: NoEgresoException

Estereotipo :	<<Exception>>
Documentación:	Excepción que expresa la imposibilidad de realizar un egreso. Se lanza cuando hay un motivo para abortar el egreso y este motivo se le notifica oportunamente al usuario.

Clase: EgresosFacade

Estereotipo :	<<Interface>>
Documentación:	Fachada del módulo Egresos del Subsistema Control Penal del SIGEP. Contiene todas las funcionalidades que brinda el módulo. @author Yadira Benavides

Operaciones

Nombre	*Parámetros*	*Visibilidad*	*Tipo retorno*
buscarDecisionesTributariasEgresoPorIndividuo	individuo : .Individuo	public	List
buscarExpedienteActivoPorIndividuo	individuo : Individuo	public	Expediente
buscarProcesosLegalesPendientesPorIndividuo	individuo :Individuo	public	List
cantidadIndividuosNoPuedenEgresarPorFecha	fecha : Date	public	Integer
cantidadIndividuosPuedenEgresarPorFecha	fecha : Date	public	Integer
listarEmbajadasIndividuo	individuo :Individuo	public	List
listarFiscalias	expediente :.Expediente	public	List
listarIndividuoNoPuedenEgresar	fecha : Date inicio : Integer cantidad : Integer	public	List

listarIndividuosEgresadosPorFecha	fecha : Date inicio : Integer cantidad : Integer ascendente : Boolean	public	List
listarIndividuosPuedenEgresarPorFecha	fecha : Date inicio : Integer cantidad : Integer	public	List
listarTiposDocumentosRegistranDefuncion		public	List
listarTribunalesCausa	expediente :.Expediente	public	List
obtenerComputosValidosPorIndividuo	individuo :.Individuo fecha : Date	public	List
obtenerDatosParaNotificarDefuncionIndividuo	individuo :.Individuo expediente :.Expediente	public	DatosNotificacionDefuncion
obtenerDatosParaNotificarEgresoPorDecisionJudicial	individuo :.Individuo expediente :.Expediente	public	DatosNotificacionBoletaSalida
obtenerDatosParaNotificarEvasionOFuga	individuo :Individuo expediente :.Expediente evasion : Boolean	public	DatosNotificacionEvasion
obtenerDatosNotificarNoEgreso	individuo :.Individuo expediente :.Expediente	public	DatosNotificacionNoEgreso
obtenerDatosPresuntoFallecimientoIndividuo	individuo :Individuo	public	Map
obtenerEmisoresReceptoresPorTipoDocumento	tipoDocumento : NomTipoDocumento	public	List
obtenerNombreCompletoIndividuo	individuo :.Individuo	public	String
obtenerUltimoComputoValidoPorIndividuo	individuo :.Individuo fecha : Date	public	MostrarComputo
registrarEgresoPorDecisionJudicialIndividuo	datosEgresoDecisionJudicial DatosEgresoDecisionJudicial	public	void
registrarEgresoPorDefuncionIndividuo	datosEgresoDefuncion : DatosEgresoDefuncion	public	void
registrarEgresoPorEvasionIndividuo	datosEgresoEvasionFuga : DatosEgresoEvasionFuga	public	void
registrarEgresoPorFugaIndividuo	datosEgresoEvasionFuga : DatosEgresoEvasionFuga	public	void

registrarPresuntaDefuncionIndividuo	individuo :.Individuo fechaPresuntaDefuncion :Date horaPresuntaDefuncion :Date descripcionHecho : String	public	void
verificarIndividuoTieneEstanciaAbierta	individuo :Individuo	public	Boolean
verificarPresuntoFallecimientoIndividuo	individuo :Individuo	public	Boolean
buscarCondicionanteEgresoPorDecision	decision :.Decision	public	NomCondicionanteEgreso
buscarEstanciaAbiertaPorExpediente	expediente :.Expediente	public	Estancia
verificarTieneNacionalidadExtranjera	individuo :Individuo	public	Boolean
buscarTiposMuerte		public	List
listarLugarFugaEvasion		public	List

Clase: EgresosFacadeImpl

Estereotipo :	Implementación
Documentación:	Implementación de la fachada del módulo Egresos del Subsistema Control Penal del SIGEP
Interfaz	EgresosFacade

Clase: EgresoManager

Estereotipos :	<<Interface>>
Documentación:	Manager que refleja el negocio asociado a un egreso en el sistema penitenciario @author Yadira

Operaciones

Nombre	*Parámetros*	*Visibilidad*	*Tipo retorno*
buscarDecisionesTributariasEgresoPorIndividuo	individuo : Individuo	public	List
buscarProcesosLegalesPendientesPorIndividuo	individuo : Individuo	public	List

registrarPresuntaDefuncionIndividuo	individuo : Individuo fechaPresuntaDefuncion : Date horaPresuntaDefuncion : Date descripcionHecho : String	public	void
registrarEgresoPorDecisionJudicialIndividuo	datosEgresoDecisionJudicial : DatosEgresoDecisionJudicial	public	void
registrarEgresoPorDefuncionIndividuo	datosEgresoDefuncion : DatosEgresoDefuncion	public	void
registrarEgresoPorEvasionIndividuo	datosEgresoEvasionFuga : DatosEgresoEvasionFuga	public	void
registrarEgresoPorFugaIndividuo	datosEgresoEvasionFuga : DatosEgresoEvasionFuga	public	void
buscarDecisionesDeEgresoRegistradasPorExpediente	expediente : Expediente	public	List
listarTiposDocumentosRegistranDefuncion		public	List
obtenerDatosPresuntoFallecimientoIndividuo	individuo : Individuo	public	Map
listarIndividuosPuedenEgresarPorFecha	fecha : Date inicio : Integer cantidad : Integer	public	List
listarIndividuosEgresadosPorFecha	fecha : Date inicio : Integer cantidad : Integer ascendente : Boolean	public	List
listarIndividuoNoPuedenEgresar	fecha : Date inicio : Integer cantidad : Integer	public	List
obtenerComputosValidosPorIndividuo	individuo : Individuo fecha : Date	public	List
obtenerUltimoComputoValidoPorIndividuo	individuo : Individuo fecha : Date	public	MostrarComputo
cantidadIndividuosPuedenEgresarPorFecha	fecha : Date	public	Integer
cantidadIndividuosNoPuedenEgresarPorFecha	fecha : Date	public	Integer
buscarCondicionanteEgresoPorDecision	decision : Decision	public	NomCondicionanteEgreso

obtenerMotivosEgresoPorTipoCentro	tipoCentro : NomTipoCentro	public	List
notificarEgresoIndividuo	idIndividuo : String	public	void

Clase: EgresoManagerImpl

Estereotipo :	Implementación
Documentación:	Implementación de la interfaz de negocio de Egresos
Interfaz	EgresoManager

Clase: NotificacionEgresoManager

Estereotipo :	<<Interface>>
Documentación	Manager que tiene toda la lógica de los datos para notificar un egreso @author Yadira

Operaciones

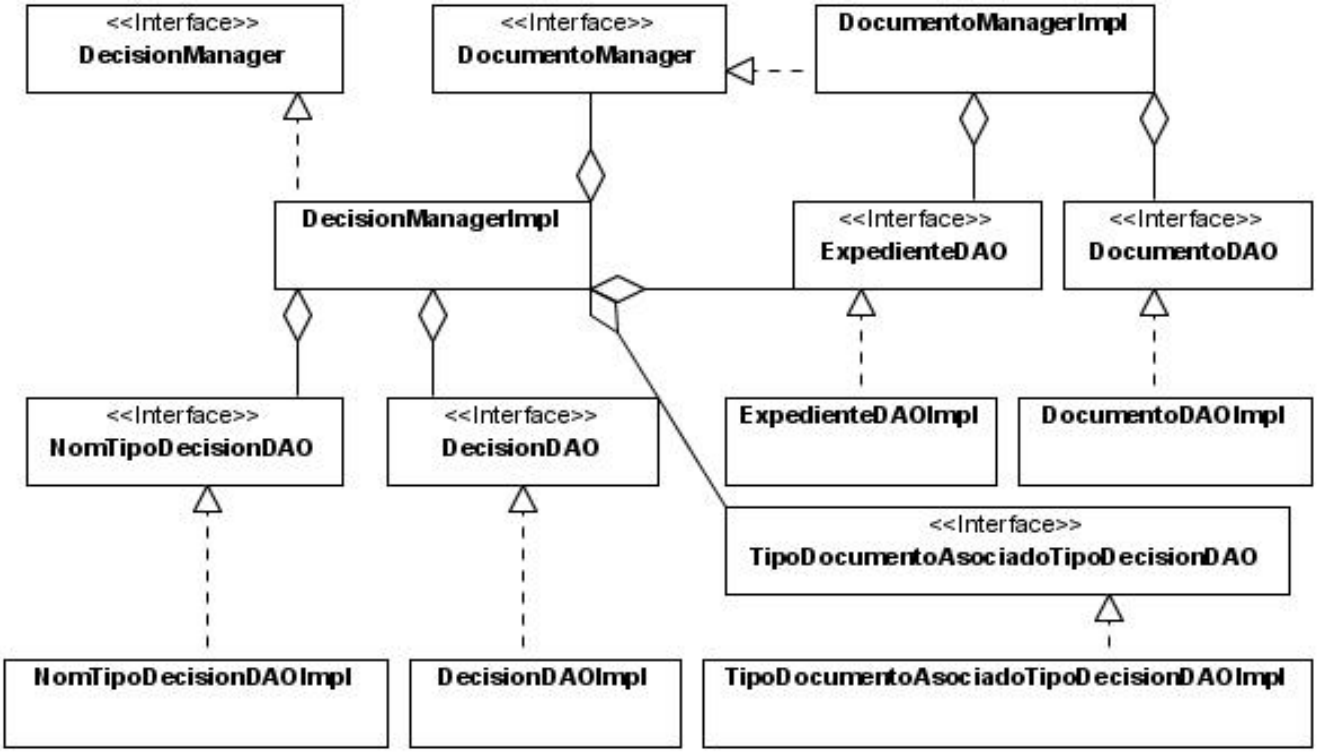
Nombre	*Parámetros*	*Visibilidad*	*Tipo retorno*
obtenerDatosParaNotificarBoletaSalida	individuo : Individuo expediente : Expediente	public	DatosNotificacionBoletaSalida
obtenerDatosParaNotificarDefuncionIndividuo	individuo : Individuo expediente : Expediente	public	DatosNotificacionDefuncion
obtenerDatosParaNotificarEvasionOFuga	individuo : Individuo expediente : Expediente evasion : Boolean	public	DatosNotificacionEvasion
obtenerDatosNotificarNoEgreso	individuo : Individuo expediente : Expediente	public	DatosNotificacionNoEgreso

Clase: NotificacionEgresoManagerImpl

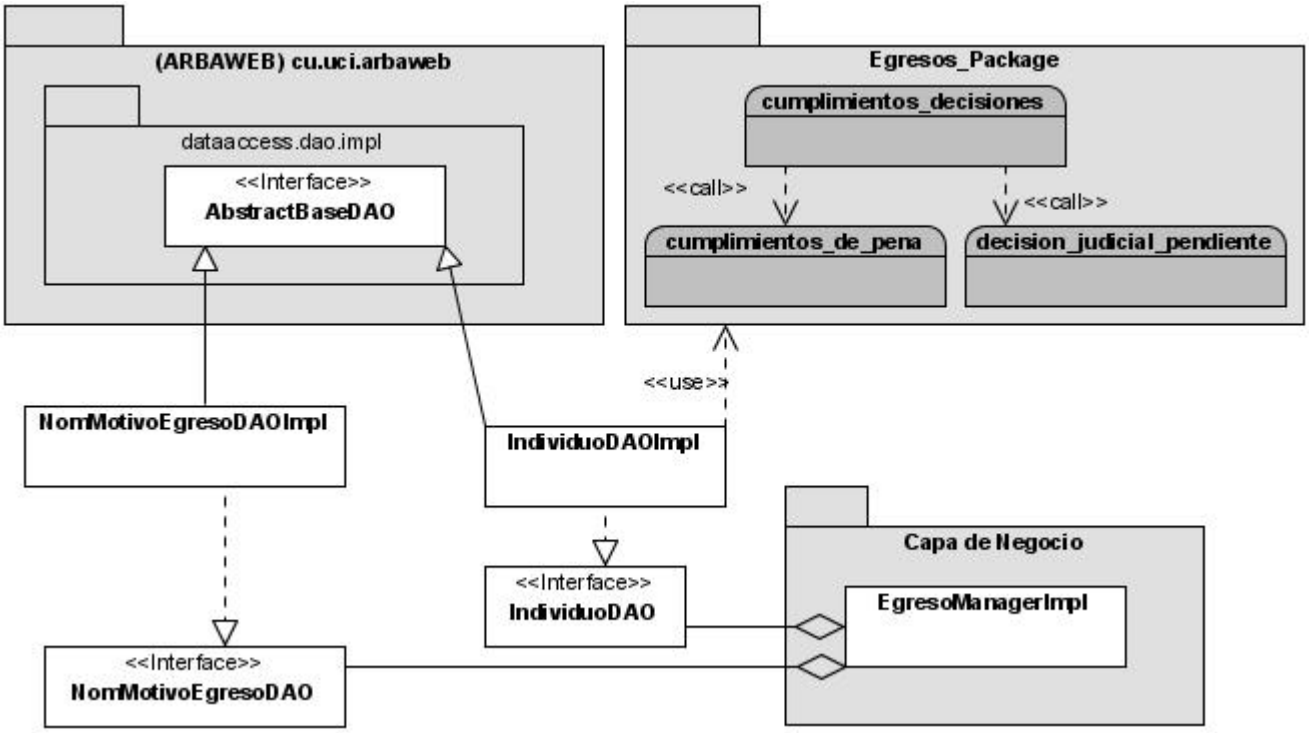
Estereotipo :	Implementación
Documentación:	Implementación de la interfaz que desarrolla toda la lógica de las notificaciones de egreso.
Interfaz	NotificacionEgresoManager

ANEXO 7. Diseño de la Capa de Acceso a Datos de los módulos Decisiones y Egresos

7.1 DCD: Capa de Acceso a Datos Decisiones



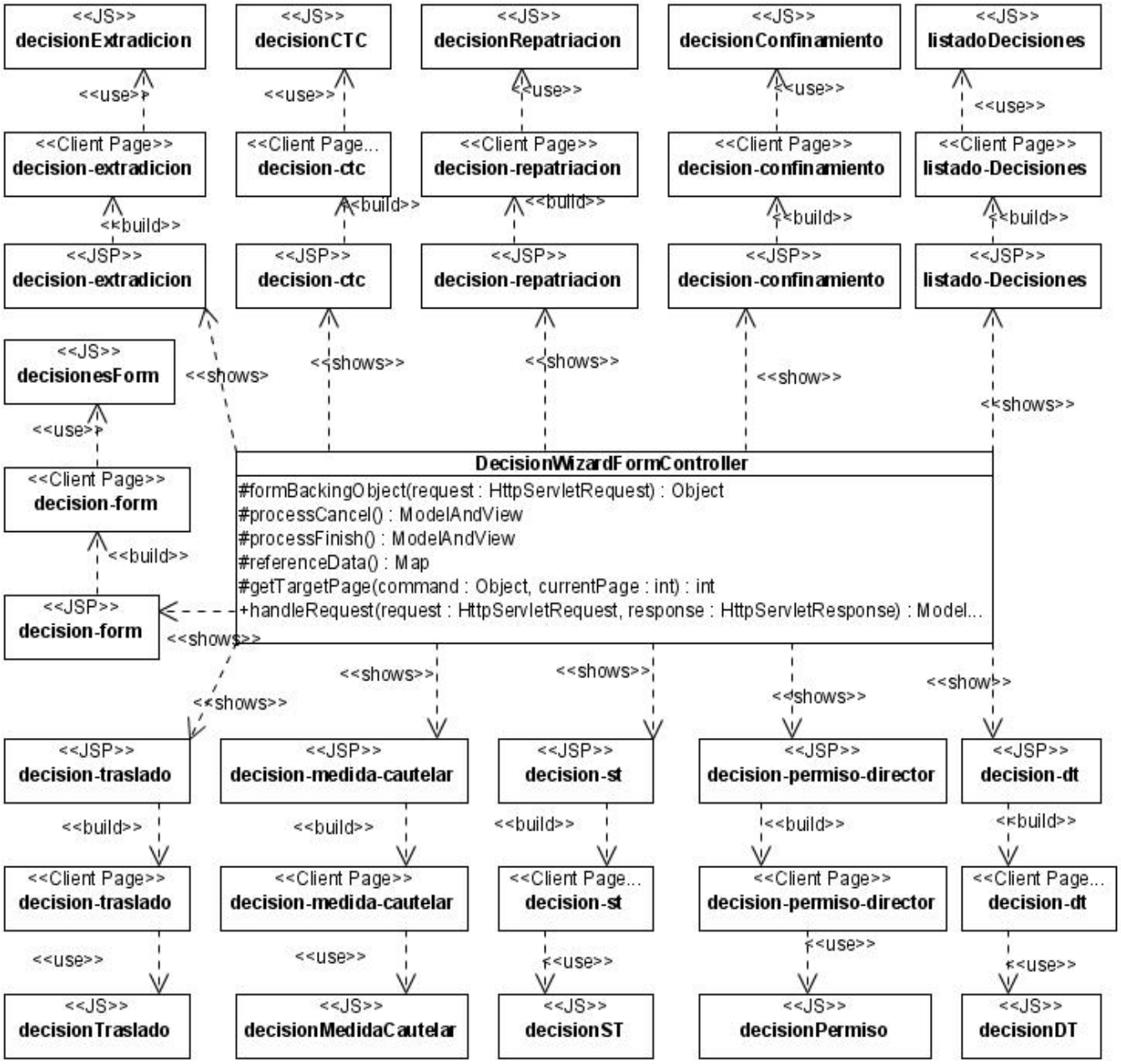
7.2 DCD: Capa de Acceso a Datos Egresos



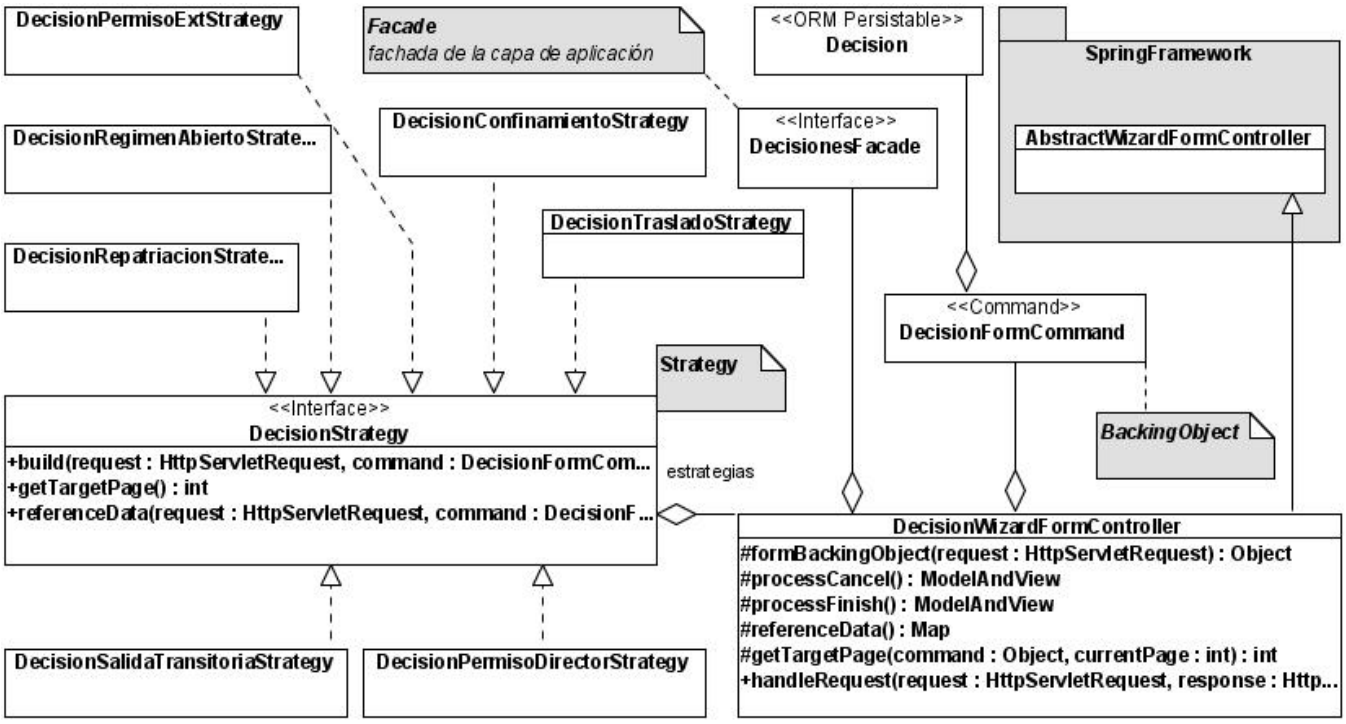
ANEXO 8. Diseño de la Capa de Presentación de los módulos Decisiones y Egresos

8.1 Capa de Presentación del módulo Decisiones

8.1.1 DCD: Decisiones

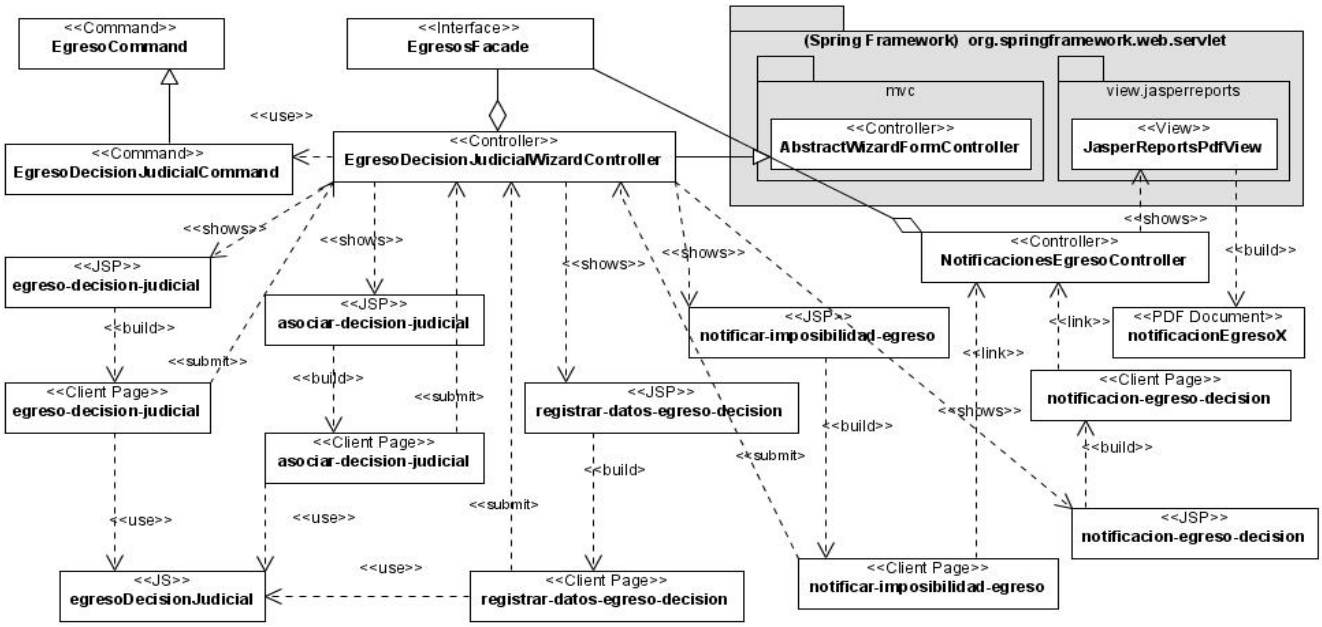


8.1.2 DCD: Utilización de los patrones Abstract Factory, Strategy, Backing Object y Facade

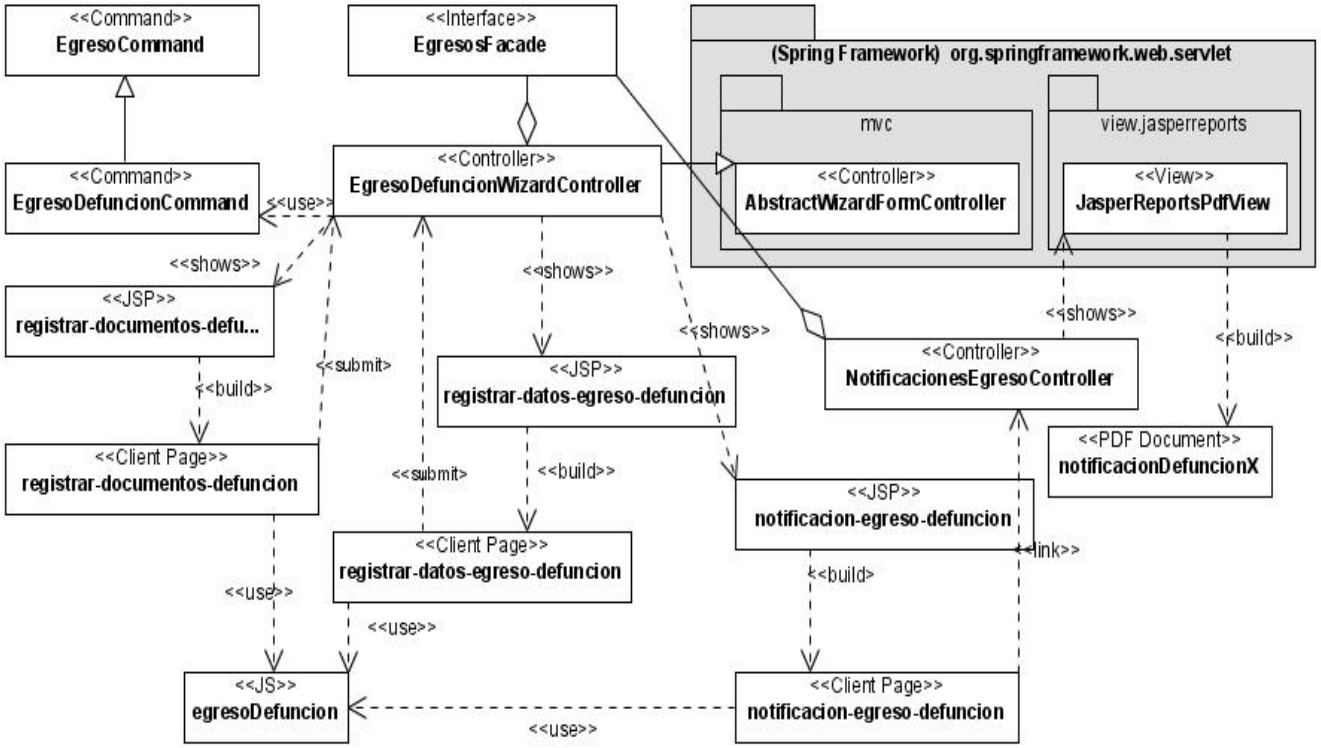


8.2 Capa de Presentación del módulo Egresos

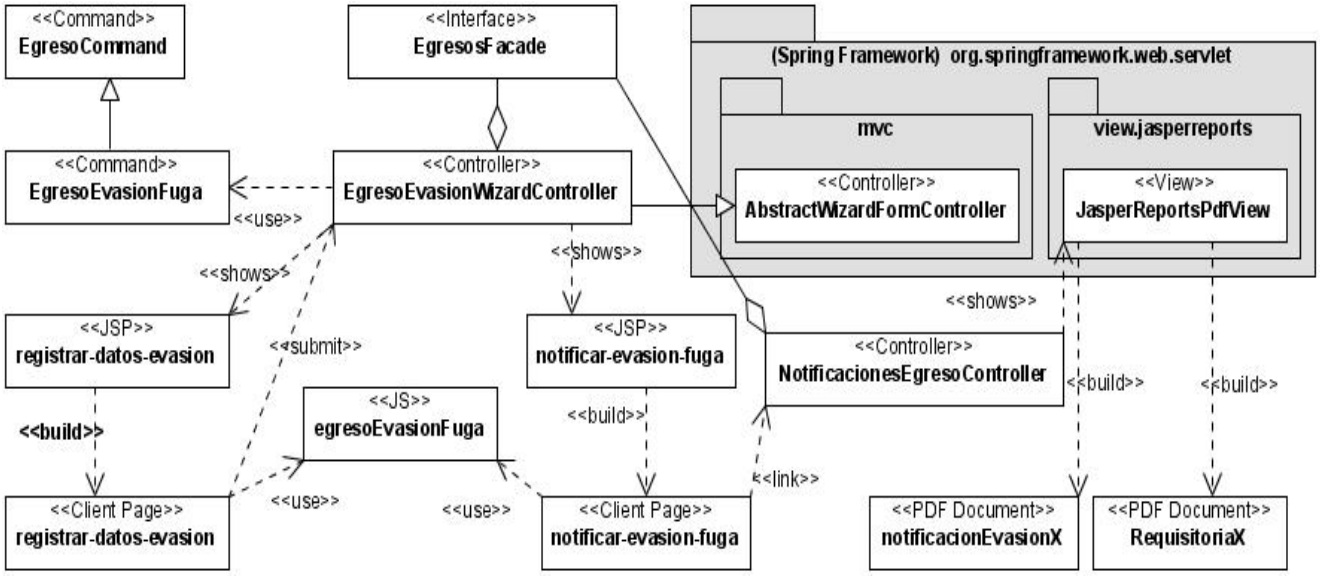
8.2.1 DCD: Egreso por Decisión Judicial



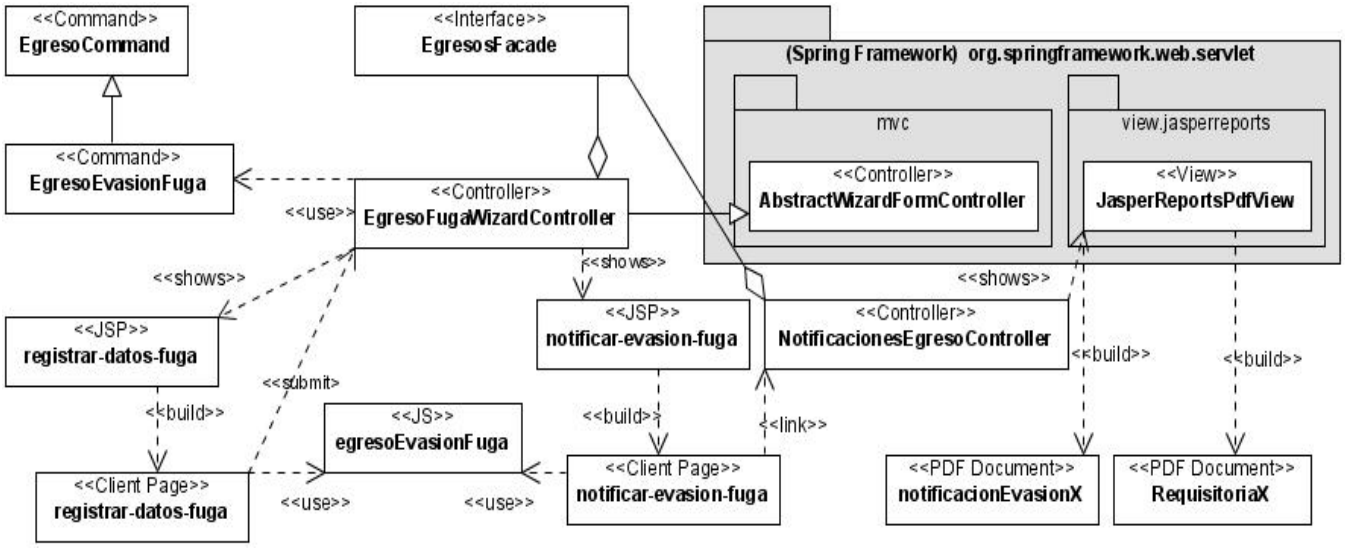
8.2.2 DCD: Egreso por Defunción



8.2.3 DCD: Egreso por Evasión



8.2.4 DCD: Egreso por Fuga



GLOSARIO

C

1. **CICPC**: Cuerpo de Investigaciones Científicas, Penales y Criminalísticas**.**
2. **CTC**: Centro de Tratamiento Comunitario**.**

D

3. **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 de proyecto SIGEP.
4. **DGSP**: Dirección General de Servicios Penitenciarios**.**

F

5. **FACP**: Fórmula Alternativa de Cumplimiento de la Pena.

I

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

P

7. **Penado:** Persona del sexo femenino o masculino que se encuentra en un centro penitenciario en cumplimiento de una sanción firme de privación de libertad**.**
8. **Procesado:** Persona de sexo femenino o masculino que se le señale como autor o participe de un hecho punible y que ingresa al Sistema Penitenciario en virtud de Auto de Privación Judicial de libertad dictado por el Juez de Control o de Juicio según el estado del proceso.
9. **Proceso Elemental del Negocio:** Documento que recoge las especificaciones del negocio asociadas a un módulo del SIGEP

R

10. **Régimen intramuros:** Establecimientos penitenciarios cerrados dónde cumplen sanción los individuos privados de libertad o los individuos que se encuentran bajo una medida cautelar privativa de libertad.
11. **Régimen extramuros:** Centros penitenciarios abiertos, en estos centros solo se ingresa por otorgamiento de régimen abierto, libertad condicional, suspensión condicional de le ejecución de la pena y suspensión condicional del proceso.

12. **Requisitoria**: Documento que contiene los datos identificativos y foto del individuo, este se elabora cuando el individuo se evade o fuga del centro penitenciario y tiene como objetivo informar la novedad para que el individuo sea circulado.

S

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

U

14. **UTASP**: Unidad Técnica de Apoyo al Sistema Penitenciario**.**