Universidad de las Ciencias Informáticas

Facultad 2

Desarrollo del módulo Supervisión del SIGEP para el régimen Extramuros del Sistema Penitenciario Venezolano

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

Autor: Gleidis Yuriannis Rosabal Espinosa

Tutores: MSc. Karina Sánchez Tamayo

Ing. Roberto Granda Ruiz

Ciudad de La Habana, mayo de 2011

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 _____ del año _____.

_____

Gleidis Yuriannis Rosabal Espinosa

_____ _____

Ing. Roberto Granda Ruiz **MSc. Karina Sánchez Tamayo**

Datos de contacto

MSc. Karina Sánchez Tamayo

- Graduado en el 2005 de la carrera Ingeniería Informática en la CUJAE y la UHO.
- Ha trabajado en la Universidad de las Ciencias Informáticas desde septiembre del 2006.
- Obtuvo recientemente la categoría docente de Asistente y la categoría científica de Máster en Gestión de Proyectos Informáticos.
- Ha sido tutor de una tesis en la UCI en temas relacionados con software de gestión desde su vinculación laboral.
- Ha participado en eventos como UCIENCIA, Informática 2009, 2011 y otros eventos.

Ing. Roberto Granda Ruiz

- Graduado en el 2010 de la carrera Ingeniería en Ciencias Informáticas de la Universidad de las Ciencias Informáticas.
- Se vinculó a la Universidad como profesor en septiembre del 2010.
- Actualmente recibe cursos de posgrado para su superación como profesional como son MIC, Ingles Básico, DIU, CTS e IP.
- Ha estado vinculado al proyecto productivo SIGEP desde el 2007 en el rol de programador de interfaz usuario.

Agradecimientos

Agradezco infinitamente a mis tutores Karina y Roberto, por haberme ayudado en todo momento en la realización del trabajo, con paciencia y dedicación.

A la Revolución por haberme dado la oportunidad de estudiar en esta universidad, demostrando que siempre hay una oportunidad para crecernos y hacernos grandes.

Dedicatoria

Dedico esta tesis a mi bisabuela Hinginia Espinosa Rodríguez, una persona muy especial en mi vida, y aunque ya no se encuentre presente se que le encantaría haberme acompañado en este momento.

A mi mamá Adis Maris Espinosa por haber sido un gran apoyo e inculcarme los valores con los que crecí.

A mi abuela Yolanda Elena Álvarez por haberme dado el cariño y la confianza para lograr uno de mis grandes sueños.

A mi hermana Gleibys por ser mi compañera de secretos y por confiar en mí.

A Lisandro por soportarme durante todos estos años, ayudándome a pasar momentos duros y compartir mis alegrías en los momentos felices.

A mis viejos amigos, en especial a Soles María, por hacerme sentir grande y entender que podía lograrlo.

A mis amigos de la universidad, por haber echo de este lugar un bonito recuerdo.

A mis familiares por estar ahí cuando los necesité.

A todos ellos muchas gracias por ayudarme a conquistar este sueño.

Resumen

Debido a la situación precaria que presentaba el Sistema Penitenciario Venezolano, la experiencia de Cuba en este aspecto y en el marco de la colaboración Cuba-Venezuela, se desarrolla el Sistema de Gestión Penitenciaria (SIGEP), como parte del Proyecto de Humanización Penitenciaria. Dentro de este sistema fue necesaria la implementación de un módulo que tuviera en cuenta los procesos de supervisión del régimen Extramuros.

Para el desarrollo del presente trabajo se utilizó como fuente de información los requisitos definidos y validados con el cliente, así como las definiciones realizadas en el marco del proyecto. Se realiza una descripción de las herramientas, tecnologías y metodología utilizada para el análisis, el diseño y la implementación del módulo Supervisión facilitando la comprensión del documento. Dentro de estas sobresalen la plataforma de desarrollo J2EE, los frameworks Spring e Hibernate, Eclipse como IDE, Visual Paradigm como herramienta de modelado y como metodología RUP. Además, se mencionan los elementos fundamentales de la arquitectura base del sistema, que fueron un paso importante en el desarrollo del mismo. Se analizan y describen los elementos más notables de la solución de las características del sistema, el diseño e implementación y los artefactos obtenidos como resultado del proceso.

Concluido el trabajo, cumplidos los objetivos trazados, el módulo se integró a la solución SIGEPv2.1 obteniendo resultados satisfactorios en las pruebas realizadas a la aplicación. El desarrollo del módulo Supervisión permitirá que se cuente con un sistema informático para apoyar la reinserción social de los individuos.

Palabras claves

Arquitectura, Diseño, Implementación, Módulo, Penitenciario, Supervisión.

Índice

Introducción

Muchas han sido las críticas que se le han hecho a la cárcel como institución de control total, negando la supuesta función resocializadora, reeducativa, rehabilitadora o reinsertiva de la pena privativa de libertad, asignándole por el contrario una función purgatoria, consuntiva de poder, distractora, simbólica y ejecutiva. (1)

El Sistema Penitenciario de la República Bolivariana de Venezuela cuenta con establecimientos caracterizados por altos niveles de hacinamiento, por lo que los reclusos se ven obligados a vivir en condiciones infrahumanas, con el consecuente deterioro sanitario y saturación de los servicios –aparejado de retardo procesal–, falta de clasificación de la población reclusa, lo que conduce a la existencia de elevados niveles de violencia en las cárceles.

Los niveles de hacinamiento, son debido al excesivo aumento de la violencia en las calles de este país, lo que ha provocado que las prisiones venezolanas se encuentren a un 150% de su capacidad. La población reclusa de Venezuela alcanza niveles superiores cada día y es considerada como una de las más violentas de América Latina.

El 1 de abril de 1980 surge el Programa de Reinserción Social, sustentado en la puesta en vigencia en esa misma fecha de la Ley de Sometimiento a Juicio y Suspensión Condicional de la Pena, derogada en 1994. La implantación de este programa ha permitido la disminución del índice de reincidencia en el delito de los penados que han entrado al programa. Asimismo se destaca la actitud positiva mantenida por los procesados, demostrando que logran entender el beneficio de este programa.

El cambio de paradigma penitenciario se consolida con la creación del Programa de Tratamiento no Institucional, al cual rápidamente se le suma la responsabilidad de administrar el régimen abierto y la libertad condicional ya previstas en la Ley de Régimen Penitenciario de 1961, por ser medidas afines a los objetivos del mismo.

Con la llegada a la presidencia de Hugo Rafael Chávez Frías, se comienzan a realizar profundas transformaciones sociales encaminadas a resolver la situación presente en el país. Esto trajo consigo que en el año 1999 se aprobaran cambios en la Constitución de la República Bolivariana de Venezuela. En el artículo 272 de la misma se plantea: “El estado garantizará un sistema penitenciario que asegure la rehabilitación del interno o interna y el respeto a sus derechos. Para ello, los establecimientos penitenciarios contarán con espacios para el trabajo, el estudio, el deporte y la recreación, funcionarán bajo la dirección de penitenciaritas profesionales con credenciales académicas universitarias (…)”. (2)

A partir de la situación que presentaba el Sistema Penitenciario Venezolano, la experiencia de Cuba en este aspecto y en el marco de la colaboración Cuba-Venezuela, se le da la tarea a la Universidad de las Ciencias Informáticas de desarrollar el Sistema de Gestión Penitenciaria (SIGEP) en el año 2006, como parte del Proyecto de Humanización Penitenciaria. Entiéndase SIGEP como la aplicación que almacena toda la información relacionada con la atención, control y tratamiento de los privados y privadas de libertad y la gestión administrativa del Establecimiento Penitenciario.

En su primera fase se comenzó a desarrollar la aplicación al régimen Intramuros, que se considera mientras el recluso cumple su condena en un centro penal interno. Posteriormente, debido a que era necesario abarcar todo el tránsito del individuo por el régimen penitenciario venezolano, se decidió la implementación del régimen Extramuros.

Dentro del régimen Extramuros se encuentran la Unidad Técnica de Supervisión y Orientación (UTSO), que no es más que la dependencia de la Dirección Nacional de Servicios Penitenciarios encargada de la supervisión y orientación de aquellos sujetos sometidos a suspensión condicional del Proceso, Suspensión Condicional de la Ejecución de la Pena y Libertad Condicional. Además se encuentra el Centro de Residencia Supervisada (CRS).

En el régimen Extramuros existían problemas en cuanto al control que llevaba el encargado de atender a los individuos del mismo. Como ejemplo de ello se puede mencionar que no existía un registro informático que sirviera de apoyo para el desarrollo de esta tarea, además de que los procesos que se realizaban en cuanto al control de los datos de las personas ubicadas en este régimen se hacía manualmente, lo cual resultaba engorroso a la hora de encontrar algún dato acerca de los “ex-reclusos” y generaba un gran cúmulo de informaciones. Además, la realización de largas entrevistas por parte del Delegado de Prueba[1] se hacía más difícil mediante el llenado manual de las mismas. Muchas veces esto también ocasionaba la pérdida de información importante que se encuentra dentro del expediente de la persona.

Teniendo en cuenta la situación anterior se define como **problema a resolver**: El proceso que existe actualmente para la supervisión de los individuos en el régimen Extramuros del Sistema Penitenciario Venezolano es insuficiente para lograr la reinserción social de los Casos[2].

Se define como **objeto de estudio** la supervisión de los individuos en el régimen Extramuros del Sistema Penitenciario Venezolano.

[1] Delegado de Prueba: “Especialista encargado de realizar las funciones de orientación, supervisión y atención a los Casos.”

[2] Caso: Individuo que cumple una medida alternativa de cumplimiento de la pena o se le ha otorgado un beneficio en un CRS o en una UTSO.

Teniendo en cuenta el problema planteado se define como **objetivo general:** Desarrollar el módulo Supervisión del régimen Extramuros del SIGEP para apoyar la reinserción social de los Casos.

Se especifica como **campo de acción** el módulo Supervisión del régimen Extramuros.

Por tanto se plantean como objetivos específicos:

- Definir el marco teórico de la investigación.
- Realizar un refinamiento de los casos de uso para el desarrollo del módulo Supervisión.
- Diseñar el módulo Supervisión para el régimen Extramuros.
- Implementar el módulo Supervisión para el régimen Extramuros.
- Realizar pruebas a la solución propuesta.

Para guiar el cumplimiento de los objetivos se definen como **tareas de investigación**:

- Estudio sobre aplicaciones web que gestionen la información que proviene de las distintas áreas de un centro penitenciario.
- Selección de metodología, tecnologías, lenguajes y herramientas para el desarrollo de la aplicación.
- Análisis de los requisitos de software correspondientes al módulo Supervisión.
- Definición de patrones de diseño y arquitectura para la posible solución.
- Realización y documentación del diseño de cada una de las capas del módulo siguiendo el flujo de trabajo definido para el proyecto.
- Realización y documentación del diseño del Modelo de Datos asociado a las funcionalidades del módulo Supervisión.
- Implementación de las clases definidas en el diseño.
- Realización de pruebas a la solución propuesta.
- Integración del módulo a la solución SIGEPv2.1.

Se plantea como **idea a defender**: El desarrollo del módulo Supervisión permite apoyar la supervisión de los Casos en el régimen Extramuros del Sistema Penitenciario Venezolano facilitando la reinserción social de los mismos.

La realización de las tareas estuvo sustentada por un conjunto de métodos de investigación:

- **Analítico – Sintético:** Este método que posee una base objetiva de la realidad se ha usado durante la confección del diseño teórico de la investigación. Se realizó un análisis sobre los sistemas penitenciarios, las diferentes tecnologías y herramientas; dividiendo su estudio en partes para un mejor entendimiento y luego realizar una síntesis específica y concreta de estos elementos.

- **Histórico – Lógico:** Este método ha sido usado en el estudio de la trayectoria y evolución de los productos informáticos existentes en diferentes países que se relacionan con los sistemas penitenciarios.
- **Modelación:** Al usar símbolos, que tributaron a la realización de Diagramas del Modelo de Diseño y de Implementación del módulo.

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

- Diagrama y Descripción de Casos de Uso.
- Modelo de Diseño
- Modelo de Implementación
- Módulo Supervisión para el régimen Extramuros funcional integrado al SIGEP v2.1.

El trabajo está estructurado de la siguiente manera:

Capítulo 1 Fundamentación Teórica: Contiene la fundamentación teórica en la que se hace un estudio del estado del arte del tema tratado. Se especifica la metodología utilizada. Se describen algunas características de las herramientas y tecnologías utilizadas para el desarrollo del módulo Supervisión del régimen Extramuros del SIGEP.

Capítulo 2 Características del Sistema: Contiene la descripción del sistema donde se obtienen como principales artefactos el Diagrama y la Descripción de Casos de Uso.

Capítulo 3 Diseño del Módulo: Contiene la descripción de los elementos principales del diseño a través de artefactos obtenidos como parte del proceso de desarrollo del módulo Supervisión.

Capítulo 4 Implementación y Pruebas: Contiene la descripción de los elementos principales de la implementación a través de artefactos obtenidos como parte del proceso de desarrollo del módulo Supervisión. Se realizan pruebas para validar y verificar la calidad de la solución propuesta.

Capítulo 1 Fundamentación Teórica

1.1 Introducción

En este capítulo se caracterizan sistemas informáticos penitenciarios que existen a nivel nacional y mundial. Esta caracterización permite determinar sus fortalezas y debilidades, además de verificar si se implementan o no los procesos de supervisión en el desarrollo de los mismos. Se definen algunos conceptos claves que sirven de apoyo para la comprensión de lo abordado en el documento. También se analizan las tecnologías, la metodología y las herramientas propuestas para el desarrollo del módulo Supervisión, mencionando algunas de las características que poseen, así como ventajas que trae para el SIGEP el uso de las mismas.

1.2 Sistemas Penitenciarios

En el Diccionario de Ciencias Jurídicas, Políticas y Sociales se entiende por sistema penitenciario el adoptado para castigo y corrección de los penados y el régimen o el servicio de los establecimientos destinados a ese objeto.

En el mismo también se define sistema penitenciario como régimen penitenciario. Entiéndase este último como "el conjunto de normas legislativas o administrativas encaminadas a determinar los diferentes sistemas adoptados para que los penados cumplan sus penas. Se encamina a obtener la mayor eficacia en la custodia o en la readaptación social de los delincuentes. 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. Entre ambos extremos existe una amplia gradación[3]". (3)

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. (4)

1.2.1 Supervisión

En el Diccionario de la Real Academia Española se define supervisar como "ejercer la inspección superior en trabajos realizados por otros."

[3] Gradación: "Serie de cosas ordenadas gradualmente".

La supervisión es la observación regular y el registro de las actividades que se llevan a cabo en un proyecto o programa. Es un proceso de recogida rutinaria de información sobre todos los aspectos del mismo. Supervisar es controlar el progreso las actividades del proyecto. Es observación sistemática e intencionada.

1.2.2 Proceso de Supervisión del SIGEP

Dentro del régimen Extramuros se encuentran las Unidades Técnicas de Supervisión y Orientación (UTSO) y los Centros de Residencia Supervisada (CRS). En ambas sedes, se llevan a cabo procesos relacionados con la supervisión, la orientación y atención a los Casos, los cuales son asignados al Delegado de Prueba**.** El proceso de supervisión tiene como objetivo ofrecer a cada residente una asistencia, supervisión y orientación personalizada, dirigida a lograr un cambio positivo en su conducta y su reinserción efectiva en la sociedad.

1.2.3 Sistemas Informáticos Penitenciarios

A nivel nacional y mundial existen diversos Sistemas Informáticos Penitenciarios que sirven de apoyo a los procesos que se desarrollan en los diferentes centros penitenciarios, lo cual facilita un control más estricto y eficiente de todo un conjunto de actividades dentro de los mismos. La mayoría de estos sistemas se han desarrollado en conjunto con los diferentes organismos nacionales e internacionales especializados en el tema. Estos han ayudado al control y estandarización de la información que de estos se genera.

Para el desarrollo del módulo Supervisión del SIGEP se tuvieron en cuenta varios sistemas informáticos, sus entornos de desarrollo, sus procesos, sus características, lo cual permitió la identificación de fortalezas y debilidades. Ejemplo de estos sistemas son:

- Sistema Automatizado para el Control del Recluso SACORE (Cuba).
- Sistema de Gestión Penitenciaria SIGPEN (Ecuador).
- Sistema Penitenciario del Gobierno de Panamá.
- Establecimiento Penitenciario Virtual en Red EPVNET.

Sistema Automatizado para el Control del Recluso SACORE (Cuba)

Surge para dar cumplimiento a la Orden 43/99 del Vice Ministro Primero del Ministerio del Interior de Cuba y tiene las siguientes características:

- Garantiza respuestas inmediatas a las solicitudes de información de los diferentes órganos e instituciones del estado como son: Jefatura del MININT[4], Ministerio de Justicia, Tribunales, Fiscalías, MINED[5], INDER[6], FMC[7], MINFAR[8].
- Recoge prácticamente la totalidad de la información de los reclusos en todas las especialidades.
- Tiene más de 200 reportes impresos.
- Permite la recuperación dinámica a partir de una solicitud de búsqueda.
- Los partes que se emiten son obtenidos de forma automatizada.
- Permite el traslado automático de todos los datos del recluso al nivel nacional.

El SACORE no contempla procesos relacionados con los CRS, ni centros de Rehabilitación, lo más cercano son los Campamentos, donde los penados salen a trabajar con una rigurosa custodia.

Sistema de Gestión Penitenciaria SIGPEN (Ecuador)

El Sistema de Gestión Penitenciaria SIGPEN facilita el control y manejo de las actividades que realizan los Centros de Rehabilitación Social en cada una de sus áreas, logrando de esta manera tener un control adecuado y oportuno de la información de las personas privadas de libertad.

El Ministerio de Justicia, Derechos Humanos y Cultos conjuntamente con la Dirección Nacional de Rehabilitación Social están en continuo monitoreo de las actividades realizadas en los CRS por medio del sistema SIGPEN, el cual permite obtener datos estadísticos reales de la situación de los sistemas penitenciarios, logrando de esta manera tomar decisiones eficaces y oportunas para mejorar el ambiente carcelario del país.

Está compuesto por diferentes secciones (Interno, Departamento Jurídico, Departamento Médico, entre otras) las cuales brindan toda la información referente al estado de los reclusos en la sección elegida.

Este sistema tiene los centros de Rehabilitación Social, los cuales presentan varias actividades que permitan continuar con la preparación de los internos para su reinserción en la sociedad.

Sistema Penitenciario del Gobierno de Panamá.

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

Este sistema fue creado a principios de año 1997, como resultado de un proyecto financiado por las Naciones Unidas y el gobierno español en coordinación con la Dirección General de Sistemas

[4] MININT: Ministerio del Interior
[5] MINED: Ministerio de Educación
[6] INDER: Instituto Nacional de Deportes, Educación Física y Recreación.
[7] FMC: Federación de Mujeres Cubanas
[8] MINFAR: Ministerio de las Fuerzas Armadas Revolucionarias

Penitenciarios de Panamá (DGSP). La finalidad del software es mantener en una base de datos los registros de los internos que están detenidos en los centros penales a nivel nacional.

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

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

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

Establecimiento Penitenciario Virtual en Red EPVNET

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

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

Tiene como precedente en España algunas aplicaciones basadas en la arquitectura Cliente-Servidor y bajo entorno Windows que cubren algunas áreas o 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 la encargada de la supervisión de los individuos para su reinserción en la sociedad.

En el año 2002, Juan Carlos Moral comenzó a trabajar en el proyecto EPVNET. En el año 2004, lo convirtió en un proyecto de software libre con licencia GNU/GPL[9], para que pudiera ser utilizado por otras personas. Para la realización del mismo se utilizó como lenguaje de programación PHP4 y como gestor de base de datos PostgreSQL, así como también HTML, CSS2. Puede ser utilizado en sistemas operativos como Linux y Windows.

El EPVNET se encuentra basado en la legislación española, garantizando el control de los movimientos, de comunicaciones y de informes que se realizan en los centros penitenciarios. Tiene automatizado los

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

procesos de: Ingresos y Salidas, Centro de Control, Comunicaciones, Seguridad y Accesos, Situaciones Regimentales, Incidentes Regimentales, Registros de Correspondencia Oficial de Entrada y Salida y Requisas, entre otras; según el propio creador.

Análisis General

Con el estudio realizado de los mencionados sistemas penitenciarios se determinó que no se tiene en cuenta un proceso para la supervisión de los ex-reclusos que por su buena actitud son merecedores de la suspensión condicional de la pena. Solo se contemplan procesos elementales de los centros penitenciarios.

Muchos de estos sistemas no se encuentran actualizados ya que existen muchos centros que por falta de infraestructura están conectados a la base de datos central. También existen sistemas cuyos centros penitenciarios no poseen interconexión, imposibilitando la integración y la centralización de las informaciones que se generan. Casi todos reflejan la situación legislativa y judicial perteneciente a un país específico, lo cual no permite que estos sean adaptados para su uso en otros sistemas penitenciarios.

En los sistemas penitenciarios donde se implementan los sistemas informáticos antes mencionados se logra automatizar los procesos para los cuales están diseñados y funcionan como herramienta de trabajo.

1.3 Sistema Penitenciario de Venezuela

El Sistema Penitenciario Venezolano, no es un elemento aislado de la situación que se presenta en América Latina, debido a que presenta características similares a los sistemas penitenciarios descritos anteriormente. Los procesos se manejaban sin la existencia de estándares para el registro informático de los mismos, así como tampoco se contaba con la presencia de sistemas informáticos que apoyaran estos procesos.

Con el desarrollo del SIGEP en colaboración con instituciones especializadas cubanas se plantea la solución 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 penal venezolano y se convierta en herramienta de trabajo para funcionarios en los centros penitenciarios y para la toma de decisiones a nivel institucional.

Con este sistema automatizado se busca:

- Aumentar la eficacia, profesionalismo y equidad en el sistema penitenciario venezolano, de manera que se logre un incremento de la confianza en el sistema penitenciario en general.
- Generar y diseminar información vital para el funcionamiento de los establecimientos penitenciarios. Esto permitirá generar estadísticas confiables y actualizadas sobre la situación jurídica de los

privados de libertad, condiciones de vida y salud, actividades de rehabilitación y reinserción, la situación operativa, la actividad administrativa, entre otras.

- Permitir la comunicación en línea con tribunales, sistemas de identificación y antecedentes penales, que complementan la información necesaria para la gestión de los procesos vinculados a los privados de libertad.

1.4 Tecnologías y herramientas utilizadas

Es necesario realizar una buena selección de herramientas y tecnologías que conforman el entorno de desarrollo, las cuales serán usadas para dar solución a determinados problemas informáticos, debido a la gran importancia que revisten, pues son imprescindibles en la producción de un software e inciden directamente en el tiempo de desarrollo y de manera general en la calidad del producto final.

Todas las tecnologías y herramientas utilizadas en el SIGEP fueron definidas por el grupo de arquitectura en etapas anteriores a su desarrollo. A continuación se identifican herramientas, tecnologías y metodología utilizadas en el diseño e implementación del módulo Supervisión.

1.4.1 Metodología de desarrollo

Una metodología, (del griego metà "más allá", odòs "camino" y logos "estudio"), hace referencia al conjunto de procedimientos basados en principios lógicos, utilizados para alcanzar una gama de objetivos que rigen una investigación científica o una exposición doctrinal.

Una metodología de desarrollo de software se refiere a un frameworks que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. A lo largo del tiempo, una gran cantidad de métodos han sido desarrollados diferenciándose por su fortaleza y debilidad.

El framework para metodología de desarrollo de software consiste en:

- Una filosofía de desarrollo de software con el enfoque del proceso de desarrollo de software.
- Herramientas, modelos y métodos para asistir al proceso de desarrollo de software.

Estos frameworks son a menudo vinculados a algún tipo de organización, que además desarrolla, apoya el uso y promueve la metodología. La metodología es a menudo documentada en algún tipo de documentación formal.

Rational Unified Process

Rational Unified Process es una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable. Es un Proceso Práctico. (5)

Su meta es asegurar la producción de software con la más alta calidad, que reúna las necesidades de los usuarios dentro del cronograma planeado y la inversión prevista. RUP deja bien definido y estructurado el proceso de diseño de un software. Define claramente quién es responsable, cómo se hacen las cosas y cuándo hacerlas. (6)

El ciclo de vida RUP es una implementación del desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones y se representa en la siguiente figura.

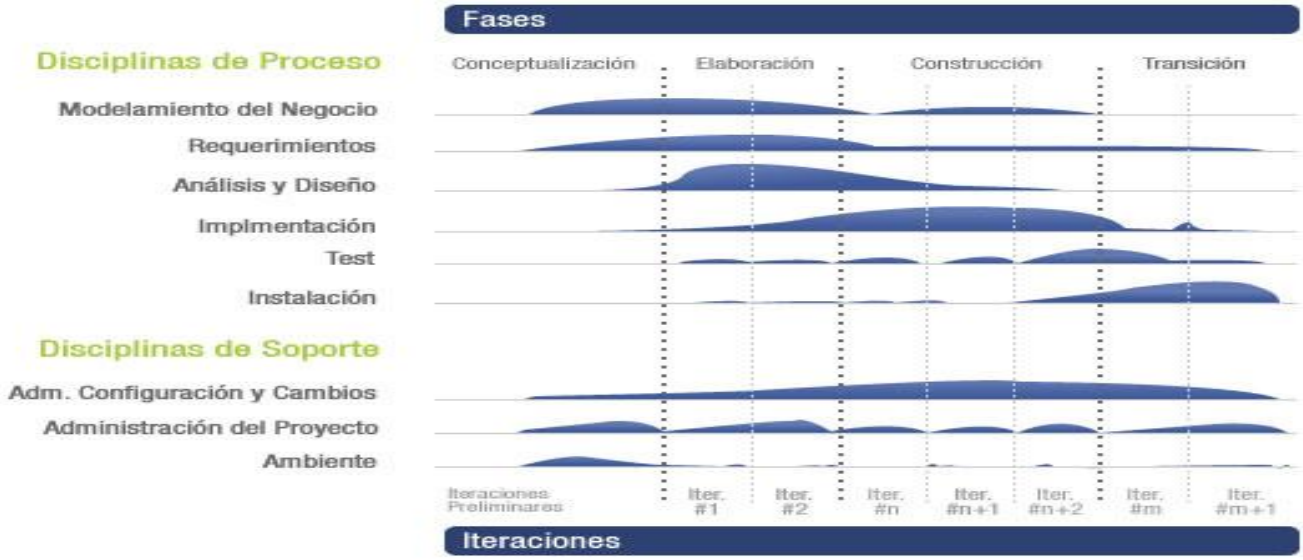


Figura 1 Ciclo de vida de RUP

Mantiene una estructura bien definida del ciclo de vida de un proyecto y, de forma general, está caracterizado por los siguientes aspectos:

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

En el SIGEP se decidió la utilización de esta metodología por ser adaptable a las necesidades del mismo, además de ser una metodología usada frecuentemente en los proyectos de gran envergadura como es el caso, unificando todo el equipo de desarrollo de software y optimizando su comunicación, proveyendo a

cada miembro de una aproximación al desarrollo de software con una base de conocimiento de acuerdo a las necesidades específicas del proyecto.

1.4.2 Lenguaje Unificado de Modelado

(LUM o **UML**, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Se utiliza para definir un sistema, para detallar los artefactos en el sistema, para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar. (7)
Desde el punto de vista puramente tecnológico, UML tiene una gran cantidad de propiedades que han sido las que, realmente, han contribuido a hacer de este el estándar de facto de la industria que es en realidad.

Algunas de las propiedades de UML como lenguaje de modelado estándar son:

- Concurrencia: es un lenguaje distribuido y adecuado a las necesidades de conectividad actual y futura.
- Ampliamente utilizado por la industria desde su adopción por OMG.
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soporta tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clases, componentes y nodos.
- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.
- Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboraciones, que sirven para evaluar el estado de las máquinas.

UML proporciona tres beneficios claves: la visualización, la gestión de la complejidad y la comunicación clara. Contiene varios diagramas que en la figura siguiente se pueden apreciar.

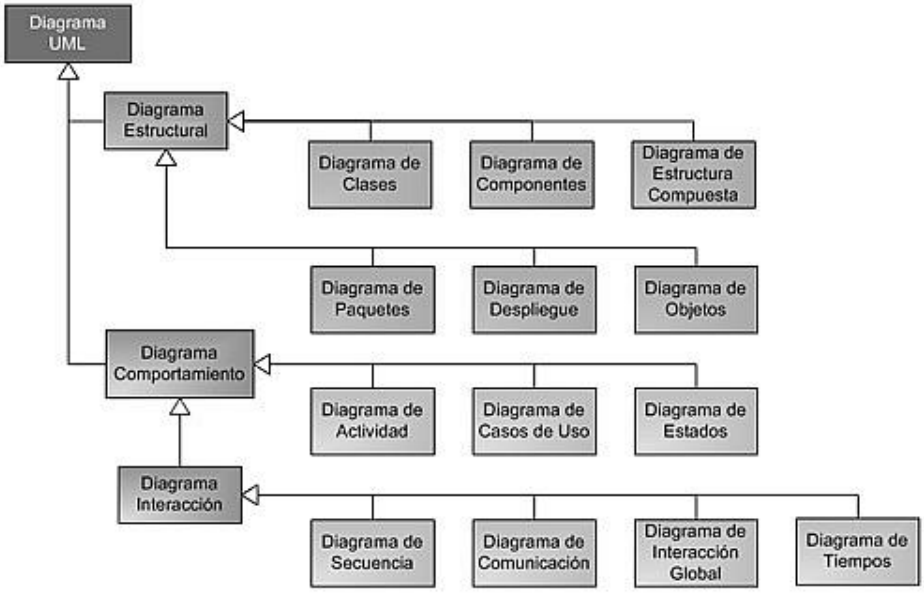


Figura 2 Diagramas UML (8)

1.4.3 Herramienta de modelado

Visual Paradigm v6.4

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

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Características de Visual Paradigm

- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento.
- Modelado colaborativo con Subversion.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, Esquemas XML, XML,.NET exe/dll, CORBA IDL
- Generación de código - Modelo a código, diagrama a código.
- Diagramas de flujo de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.

- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- Importación y exportación de ficheros XML.
- Integración con Visio - Dibujo de diagramas UML con plantillas de MS Visio.
- Editor de figuras.

Los beneficios que se consiguen al utilizar Visual Paradigm son varios, por un lado el uso de lenguajes visuales facilita su asimilación y entendimiento por parte del equipo de desarrollo; el tiempo invertido en el desarrollo de la arquitectura se minimiza y la trazabilidad y documentación del proyecto se realiza de una forma ordenada. Pero si hay una ventaja que destaca sobre todas las demás es la notable efectividad y productividad que se consigue en labores de diseño arquitectónico y mantenimiento haciendo uso de Visual Paradigm frente a la realización de las mismas tareas en ausencia de modelos.

Visual Paradigm se puede integrar con el IDE Eclipse lo que permite la realización de ingeniería inversa a Java y la generación de informes para generación de documentos, esto es lo que ha favorecido su elección como herramienta de modelado para SIGEP.

1.4.4 Plataforma de desarrollo J2EE

La plataforma Java 2 Enterprise Edition (J2EE) define el estándar para el desarrollo de aplicaciones empresariales de varios niveles. Simplifica las aplicaciones empresariales basándolas en componentes modulares estándar, proporcionando un completo conjunto de servicios a esos componentes, y manejando muchos detalles del comportamiento de la aplicación automáticamente, sin necesidad de programación compleja.

J2EE consiste en:

Una especificación: define los requisitos que debe cumplir una implementación de un producto J2EE.

Un Modelo de Programación: es una guía de diseño para desarrollar aplicaciones J2EE.

Una plataforma: conjunto de *API*[10]*s*, tecnologías y herramientas de desarrollo.

Una implementación de referencia: Implementación de ejemplo de los servicios brindados por la plataforma.

[10] 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.

Java EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas de Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, JavaServer Pages y varias tecnologías de servicios web. Dentro de estas especificaciones mencionadas, las usadas en el SIGEP son servlets, Java Server Pages y XML.

Java Servlets

Los Servlets son módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java. Los servlets, son objetos que corren dentro del contexto de un contenedor de servlets y extienden su funcionalidad. El uso más común de los mismos es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web. Java Servlet provee a los desarrolladores web de un mecanismo simple y consistente para extender la funcionalidad de un servidor Web y los sistemas de acceso de negocio existentes. Realiza la función de capa intermedia entre la petición proveniente de un navegador web u otro cliente HTTP.

Java Server Pages

Sitio web oficial: http://java.sun.com/products/jsp/

JavaServer Pages (JSP) permite a los diseñadores web desarrollar rápidamente y mantener fácilmente páginas Web dinámicas, que son aprovechadas por los sistemas empresariales existentes, rico en información. La tecnología JSP permite el desarrollo rápido de aplicaciones basadas en Web que son independientes de la plataforma. Separa la interfaz de usuario de la generación de contenidos, permitiendo a los diseñadores cambiar el diseño de la página en general, sin alterar el contenido dinámico subyacente.

La tecnología JavaServer Pages es una extensión de la tecnología Java Servlet. Juntos, la tecnología JSP y servlets proporcionan una alternativa atractiva a otros tipos de secuencias de comandos Web dinámica y programación, ofreciendo: independencia de la plataforma, un rendimiento mejorado, la separación de la lógica de la pantalla, facilidad de administración, la extensibilidad en la empresa, y, sobre todo, la facilidad de uso.

1.4.5 Servidor de aplicaciones Apache Tomcat v5.5.17

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

Apache Tomcat es una aplicación de software de código abierto de Java Servlet y tecnologías JavaServer Pages. Es desarrollado en un entorno abierto y participativo y publicado bajo la licencia Apache versión 2. Es la intención de ser una colaboración de los mejores desarrolladores de su clase de todo el mundo. Apache Tomcat es una marca registrada de la Fundación de Software Apache.
Como fue desarrollado usando el lenguaje de programación Java, este puede funcionar en cualquier sistema operativo que disponga de la máquina virtual de Java. Tomcat tiene seguridad de nivel de aplicación a partir de la versión 4.0, contiene además una aplicación de administración intuitiva basada en web, y mejor escalabilidad y rendimiento.

Ventajas del servidor de aplicaciones Web Tomcat:

- Es gratis
- Fácil de instalar.
- Ocupa muy poco espacio.
- Es bastante rápido.
- Es fiable
- Su solidez se basa en que miles de desarrolladores contribuyen con código.

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

1.4.6 Control de versiones

http://subversion.apache.org/

Subversion es un sistema centralizado de control de versiones de código abierto. Se caracteriza por su fiabilidad como un refugio seguro para los datos importantes, la simplicidad de su modelo y uso, y su capacidad para apoyar las necesidades de una amplia variedad de usuarios y proyectos, de los individuos y de las operaciones empresariales a gran escala.

La posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Como el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada —si se ha hecho un cambio incorrecto a los datos, simplemente se deshace el cambio

1.5 Frameworks

Un framework es “una mini arquitectura reusable que provee una estructura y comportamiento genéricos para una familia de abstracciones de software […]” (9)

En el desarrollo de Software, un framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

1.5.1 Spring Frameworks v2.0

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

Spring es un framework de código abierto, creado para disminuir la complejidad del desarrollo de aplicaciones empresariales. Sin embargo la utilidad de Spring no se limita a aplicaciones del lado del servidor, ya que cualquier aplicación en java puede usar este framework para beneficiarse de su simplicidad, su bajo acoplamiento, y capacidad de ser probado. (10)

A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa y un sustituto del modelo de Enterprise JavaBean. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Dentro de las características que presenta es que está basado en la inversión de control y utiliza la Programación Orientada a Aspectos. Está dividido en módulos, los cuales son elegidos en dependencia de la necesidad de desarrollo.

De estos módulos los más usados en la arquitectura del SIGEP son:

Spring Core (Núcleo) es la parte fundamental del framework y permite administrar la funcionalidad de contenedor de beans**.**

Spring MVC permite una separación entre código de modelo de dominio y las formas web y permite el uso de validación.

Spring ORM provee capas de integración para APIs de mapeo objeto – relacional como Hibernate, en el SIGEP está basado en el patrón DAO.

1.5.2 Hibernate v3.2.0 cr4

Sitio web oficial: *https://www.hibernate.org/*

Hibernate es un framework de código abierto que se distribuye bajo una licencia GNU LGPL. Se ha convertido muy popular en Java, como opción para gestionar la capa de acceso a datos en aplicaciones empresariales. Mientras que los lenguajes de programación modernos, como Java, proveen un modo bastante intuitivo de manejar vistas orientadas a objetos de entidades a nivel de aplicación de negocio, los datos que yacen bajo estas entidades son todavía tratados de manera relacional en las bases de datos.

Hibernate constituye la combinación de ambas tecnologías, lo cual es hoy en día aun una de las tareas más difíciles de enfrentar en la computación empresarial. (11)

Hibernate es una herramienta de Mapeo Objeto-Relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible. Ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria") que permite crear consultas mediante la manipulación de los criterios de los objetos en tiempo de ejecución. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

1.5.3 Dojo v0.4

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

Dojo es un Framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX[11]. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop APIs, widget APIs, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX. Dojo resuelve asuntos de usabilidad comunes como pueden ser la navegación y detección del navegador, soportar cambios de URL en la barra de URLs para luego regresar a ellas (bookmarking), y la habilidad de degradar cuando AJAX/JavaScript no es completamente soportado

[11] AJAX: ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML.* Es una técnica de desarrollo web para crear aplicaciones interactivas. Es asíncrona, en el sentido de que los datos adicionales se requieren al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página.

en el cliente. Proporciona una gama más amplia de opciones en una sola biblioteca JavaScript y es compatible con navegadores antiguos.

1.6 Entorno Integrado de Desarrollo

Un Entorno de Desarrollo Integrado (en inglés Integrated Development Environment) es un programa informático compuesto por un conjunto de herramientas de programación.
Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

1.6.1 Eclipse v3.4

Sitio web oficial: http://www.eclipse.org/
Eclipse es una comunidad de código abierto cuyos proyectos se centran en la creación de una plataforma de desarrollo extensible, tiempos de ejecución y los marcos de aplicación para la construcción, despliegue y gestión de software en el ciclo de vida completo del software. Es un Entorno de Desarrollo Integrado de código abierto multiplataforma. Esta plataforma, típicamente ha sido usada para desarrollar Entornos de Desarrollo Integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).
Eclipse dispone de un editor de texto con resaltador de sintaxis. La compilación es en tiempo real. Tiene pruebas unitarias con JUnit, control de versiones con CVS, integración asistentes (wizards) para creación de proyectos, clases, tests, etc., y refactorización. Asimismo, a través de "plugins" libremente disponibles es posible añadir control de versiones con Subversion e integración con Hibernate.

1.6.2 Plugins

Un plug-in (o plugin) es un conjunto de componentes de software que agrega capacidades específicas para una aplicación de software más grande. Si se admite, los plug-ins permiten personalizar la funcionalidad de una aplicación. Por ejemplo, los plug-ins son de uso común en los navegadores web para visualizar el video, escanear en busca de virus, y mostrar los nuevos tipos de archivos. Ejemplos de plug-ins más conocidos figura el Adobe Flash Player.
Algunos de los plugins de Eclipse utilizados en el SIGEP son:

Spring IDE v2.0

Spring IDE agiliza el trabajo en el framework Spring, sobre todo en los proyectos grandes ya que cuenta con editores como XML que permite el autocompletado, un validador de los beans configurados y una ampliación del servicio de búsqueda de Eclipse para buscar artefactos de Spring.

Hibernate Tools

Hibernate Tools es un conjunto de herramientas para Hibernate3 Implemented como un conjunto integrado de plugins. Posee un editor de mapeo XML, que permite el apoyo a la auto-completamiento y el resaltado de sintaxis. El editor admite incluso semántica auto-completado de nombres de clase, de propiedad, nombres de los campos, nombres de tabla y columna.

Web Tools Plataform (WTP)

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

Eclipse Web Tools Platform (WTP) integra a la perfección todas las herramientas actuales de Java que el desarrollador web necesita. Es a la vez un recurso sin precedentes de código abierto para desarrolladores de trabajo y una poderosa base para los productos comerciales del estado de la técnica. Incluye editores para JavaServer Pages, HTML, CSS, JavaScript, XML, y XSD (XML Schema Definition) y API para soportar el despliegue, ejecución y prueba de aplicaciones.

Subclipse v1.0.1

Sitio web oficial: http://subclipse.tigris.org/

Es el encargado de la conexión del IDE Eclipse a repositorios de Subversion. Este último es un sistema de control de versiones que permite el trabajo en equipo en el desarrollo de un producto, y la manipulación del proyecto que reside en este. En el caso de SIGEP es el repositorio que se usa.

SDE v4.4.1 (Smart DEvelopment Enviroment)

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

Este plug-in es una herramienta CASE que permite la integración de la herramienta Visual Paradigm con el IDE Eclipse. Soporta todos los diagramas UML generados a lo largo del desarrollo del software.

También permite hacer ingeniería inversa para obtener diagramas de clases a partir del código Java y viceversa. Fundamentalmente mantiene la sincronización de las clases con los modelos, además de generar documentación.

1.7 Gestor de base de datos Oracle 10g Standard Edition

Un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos. Ayuda a realizar las siguientes acciones:

- Definición de los datos
- Mantenimiento de la integridad de los datos dentro de la base de datos
- Control de la seguridad y privacidad de los datos
- Manipulación de los datos

Oracle

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

Oracle es un sistema de gestión de base de datos relacional, desarrollado por Oracle Corporation. Se considera como uno de los sistemas de bases de datos más completos destacando soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma. Garantiza la seguridad e integridad de los datos, la ejecución de transacciones de forma correcta sin causar inconsistencias, una fácil administración y almacenamiento de grandes volúmenes de datos.

La versión de Oracle que se usa en el SIGEP es la versión 10 G. Esta versión de Oracle incluye capacidades que permiten a los clientes integrar más fácilmente aplicaciones y datos; implementar procesos de negocios flexibles y perfeccionados; simplificar el desarrollo y la navegación Web; además de aprovechar al máximo los activos existentes de información. Los productos proveen servicios comprensivos para Servicios Web.

Diseñado para que los usuarios puedan conectar y ampliar sus sistemas heterogéneos existentes, Oracle Application Server 10g está certificada con software de código abierto, incluidos Spring, Hibernate, SubVersion, y Eclipse, las cuales son herramientas utilizadas en el proyecto SIGEP. Esta versión también incluye un motor ágil de reglas de negocios que los clientes pueden utilizar para modificar o agregar reglas de negocios sin que sea necesario rescribir las aplicaciones, lo que acelera el tiempo de llegada al mercado a la vez que reduce los costos de desarrollo.

1.8 Formato de texto

1.8.1 JSON

JSON es la abreviación de JavaScript Object Notación – Notación de Objetos de JavaScript. Es ampliamente reconocido por una gran variedad de lenguajes como Java, PHP, JavaScript, C++, C# entre otros.

Como toda herramienta de uso informático, JSON presenta algunas características que lo hacen más selecto que otros formatos de intercambio de datos. A continuación se mencionan algunas características de este formato:

- Es un subconjunto de la notación literal de objetos de JavaScript
- Es un formato alternativo de envío y recepción de datos, es decir remplaza a XML en el envío de texto plano.
- Este formato de datos es más liviano que XML.
- Hace el código más sencillo ya que utiliza el código JavaScript como modelo de datos.
- JSON proporciona un buen enfoque orientado a objetos para la caché de meta datos al cliente.
- Ayuda a verificar los datos y la separación de la lógica.

El beneficio de JSON, entonces, no es que sea más pequeño a la hora de transmitir, sino que representa mejor la estructura de los datos y requiere menos codificación y procesamiento.

1.8.2 XML

El XML es un acrónimo de eXtensible Markup Language. Es considerado como un metalenguaje de definición de documentos estructurados mediante marcas o etiquetas. En la práctica corresponde a un estándar que permite a diferentes aplicaciones interactuar con facilidad a través de la red.
XML posee características que lo hacen elegible por encima de otros metalenguajes, características que lo hacen despuntar en la comunidad internacional y facilita su uso en diferentes ramas del desarrollo web. Dentro de estas características están:

- XML es un subconjunto simplificado pero estricto de SGML (Standard Generalized Markup Language):
- Extensible: se pueden definir nuevas etiquetas.
- Estructurado: se puede modelar datos a cualquier nivel de complejidad, y su definición está en una DTD, Document Type Definition.
- Validable: cada documento se puede validar frente a una DTD, o en su defecto, se puede declarar bien formado.
- Independiente de medio: para publicar contenidos en múltiples formatos.
- Independiente de fabricante y de plataforma: para poder utilizar cualquier herramienta estándar.
- Es fácil de aprender y de usar.

En la tecnología actual XML posee una importancia muy alta ya que es la base de numerosos procesos y técnicas. Otra rama del desarrollo con la que está adquiriendo relevante importancia es en las bases de

datos, no solo como soporte para la transferencia de datos sino como formato de almacenamiento, dejando claro que las bases de datos que utilizan documentos XML como unidades de almacenamiento fundamentales son las bases de datos nativas de XML.

Se considera que la principal importancia o dato significativo de este formato de texto radica en que constituye la base o soporte para disímiles tecnologías, además obtiene un valor añadido por ser un estándar muy aceptado internacionalmente.

1.9 Conclusiones parciales

En este capítulo fueron analizados varios sistemas informáticos evidenciándose en la mayoría que se ajustan a las leyes de un país en específico y que muchas veces solo implementan procesos elementales de los sistemas penitenciarios. No obstante a esto, los que están implementados sirven como herramientas de trabajo. Se hace referencia a como con el surgimiento del SIGEP se centralizaron, estandarizaron, y se implementaron procesos fundamentales del Sistema Penitenciario Venezolano.

Para el diseño y la implementación del módulo Supervisión se utilizará como metodología RUP, como herramienta de modelado el Visual Paradigm, y como entorno de desarrollo se utiliza Eclipse, mencionándose los principales pluggins que lo integran y la forma de utilización en el proyecto. Todas las herramientas y metodologías se basan en la arquitectura que usa el SIGEP, las cuales fueron definidas en su primera iteración, destacándose que la mayoría de estas herramientas son libres.

Capítulo 2 Características del Sistema

2.1 Introducción

En el presente capítulo se representan conceptos importantes como el modelo de dominio. Se enumeran los requisitos funcionales y no funcionales que debe tener el sistema que se propone. Se realiza un refinamiento de los casos de uso previamente identificados en el proyecto, lo que permite hacer una concepción general del sistema y mostrarla mediante un Diagrama de Caso de Uso, las relaciones de los actores que interactúan con el sistema, y las secuencias de acciones que se realizan. También se elaborará una descripción en formato expandido de los casos de uso del sistema.

2.2 Modelo de Dominio

Un dominio es un área de interés, usualmente representando un espacio del problema que es un subconjunto de una o más disciplinas. Por ejemplo, la investigación del ácido desoxirribonucleico (ADN) es un dominio. (12)

El Modelo de Dominio es una representación visual estática del entorno real objeto del proyecto. Es decir, un diagrama con los objetos que existen (reales) relacionados con el proyecto que se desarrolla y las relaciones que hay entre ellos. Pero no son clases de software (aunque algunos objetos del Modelo de Dominio pueden terminar siéndolo). Se llama "de Dominio" para distinguirlo del Modelo de Negocio (Model of Business) que en RUP (Rational Unified Process) es un concepto más amplio. El Modelo de Dominio se centra en una parte del negocio, la relacionada con el ámbito del proyecto. En este contexto el término "dominio" representa una parte del "negocio". Se dice que es estática porque no representa la interacción en el tiempo de los objetos, sino que representa una visión "parada" de las clases y sus interacciones. Este modelo ayuda a comprender los conceptos que utilizan los usuarios, los conceptos con los que trabajan y con los que deberá trabajar la aplicación.

2.2.1 Diagrama de Clases del Modelo del Dominio

El Modelo de Dominio correspondiente a los procesos Entrevista y Nivel de Supervisión se muestra a continuación.

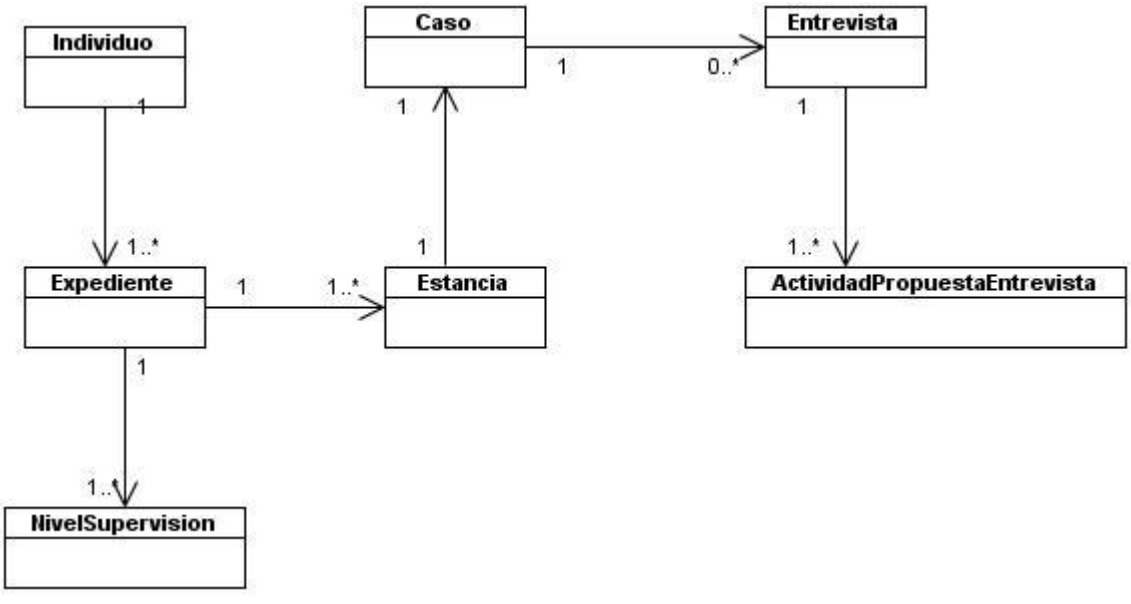


Figura 3 Modelo de Dominio

2.2.2 Descripción de las clases del dominio

- **Individuo:** Representa al procesado o penado de libertad.
- **Expediente:** Expediente penitenciario del individuo.
- **Estancia:** Período de tiempo que permanece un individuo en un centro.
- **Caso:** Individuo que está bajo una Fórmula Alternativa de Cumplimiento de Pena (FACP) o un beneficio en un Centro de Residencia Supervisada(CRS) o una Unidad Técnica de Supervisión y Orientación (UTSO).
- **Entrevista:** Constancia del encuentro del Delegado de Prueba con el individuo que atiende y supervisa.
- **Actividad Propuesta:** Actividades que quedan propuestas para realizar con el individuo una vez terminada la entrevista.
- **Nivel de Supervisión:** Frecuencia de presentación del individuo ante el Delegado de Prueba.

2.3 Requerimientos

La IEEE Standard Glossary of Software Engineering Terminology define un requerimiento como:

1. Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
2. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.
3. Una representación documentada de una condición o capacidad como en 1 o todas las ideas que los clientes, usuarios y miembros del equipo del proyecto tengan acerca de lo que debe hacer el sistema, deben ser analizadas como candidatas a requisitos. (13)

2.3.1 Requisitos Funcionales

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir. Son considerados característica requerida del sistema que expresa una capacidad de acción del mismo – una funcionalidad; generalmente expresada en una declaración en forma verbal.
Son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de como se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. (14)

Los requerimientos del módulo Supervisión son:

- **RF1 Registrar Entrevista**
- **RF2 Modificar Entrevista**
- **RF3 Eliminar Entrevista**
- **RF4 Mostrar Detalles de Entrevista**
- **RF5 Registrar Fecha de Próxima Entrevista**
- **RF6 Generar Control de Entrevista**
- **RF7 Registrar Nivel de Supervisión**
- **RF8 Mostrar Histórico de Nivel de Supervisión.**

2.3.2 Requisitos No Funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con todas las funcionalidades requeridas, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación.

Existen múltiples categorías para clasificar a los requisitos no funcionales, y a continuación se muestran algunas con las que debe cumplir el SIGEP.

- **RNF1 Requisitos de Software**

✓ La construcción de la aplicación funcionará bajo una arquitectura basada en Tres Capas.

Requerimientos mínimos para los servidores:

✓ Sistemas operativos Ubuntu 8.04 para las máquinas clientes, Windows XP para las máquinas clientes con dispositivos (Área de Control Penal) y Red Hat Advanced Server 5 para los servidores de Aplicación y de Base de Datos.

✓ Mozila Firefox 2.0

✓ Máquina Virtual de Java versión 1.3 o Superior; etc.

✓ Apache Tomcat 5.5.17 para el servidor de Aplicaciones

✓ Oracle 10g Standard Edition

✓ Jre 6.0

- **RNF2 Requisitos de Hardware**

✓ Se requiere tarjeta de red.

✓ Memoria RAM de 4GB o superior

✓ Disco duro de 250 GB o superior

✓ Procesador de 3 GHz o superior.

- **RNF3 Restricciones en el diseño y la implementación**

✓ Se usará el lenguaje de Programación Java.

✓ Se usará como arquitectura ArbaWeb compuesta por los estilos arquitectónicos Tres Capas y modelo Cliente Servidor.

- **RNF4 Portabilidad**

✓ El sistema será multiplataforma, debe ser compatible con los sistemas operativos Windows y Linux.

- **RNF5 Confiabilidad**

✓ La herramienta de implementación a utilizar debe tener soporte para recuperación ante fallos y errores. La información manejada por el sistema está protegida de acceso no autorizado y de divulgación a terceras personas.

- **RNF6 Legales**

- ✓ El componente debe ajustarse y regirse por la ley, decretos leyes, decretos, resoluciones y manuales establecidos, que norman los procesos que serán automatizados. Este sistema estará regido por las leyes establecidas en el Sistema Penitenciario de Venezuela.

- ➢ **RNF7 Rendimiento**
- ✓ El tiempo de respuesta debe ser rápido por lo que no debe exceder un tiempo máximo de 5 segundos. Se debe garantizar la integridad, disponibilidad y confidencialidad de la información manipulada, y el sistema no debe estar fuera de servicio más de 2 horas a partir de un fallo.

2.4 Descripción del Sistema

Para la definición de los casos de uso se hizo un refinamiento de los que previamente se habían descrito en el proyecto, para que dichos casos de uso cumplieran con algunos patrones que a continuación se mencionarán.

Completar una Única Meta: Este patrón plantea que objetivos inadecuados pueden perder a los escritores a la hora de determinar dónde un caso de uso termina y otro comienza. Por lo que es necesario escribir cada caso de uso dirigiéndose hacia una completa y bien definida meta, ya que un caso de uso que reúne muchas metas se convierte en algo complicado, confuso a la hora de leerlo y difícil de desarrollar.

El nombre revela la intención: Con este patrón se hace referencia a la utilización de nombres descriptivos para los casos de uso, como una buena práctica, porque ellos revelan exactamente la intención de cada caso de uso. El nombre debe reflejar el objetivo e intención que el actor está intentando lograr. Un nombre apropiado facilita el manejo del caso de uso y permite tener una vista general del trabajo en su conjunto.

Escenario más Fragmentos: Este patrón plantea que el lector debe ser capaz de seguir el camino a través del flujo específico en que ellos están interesados, para ello se deben escribir los eventos del flujo principal como un escenario simple sin considerar posibles fallos.

Alternativas Exhaustivas, Íntegras: El problema por el cual es importante utilizar este patrón está dado ya que un caso de uso puede tener muchas alternativas, pero si falta algún recurso, los desarrolladores pueden entender mal el comportamiento del sistema y entonces el sistema sería deficiente. Para cumplir con este patrón es importante capturar todos los fallos y alternativas que deben ser manejados en el caso de uso. Una vez que se tengan identificados todos los casos de uso y su flujo principal, es necesario

identificar y capturar tantas variaciones como sea necesario, para que los desarrolladores conozcan los errores y se manejen los mismos.

Adorno o Decoración: Si se tiene alguna información que en realidad no pertenece al caso de uso pero se considera, de alguna forma valiosa, entonces se añade al caso de uso como Adorno, creando campos dentro de la plantilla del caso de uso que apoye la información auxiliar que es útil asociar con el caso de uso. Detalles no funcionales como reglas de negocio, bocetos de interfaces de usuarios, interfaces de protocolos externos, reglas de validación de datos y aún asuntos sobresalientes deben ser añadidos al caso de uso en la sección suplementaria.

Preciso y Legible: Este patrón hace referencia a que un caso de uso debe ser legible para los clientes y los desarrolladores. Cada caso de uso que se escriba debe exactamente describir una Meta Única y Completa sin ser tan verboso y que comunique la suficiente información para entenderlo adecuadamente.

CRUD: Es el acrónimo de Crear, Recuperar, Actualizar y Eliminar. Este patrón es una excepción típica al nombrar un caso de uso, según su objetivo. Es utilizado con la idea de agrupar varios objetivos en un caso de uso, llamado por convención, Gestionar<X>. Por ejemplo, los objetivos "editar usuario", "eliminar usuario", etcétera, todos se satisfacen en el caso de uso Gestionar Usuarios.

2.4.1 Definición de actores

Un actor es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Un actor caracteriza las interacciones que los usuarios exteriores pueden tener con el sistema. En tiempo de ejecución, un usuario físico puede estar limitado a los actores múltiples dentro del sistema. Diferentes usuarios pueden estar ligados al mismo actor y por lo tanto pueden representar casos múltiples de la misma definición de actor. (James Rumbaugh). (7)
Durante el levantamiento de requisitos se definió un único actor del sistema y a continuación se realiza su descripción.

Actor	Descripción
Delegado de Pruebas	**Es el especialista que atiende y supervisa a los individuos.**

2.4.2 Modelo de Casos de Uso del Sistema

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. (15)

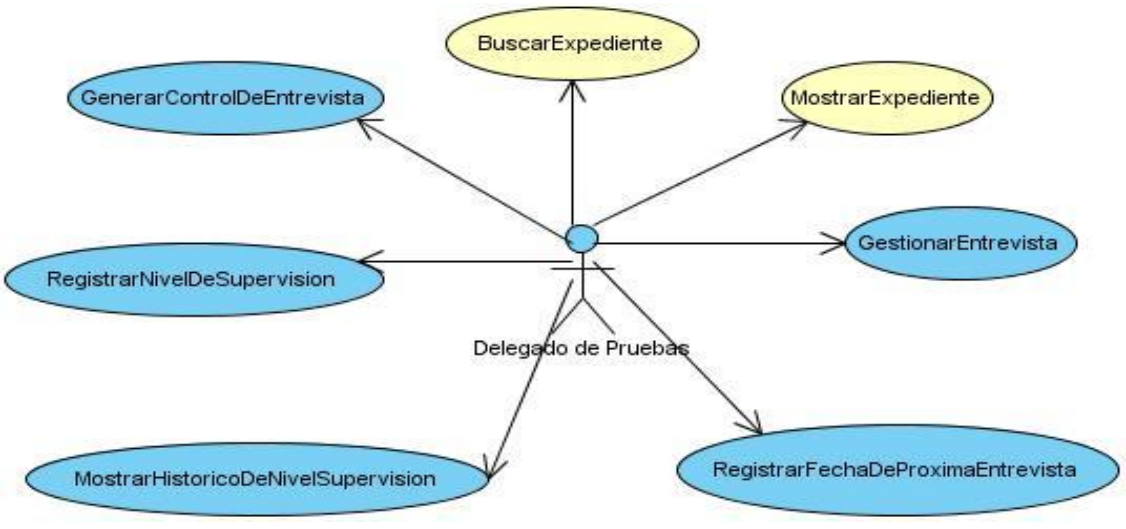


Figura 4 Diagrama de Casos de Uso

En el diagrama se representan los casos de uso BuscarExpediente y MostrarExpediente de un color distinto al de los demás, debido a que estas funcionalidades las realiza el módulo pero no se implementan en él.

2.5 Descripción Textual del Casos de Uso Gestionar Entrevista

A continuación se muestra la descripción del casos de uso Gestionar Entrevista.

Gestionar Entrevista

Caso de Uso:	Gestionar Entrevistas
Actores:	Delegado de Prueba
Resumen:	El caso de uso se inicia cuando el Delegado de Prueba indica que desea registrar, modificar, consultar o eliminar entrevistas. Se muestra un listado de todas las entrevistas en las que ha participado o no el individuo. Por cada entrevista se permite consultar los detalles, modificar y eliminar la información asociada. El caso de uso termina cuando el

	Delegado de Prueba ha registrado, modificado, consultado o eliminado correctamente las entrevistas.
Precondiciones:	El Delegado de Prueba se ha identificado y autenticado correctamente en el sistema.
Referencias:	RF1,RF2,RF3.RF4
Prioridad:	Media
Complejidad:	Media
Acción que inicia el Caso de Uso:	El Delegado de Prueba indica que desea gestionar las entrevistas de un individuo.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
El Delegado de Pruebas necesita Registrar, Modificar, Eliminar o Consultar algún detalle de una entrevista realizada.	a) Si el Delegado de Pruebas decide registrar una entrevista realizada, ir a la sección “Registrar Entrevista” b) Si el Delegado de Pruebas decide modificar una entrevista realizada, ir a la sección “Modificar Entrevista” c) Si el Delegado de Pruebas decide eliminar una entrevista realizada, ir a la sección “Eliminar Entrevista” d) Si el Delegado de Pruebas decide consultar detalles de una entrevista realizada, ir a la sección “Mostrar Detalles de Entrevista”
Sección Registrar Entrevista	
Acciones del Actor	**Respuesta del Sistema**
1.1 El Delegado de Prueba indica si el individuo estuvo presente o no en la entrevista.	
	1.2 El sistema muestra los datos de la entrevista según la selección del Delegado de Prueba: • Si el individuo se presentó, ver Figura 1 • Si el individuo no se presentó, ver Figura 2.

	1.3 El sistema muestra el nombre y los apellidos del Delegado de Prueba que tiene asignado actualmente el caso. Si la entrevista fue realizada por otro Delegado de Prueba, el sistema permite registrar sus datos, ver Flujo Alterno, 1.4.a La gestión de caso fue realizada por otro Delegado de Prueba.
1.4 El Delegado de Prueba introduce los datos relacionados con la fechas de planificación y ejecución de la entrevista.	
	1.5 El sistema valida los datos introducidos.
	1.6 El sistema registra los datos de la entrevista.

Prototipo de Interfaz

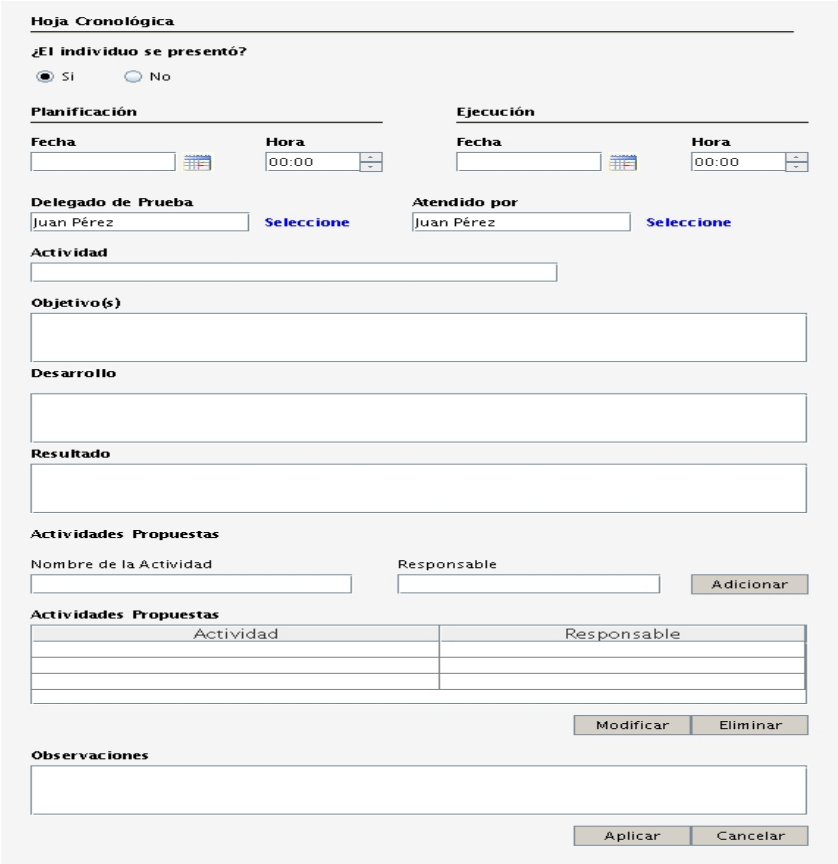


Figura 1: Datos de la entrevista cuando se presenta el individuo.

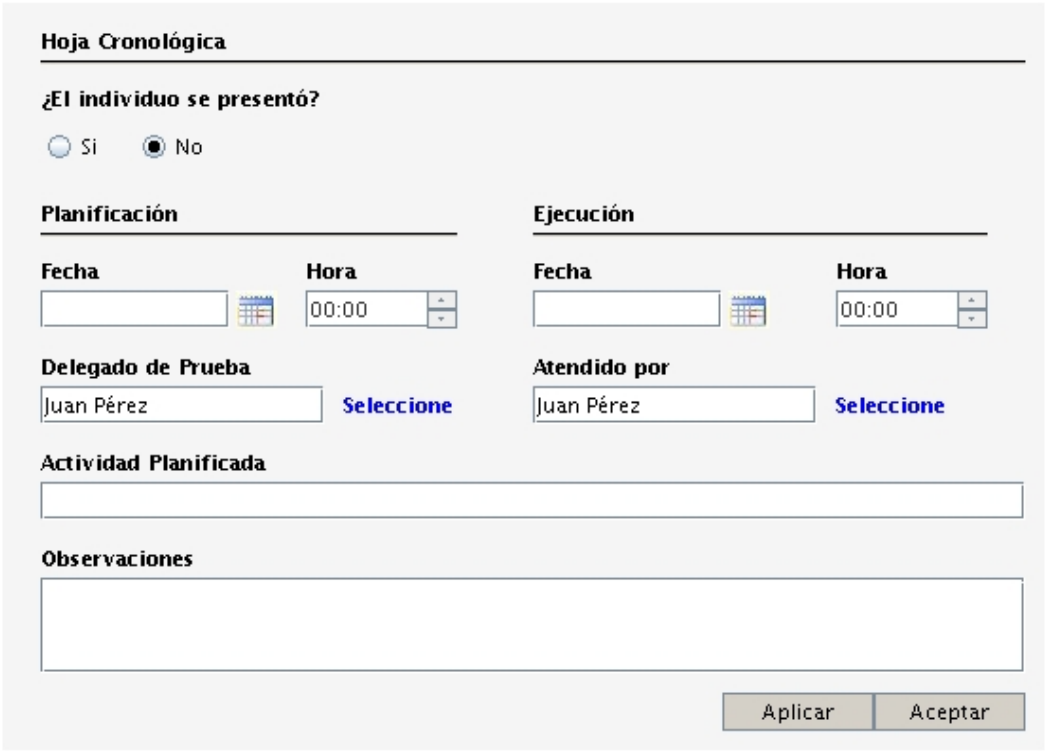


Figura 2: Datos de la entrevista cuando no se presenta el individuo.

Flujo Alterno	
1.4. a) La entrevista fue realizada por otro Delegado de Prueba.	
Acciones del Actor	**Respuesta del Sistema**
	1 El sistema brinda la posibilidad de registrar o seleccionar el nombre y apellidos del funcionario que atendió la entrevista, ver Figura 1.
2 El Delegado de Prueba indica que desea seleccionar los datos del funcionario que atendió la entrevista.	
	3 El sistema muestra los datos del funcionario y permite seleccionarlo, ver Figura 3.
4 El Delegado de Prueba selecciona el funcionario deseado.	
	5 El sistema muestra en la Hoja Cronológica los datos del funcionario, ver

	Figuras 1 y 2 respectivamente.
	6 Ir al paso 1.5 del Flujo Normal de Eventos Registrar Entrevista.

Prototipo de Interfaz

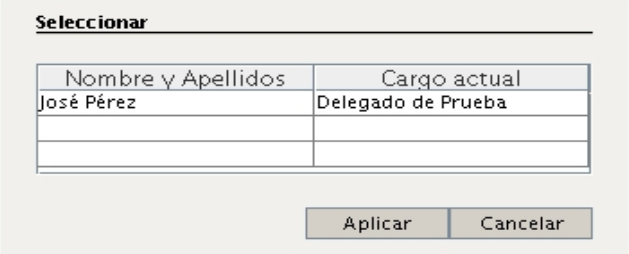


Figura 3: Seleccionar Funcionario.

***Existen errores en los datos introducidos por el Delegado de Prueba.**

Acciones del Actor	**Respuesta del Sistema**
	1 El sistema muestra los errores en los datos introducidos.
2 El Delegado de Prueba rectifica los datos introducidos e indica que desea guardarlos en el sistema, ir al paso 1.6 del Flujo Normal de Eventos Registrar entrevista, o ir al paso 2.3 del Flujo Normal de Eventos Modificar Entrevista.	

Prototipo de Interfaz

No aplicable

***El Delegado de Prueba indica que desea cancelar.**

Acciones del Actor	**Respuesta del Sistema**
	1 El sistema solicita que se confirme si desea cancelar la operación, ver Figura 4.
2 El Delegado de Prueba confirma la cancelación.	

Prototipo de Interfaz

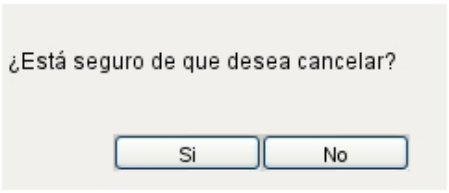


Figura 4: Cancelar.

Sección Modificar Entrevista	
Acciones del Actor	**Respuesta del Sistema**
	2.1 El sistema muestra de manera editable los datos que tiene registrados de la entrevista y permite modificarlos, ver Figuras 1 y 2.
2.2 El Delegado de Prueba modifica los datos de la entrevista e indica que desea guardarlos en el sistema.	
	2.3 El sistema valida los datos introducidos
	2.4 El sistema guarda los datos de la entrevista.
Sección Eliminar Entrevista	
Acciones del Actor	**Respuesta del Sistema**
3.1 El Delegado de Prueba selecciona la entrevista que desea eliminar e indica que desea eliminarla.	
	3.2 El sistema solicita confirmación para eliminar.
3.3 El Delegado de Prueba confirma la eliminación de la entrevista.	
	3.4 El sistema elimina la entrevista y actualiza la tabla que muestra todas las entrevistas realizadas.
Prototipo de Interfaz	

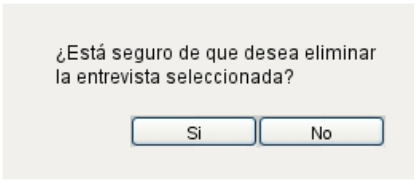


Figura 5: Eliminar entrevista.

Sección Mostrar detalles de entrevista	
Acciones del Actor	**Respuesta del Sistema**
	4.1 El sistema muestra los detalles de la entrevista seleccionada: Si en la entrevista seleccionada se registró que el individuo estuvo presente, ver Figura 6. Si en la entrevista seleccionada se registró que el individuo no estuvo presente, ver Figura 7.
4.2 El Delegado de Prueba indica que desea salir de la pantalla que muestra los detalles de la entrevista.	
	4.3 El sistema cierra la pantalla que muestra los datos de la entrevista.
Prototipo de Interfaz	

Figura 6: Consultar detalles de entrevista cuando se presentó el individuo.

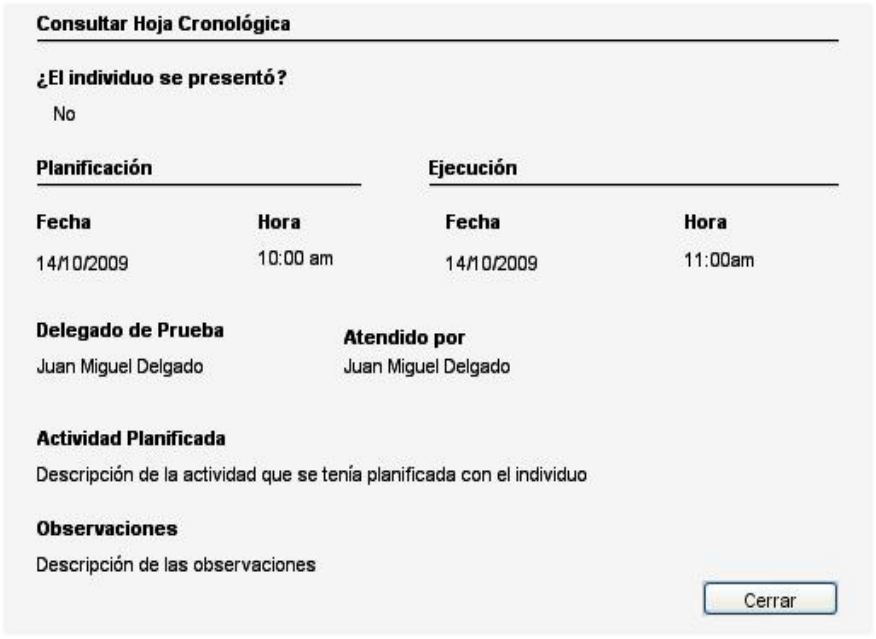


Figura 7: Consultar detalles de entrevista cuando el individuo no se presentó.

Poscondiciones	Se ha registrado, modificado o eliminado una entrevista realizada. Se pudo ver los detalles de una entrevista seleccionada.

Ver Anexo 1 para las descripciones de los Casos de Uso del Sistema correspondientes a las restantes funcionalidades.

2.6 Conclusiones parciales

En el capítulo se describió la solución propuesta a partir de los procesos fundamentales que se definieron en el negocio. Dichos procesos dieron paso a los requerimientos funcionales del sistema, los cuales fueron representados en un Diagrama de Casos de Uso del sistema y se detallaron además cada uno de ellos. A partir de los casos de uso se construirá el sistema con el objetivo de cumplir con las funcionalidades descritas en este capítulo.

Capítulo 3 Diseño del módulo Supervisión

3.1 Introducción

En el presente capítulo se proporcionan elementos de la solución de diseño por capas, para las funcionalidades definidas en el módulo Supervisión. Se realiza una pequeña descripción de la arquitectura del sistema, que será un paso fundamental para el proceso de diseño e implementación del sistema. Se muestran los resultados de cada una de las de las actividades a realizar, definidas por el SIGEP como una adecuación de la metodología de desarrollo RUP para la fase de diseño, aunque solo se muestran una parte de los diagramas realizados pertenecientes a funcionalidades significativas del módulo.

3.2 Arquitectura del SIGEP

Para el desarrollo del SIGEP se definió e implementó como arquitectura base ArBaWeb (Arquitectura Base sobre la Web), la cual se basa en los estilos arquitectónicos Cliente Servidor y arquitectura en Tres Capas.
ArBaweb tiene como fin: orientar el diseño de las capas lógicas, organizar la forma, codificar según propuestas de convenciones o estándares de códigos y recursos, brindar una estructura física para soportar el código y proponer mecanismos de colaboración entre los componentes integrados en ella. (16)
ArBaWeb propone que la aplicación se divida en tres capas: acceso a datos, lógica de negocio y de presentación, donde cada una interactúa con la inmediata superior mediante interfaces logrando un bajo acoplamiento. (16)
Un estilo arquitectónico es un conjunto coordinado de restricciones arquitectónicas que restringe los roles o rasgos de los elementos arquitectónicos y las relaciones permitidas entre esos elementos dentro de la arquitectura que se conforma a ese estilo.
La arquitectura cliente servidor consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.
En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y se hace más claro el diseño del sistema.
La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

La Programación por Capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que se necesite algún cambio, sólo se ataca al nivel requerido sin tener que revisar código de otros niveles. Permite un diseño basado en niveles de abstracción creciente, posibilitando a los implementadores, dividir un problema en una secuencia de pasos incrementales.

En esta arquitectura a cada nivel se encomienda una tarea simple, lo que permite que el diseño pueda ampliarse con facilidad en caso de que las necesidades aumenten. La siguiente figura muestra las diferentes capas de esta arquitectura.

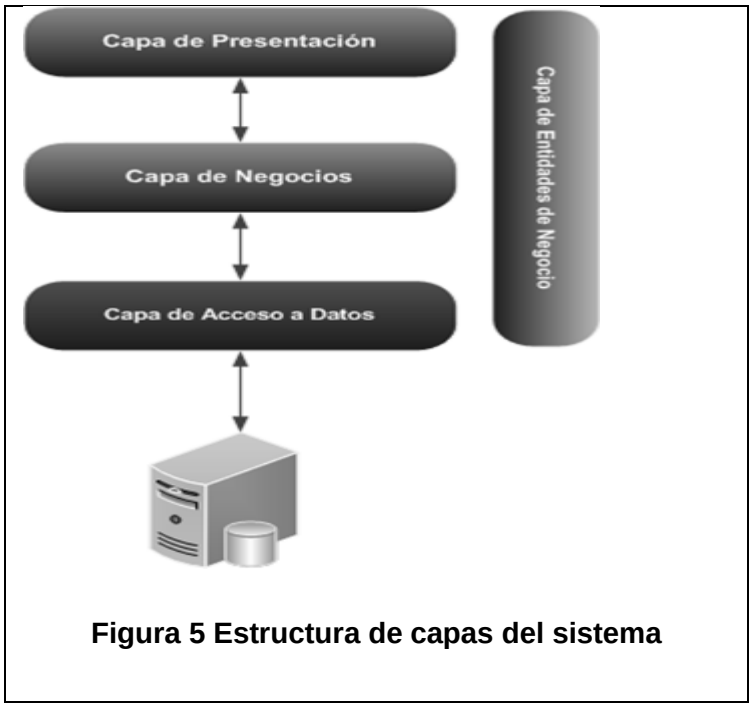


Figura 5 Estructura de capas del sistema

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

3.2.1 Capa de Acceso a Datos

En la capa de acceso a datos es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Esta capa aísla a la capa de negocio de la tecnología de persistencia, por lo que todo su diseño se basa en el objetivo principal de que exista el menor acoplamiento posible entre la aplicación y el gestor de base de datos Oracle. En el cumplimiento de este requisito, el uso del framework Hibernate ORM tiene un papel muy importante. Mapeo entre el lenguaje relacional de la base de datos y el lenguaje de objetos de la aplicación, brinda una serie de interfaces básicas muy útiles a la hora de diseñar los métodos más comunes que se realizan sobre los datos persistidos.

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

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

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

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

Por cada entidad persistente se crea una interfaz encargada de brindar a la capa de negocio, funcionalidades de acceso a datos. Cada una hereda de la interfaz base:

cu.uci.arbaweb.dataaccess.dao.BaseDAO. Las implementaciones de dichas interfaces a su vez heredan de la clase *cu.uci.arbaweb.dataaccess.dao.impl.AbstractBaseDAO*, la cual implementa la interfaz base.

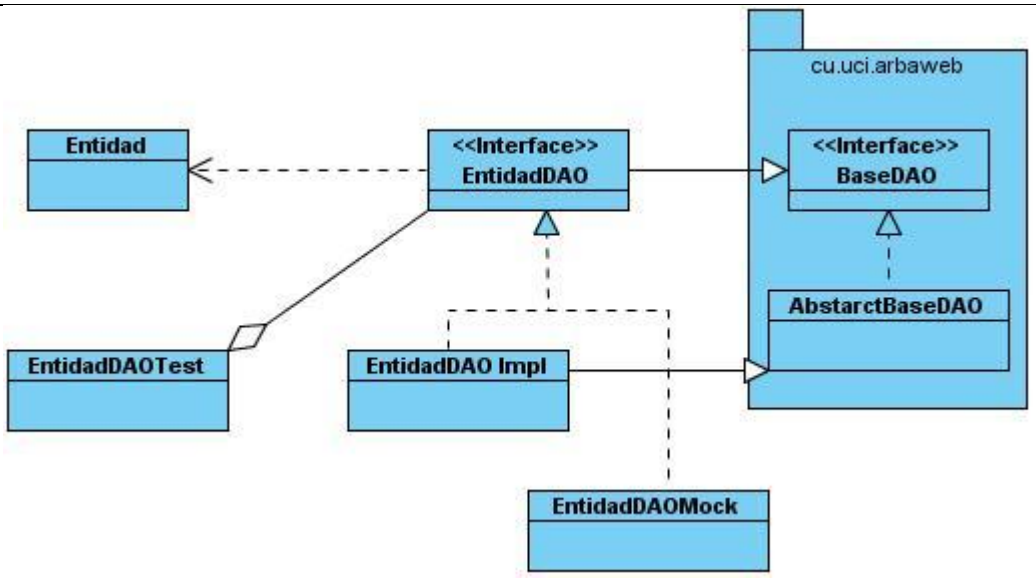


Figura 6 Capa de Acceso a Datos

3.2.2 Capa Lógica de Negocio

En la Capa de Lógica de Negocio es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina Capa de Negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

En esta capa radican los objetos de negocio o Business Objects. Los objetos de negocio separan los datos y la lógica de negocio usando un modelo de objetos. Los objetos de negocio son managers e interfaces. Cada módulo definirá una o más fachadas en caso de que se requiera, que agrupen los métodos de negocio implementados en los manejadores o managers. La fachada de un módulo se basa en el patrón Facade para permitir una clara división entre las capas arquitectónicas.

Aunque la lógica de negocio puede estar implementada en los objetos del dominio, ArBaWeb propone la creación de objetos que se encarguen de manejar las transacciones (Managers), garantizando la ejecución de cada una de las acciones que la conforman y en caso contrario, avisar de la imposibilidad de ejecución. (16)

Los Managers son las clases que se especializan en un conjunto de funcionalidades que representan el negocio sobre una o varias entidades. Son las únicas clases en la aplicación que tendrán lógica de

negocio, mientras que las fachadas se limitarán solamente a agrupar las funcionalidades para ser expuestas a capas superiores.

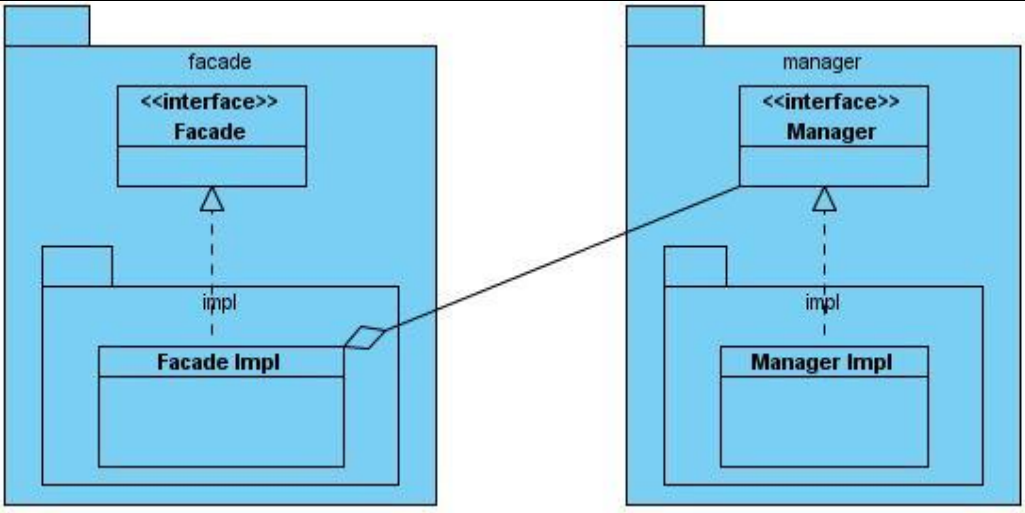


Figura 7 Capa Lógica de Negocio.

Las funcionalidades que presenta esta capa son:

- **Lógica de negocio específica de procesos de negocios:** En esta definición de arquitectura los objetos de dominio no presentan ningún tipo de lógica de negocio, sino que esta responsabilidad recae sobre los objetos de negocio, permitiendo usar a los objetos de dominio como objetos de transferencia que se mueven entre las capas arquitectónicas de la aplicación.

- **Puntos de entrada muy bien definidos para las operaciones de negocio implementadas:** Los objetos de negocio brindan las interfaces usadas por la capa de presentación.

- **Control de transacciones:** Las políticas transaccionales de la aplicación serán planteadas sobre los objetos de negocio.

- **Ejecución de restricciones de seguridad:** Las restricciones de seguridad en esta capa se realizarán en los puntos de entradas a la capa media, es decir en los objetos de negocio.

- **Ejecución de la auditoría:** Sobre esta capa se llevará a cabo la auditoría sobre los métodos de negocio.

3.2.3 Capa de Presentación

La Capa de Presentación descansa sobre una capa de servicios de negocio. Esto significa que dentro del SIGEP esta capa no contiene lógica del negocio alguna, sino que en ella residen la definición de las peticiones que el usuario puede realizar sobre la aplicación y los controladores que manejan el flujo web configurados a través de los ficheros del Framework Spring, y la comunicación con las interfaces de la Capa de Negocio.

La Capa de Presentación, basada en Spring Framework, se compone de tres tipos de objetos fundamentales:

- **Modelo:** son objetos contenedores de datos y los encargados de mostrar los datos resultantes de una petición a la Capa Lógica del Negocio.
- **Vistas:** son las que muestran al usuario el modelo en respuesta de la petición hecha al sistema. Las más comúnmente usadas dentro del SIGEP se encuentran en forma de objetos HTML, XML y PDF.
- **Controladores:** son los encargados de manejar las entradas de datos del usuario, invocar las funcionalidades requeridas dentro de la Lógica del Negocio y devolver el modelo para su presentación en las vistas.

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.

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

Con el fin de lograr que la interfaz gráfica del sistema tuviera una apariencia de aplicación de escritorio, entre otros motivos, se utiliza la biblioteca JavaScript Dojo. Entre los elementos más utilizados en el SIGEP se encuentran las tablas, para las cuales se define e implementa un controlador web encargado de proporcionar los datos específicos que estas muestran. Cada controlador utilizado para la gestión de una tabla hereda del controlador común *AbstractLoadTableController* y redefine el método *getJSONData.*

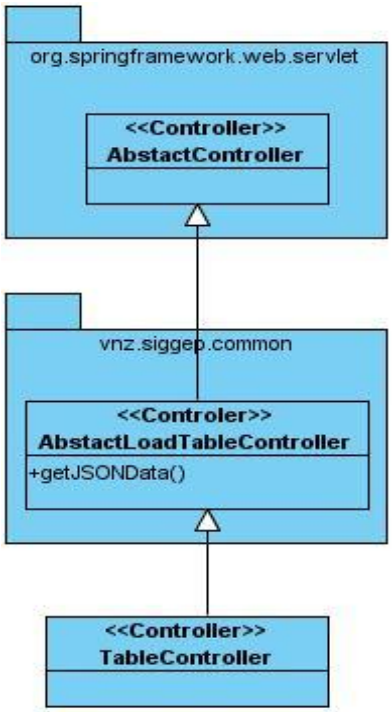


Figura 8 Capa de Presentación

3.2.4 Estructura del Sistema

En ArBaWeb se propone como unidad de organización: subsistemas y módulos. Iósev Pérez y Luis A Pimentel definen subsistema como: “un conjunto de funcionalidades del sistema que tienen características muy propias y que a su vez están subdivididas en módulos” y módulo es definido como: “un conjunto de funciones que debe realizar el sistema, las cuales son agrupadas por tener características muy similares y se definen en la etapa de diseño”. Un módulo está compuesto por una estructura de paquetes que contribuye a la separación por capas de la aplicación de forma lógica y física. (16)

Subsistemas

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

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

Los subsistemas de negocio son aquellos identificados a partir de la captura de requisitos y contienen la solución a los requisitos funcionales y no funcionales de procesos o áreas de procesos dentro del sistema

penitenciario venezolano, por ejemplo: los subsistemas Control Penal, Seguridad y Custodia, Tratamiento y Salud Integral.
Los subsistemas de soporte son aquellos que cubren funcionalidades como respuesta a requisitos no funcionales esperados del SIGEP; por ejemplo, el subsistema Administración, que se encarga de administrar la aplicación.

Estructura de un módulo

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

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

dao: Se encuentran las interfaces DAOs.

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

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

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

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

exception: Se encuentran las excepciones y las clases de utilidad para las validaciones.

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

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

managers: Se encuentran las interfaces de los managers

managers.impl: Se encuentran las implementaciones de los managers.

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

web: Se encuentran los controladores de la Capa de Presentación.

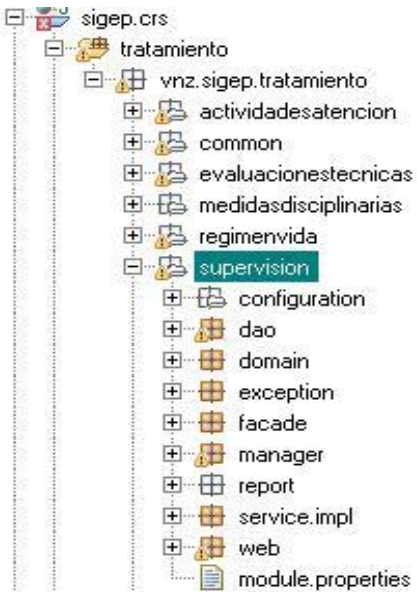


Figura 9 Estructura de un módulo

Mecanismos de colaboración

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

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

3.3 Modelo Lógico de Datos

El Diagrama del Modelo Lógico de Datos o Diagrama de Clases Persistentes, muestra las clases capaces de mantener su valor en el espacio y en el tiempo. El Diagrama de Clase se puede usar para modelar la estructura lógica de la base de datos, con clases representando tablas, y atributos de clase representando columnas. Las clases persistentes y sus atributos hacen referencia directamente a las entidades lógicas y a sus atributos.

Cuando se define correctamente el Modelo Lógico, se hace mucho menos engorroso llegar al Modelo de Datos o Modelo Físico, como también se le denomina en la metodología RUP. Se plantea que el Modelo de Datos representa la estructura o descripción física de las tablas de la base de datos y es obtenido a partir del Diagrama de Clases Persistentes.

Ver Anexo 2 Diagrama de Clases Persistentes

3.4 Modelo Físico de Datos

A partir del Modelo Lógico, donde se reflejan las clases persistentes, los nomencladores utilizados, así como sus relaciones, se obtuvo como resultado el Modelo de Datos del módulo. El Diagrama de Clases Persistentes obtenido en la actividad precedente provee al Modelo de Datos de una aproximación inicial de la estructura que tendrá la base de datos. Esta estructura garantizará que la gestión de la información ocurra de manera que se cumplan las restricciones necesarias. El Modelo de Datos del módulo Supervisión incluye las entidades persistentes, así como los nomencladores utilizados. El Modelo de Datos evoluciona a la par de las actividades de diseño, ya que el desarrollo se realiza de manera iterativa e incremental, cambiando en ciertas fases algunos elementos como procedimientos almacenados, funciones del gestor de base de datos, etc.
Este Modelo de Datos está destinado a garantizar un rendimiento óptimo en el acceso a datos y un mínimo acoplamiento al sistema utilizado para gestionar la base de datos.

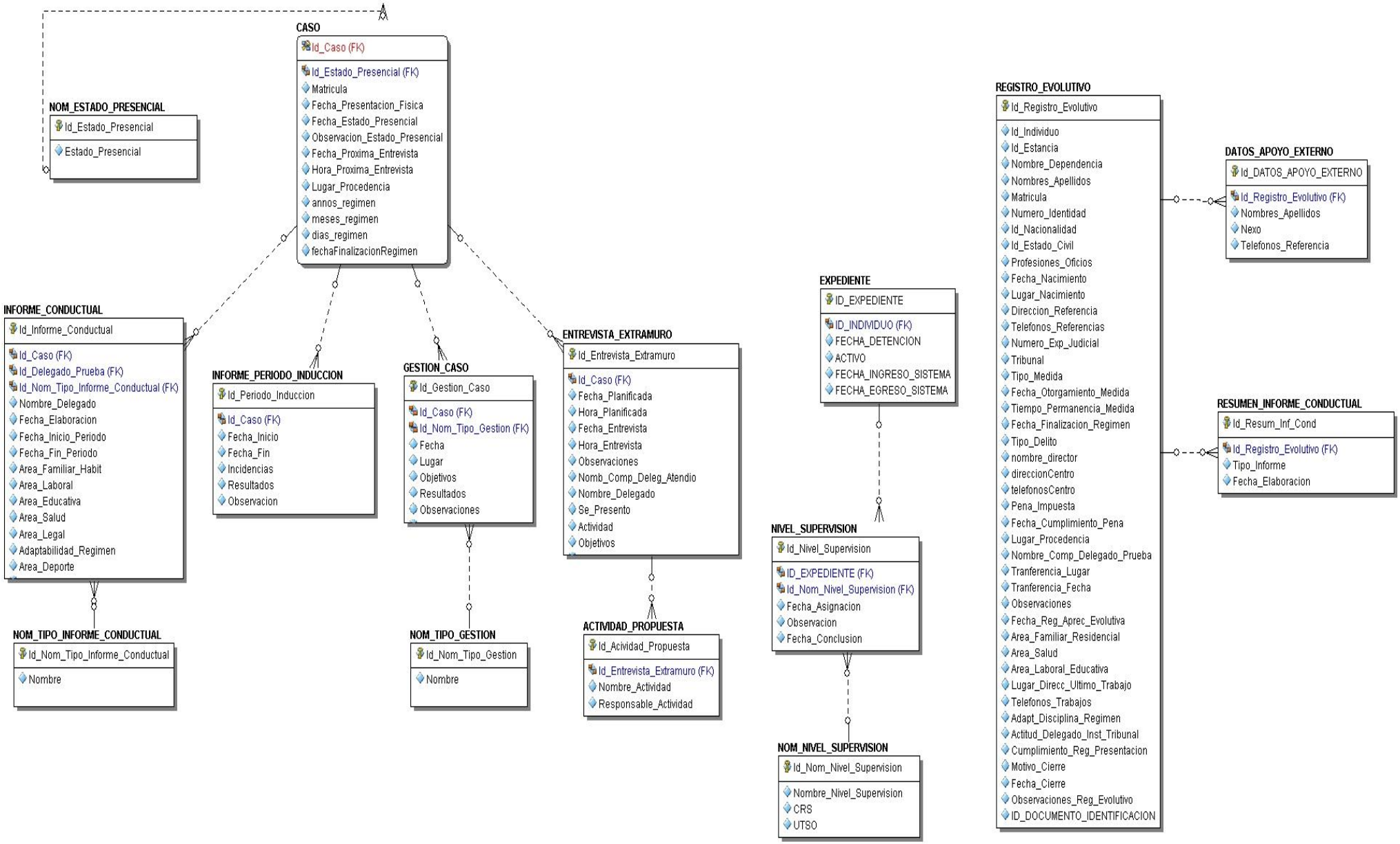


Figura 10 Modelo de datos.

3.5 Patrones de Diseño

Un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos; en teoría indica la manera de utilizarlo en circunstancias diversas. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas. (17)

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Los patrones que se usaron en el diseño del módulo Supervisión fueron los siguientes:

3.5.1 Data Access Object (DAO)

El propósito del patrón DAO es abstraer y encapsular todos los accesos a la fuente de datos. Con esto se obtiene varias ventajas:

1) Se tiene un paliativo[12] al problema del diferencial de impedancia (transparencia).

2) Se baja el nivel de acoplamiento entre clases, reduciendo la complejidad de realizar cambios.

3) Se aísla las conexiones a la fuente de datos en una capa fácilmente identificable y mantenible.

Su principal ventaja es que separa en una capa aparte todo el acceso a datos y abstrae la comunicación con la fuente de datos, esto hace que sea mucho más fácil una migración de fuente de datos en caso de ser necesaria.

3.5.2 Fachada (Facade)

Facade o fachada es un patrón de diseño que sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas.

Facade puede hacer una biblioteca de software más fácil de usar y entender, ya que implementa métodos convenientes para tareas comunes; hacer el código que usa la biblioteca más legible, por la misma razón;

[12] Paliativo: “Del verbo paliar, que mitiga, suaviza o atenúa”.

puede reducir la dependencia de código externo en los trabajos internos de una biblioteca, ya que la mayoría del código lo usa Facade, permitiendo así más flexibilidad en el desarrollo de sistemas.
Este patrón es utilizado en SIGEP para separar las capas, lo que hace posible la independencia entre ellas, por lo que un cambio en la capa inferior seria transparente para la capa inferior inmediata.

3.5.3 Controlador (Controller)

Una clase que tenga la responsabilidad de controlar el flujo de eventos de un sistema se denomina controlador. Los controladores delegan las tareas en otras clases generalmente de la Capa de Negocio. Aporta al SIGEP la ventaja de que toda la gestión de peticiones se centraliza, lo que hace posible mayor agilidad en la identificación de errores y en la corrección de los mismos. Además aumenta el mantenimiento y la reusabilidad del código.

3.5.4 Modelo-Vista-Controlador (MVC)

Es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El estilo de llamada y retorno MVC (según CMU), se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de Negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista. Este patrón hace posible que un cambio en la vista no implica un cambio del modelo.

3.6 Diseño de las funcionalidades del módulo Supervisión

Para el desarrollo del módulo Supervisión se tuvieron en cuenta los procesos identificados durante el levantamiento de requisitos. Se identificaron dos procesos fundamentales que son: Entrevistas, y Nivel de Supervisión.
Las entrevistas son los encuentros planificados que tiene el Delegado de Prueba con el individuo y como resultado de cada una se llena una hoja cronológica. El control de entrevistas muestra el listado de todas las entrevistas que se le han realizado al individuo. El sistema permite registrar las hojas cronológicas especificando siempre si el individuo se presentó o no a la entrevista. Se registran además la fecha y la hora del próximo encuentro del individuo con su Delegado de Prueba. De cada Hoja Cronológica registrada se pueden consultar los detalles, modificar y eliminar la información asociada a la misma. El control de entrevistas podrá ser generado para imprimirse.

El Nivel de Supervisión determina la frecuencia de presentaciones que realizará el penado sometido a una medida restrictiva de libertad ante el Delegado de Prueba asignado. Se clasifica en: Mínimo, Medio, Máximo y Especial.

Dentro de cada proceso se ubican diferentes casos de uso y cada uno de ellos contiene a su vez distintas funcionalidades, que en general son a las que deben responder el diseño y la implementación del módulo Supervisión.

A continuación se muestra como queda especificado por casos de uso y funcionalidades los procesos identificados.

Proceso Entrevista

CUS1: Gestionar entrevistas

1. Eliminar Entrevista
2. Registrar Entrevista
3. Modificar Entrevista
4. Mostrar Detalles de Entrevista

CUS2: Registrar Fecha de Próxima Entrevista

CUS3: Generar Control de Entrevista

Proceso Nivel de Supervisión

CUS4: Registrar Nivel de Supervisión

CUS5: Mostrar Histórico de Niveles de Supervisión

A continuación por cada funcionalidad definida se describen los elementos más significativos, así como el Diagrama de Clases del Diseño del caso de uso Gestionar Entrevista, en representación de los correspondientes a las restantes funcionalidades.

3.6.1 Gestionar Entrevista

Esta funcionalidad permite registrar, modificar, eliminar y consultar los datos de una entrevista. De cada entrevista se registra si el supervisado se presentó o no, en caso que se presente, se registra la información de la Hoja Cronológica. En caso de no presentación se registran las observaciones y la actividad planificada. Para registrar la información asociada a una entrevista de un caso, el individuo debe tener designado un Delegado de Prueba.

A continuación se representa el diagrama de clases del diseño correspondiente al CUS Gestionar Entrevista, dividido en tres capas como cumplimiento de la arquitectura planteada en el SIGEP.

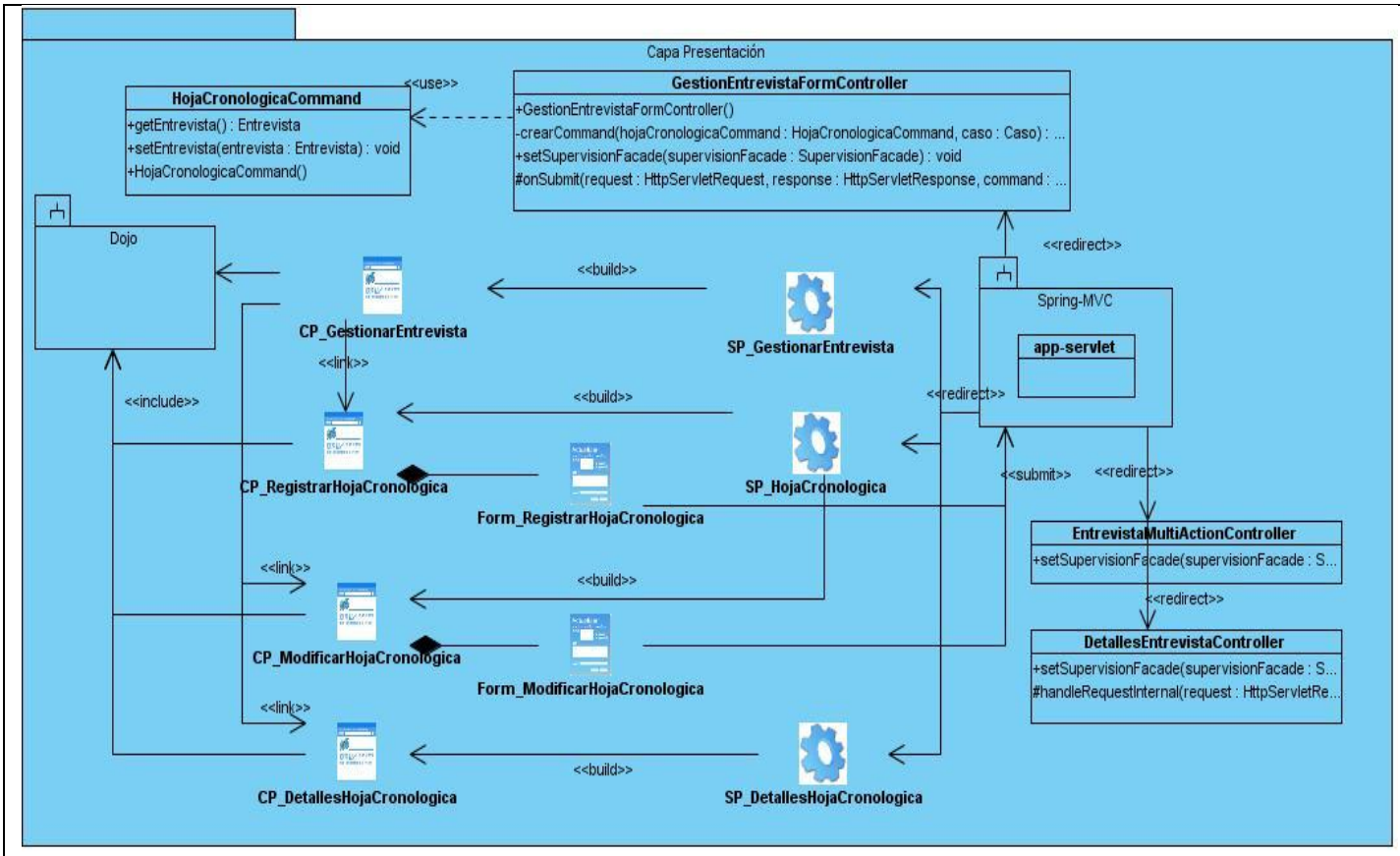


Figura 11 Capa Presentación

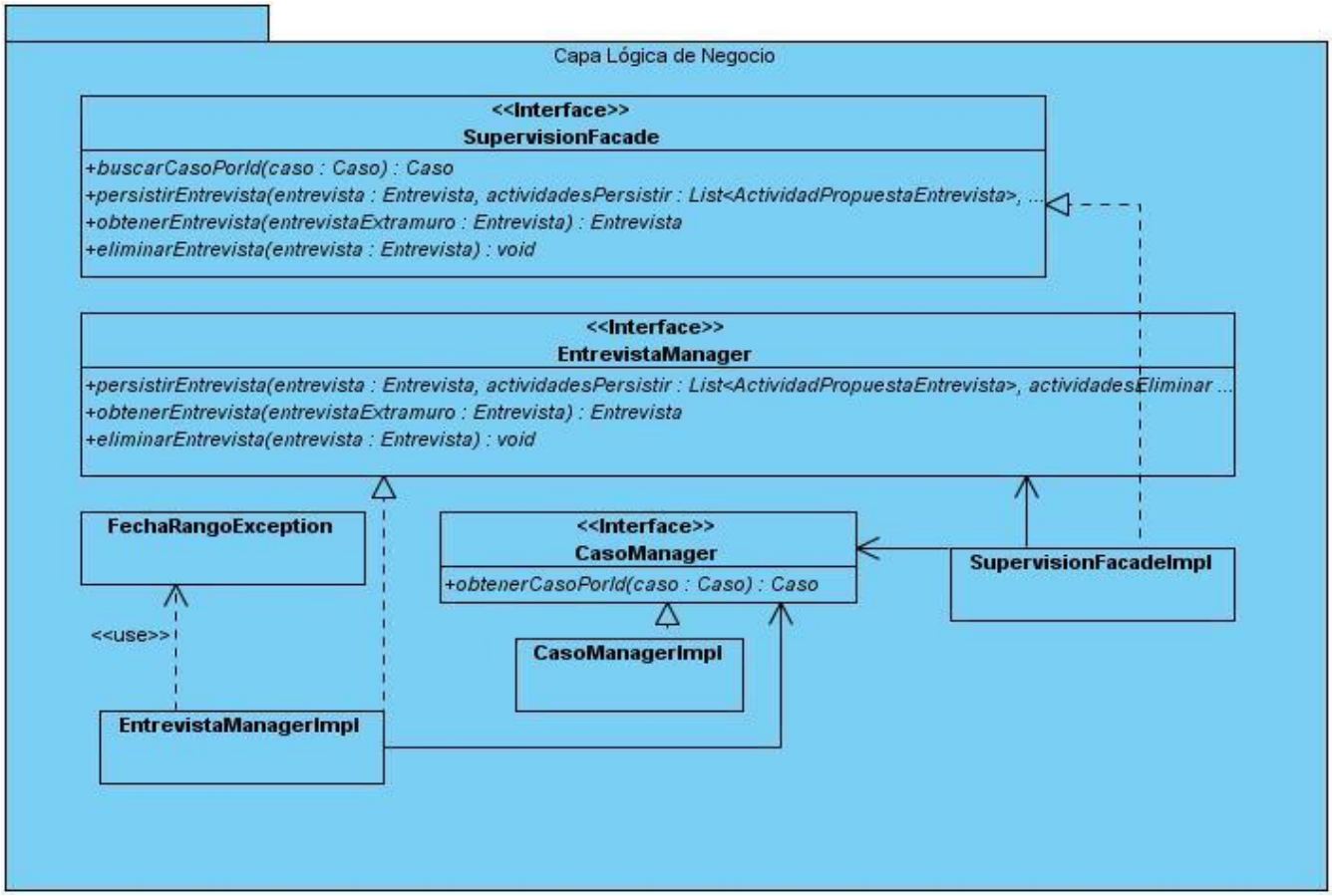


Figura 12 Capa Lógica de Negocio

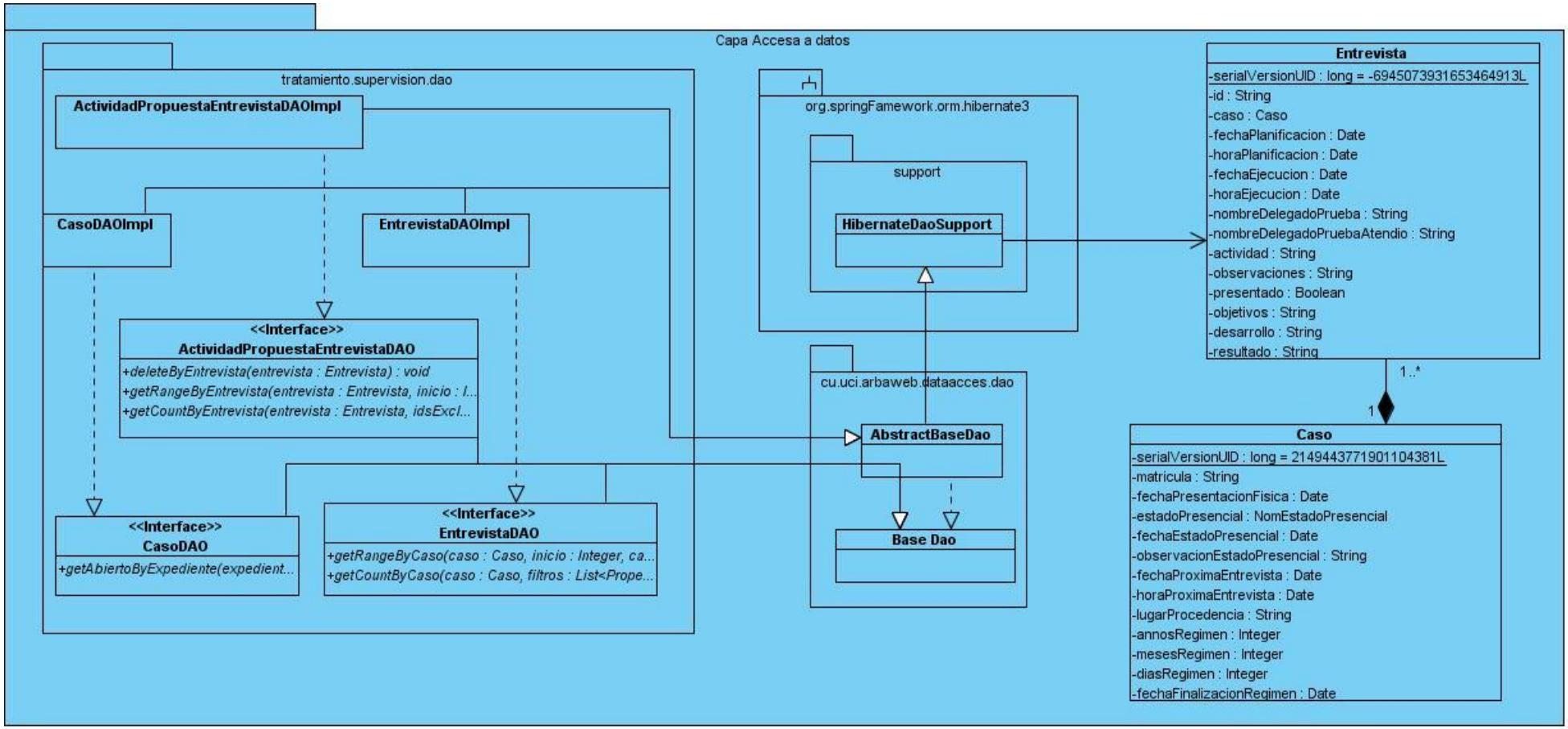

Capa Accesa a datos
tratamiento.supervision.dao
ActividadPropuestaEntrevistaDAOImpl
CasoDAOImpl
EntrevistaDAOImpl
<<Interface>>
ActividadPropuestaEntrevistaDAO
+deleteByEntrevista(entrevista : Entrevista) : void
+getRangeByEntrevista(entrevista : Entrevista, inicio : I...
+getCountByEntrevista(entrevista : Entrevista, idsExcl...
<<Interface>>
CasoDAO
+getAbiertoByExpediente(expedient...
<<Interface>>
EntrevistaDAO
+getRangeByCaso(caso : Caso, inicio : Integer, ca...
+getCountByCaso(caso : Caso, filtros : List<Prope...
org.springFamework.orm.hibernate3
support
HibernateDaoSupport
cu.uci.arbaweb.dataacces.dao
AbstractBaseDao
Base Dao
Entrevista
-serialVersionUID : long = -6945073931653464913L
-id : String
-caso : Caso
-fechaPlanificacion : Date
-horaPlanificacion : Date
-fechaEjecucion : Date
-horaEjecucion : Date
-nombreDelegadoPrueba : String
-nombreDelegadoPruebaAtendio : String
-actividad : String
-observaciones : String
-presentado : Boolean
-objetivos : String
-desarrollo : String
-resultado : String
1..*
1
Caso
-serialVersionUID : long = 2149443771901104381L
-matricula : String
-fechaPresentacionFisica : Date
-estadoPresencial : NomEstadoPresencial
-fechaEstadoPresencial : Date
-observacionEstadoPresencial : String
-fechaProximaEntrevista : Date
-horaProximaEntrevista : Date
-lugarProcedencia : String
-annosRegimen : Integer
-mesesRegimen : Integer
-diasRegimen : Integer
-fechaFinalizacionRegimen : Date

Figura 13 Capa Acceso a Datos

3.6.2 Mostrar Detalles de Entrevista

Esta funcionalidad ofrece la posibilidad de consultar los detalles de una entrevista que se seleccionó previamente en el listado de entrevistas del individuo.

3.6.3 Registrar Fecha de Próxima Entrevista

Funcionalidad que permite registrar la fecha y la hora de la próxima entrevista del individuo con el Delegado de Prueba. La fecha de la próxima entrevista debe ser mayor o igual a la fecha actual.

3.6.4 Generar Control de Entrevista

Permite obtener un informe de las entrevistas en formato imprimible.

3.6.5 Registrar Nivel de Supervisión

El individuo siempre tendrá un nivel de supervisión y partiendo de ello también se le podrá registrar un nuevo nivel de supervisión. Este nivel de supervisión es otorgado por el Delegado de Prueba y puede ser modificado como resultado de una entrevista realizada. Los niveles de supervisión en CRS son: “Mínimo”, “Medio”, “Máximo”. En UTSO: “Mínimo”, “Medio”, “Máximo” y “Especial”.

3.6.6 Consultar Histórico de Niveles de Supervisión

Esta funcionalidad permite consultar todos los niveles de supervisión por los que ha pasado el individuo.

3.7 Conclusiones parciales

En el capítulo se hizo referencia y se describieron los elementos fundamentales del diseño del módulo Supervisión, siguiendo las regulaciones establecidas para el flujo de trabajo Análisis y Diseño que propone la metodología de desarrollo RUP y cumpliendo además con los requisitos definidos para el módulo, ajustados a las definiciones arquitectónicas del SIGEP.

Los Diagramas de Clases del Diseño de las funcionalidades definidas se encuentran estructurados en tres capas, cumpliendo así con la arquitectura planteada por el SIGEP. En las mismas se utilizaron patrones de diseño para promover la flexibilidad y facilitar la capacidad de extensión de sus respectivas implementaciones.

Capítulo 4 Implementación y Pruebas

4.1 Introducción

En el presente capítulo se analizarán los principales artefactos del flujo de implementación como el Modelo de Implementación y en especial el Diagrama de Componente. Se muestran algunos ejemplos de la implementación de las funcionalidades más significativas para el módulo Supervisión, y de las tres capas. También se desarrolla el Flujo de Trabajo Prueba, para medir la calidad del producto y el grado en el que cumple con los requisitos previamente identificados.

4.2 Modelo de Implementación

Un Modelo de Implementación incluye suficiente información para construir el sistema. Debe incluir, no solamente la semántica lógica del sistema y los algoritmos, las estructuras de datos y los mecanismos que aseguran su funcionamiento apropiado, sino también las decisiones de organización sobre los artefactos del sistema que son necesarios, permitiendo así el trabajo cooperativo de las personas y el procesamiento por parte de las herramientas.

4.2.1 Diagrama de Componentes

Los Diagramas de Componentes describen los elementos físicos del sistema y sus relaciones, muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc.
Más adelante, en las figuras 14,15 y 16 se representa el Diagrama de Componente del caso de uso Gestionar Entrevista, en correspondencia con el Diagrama de Clases del Diseño y la arquitectura del SIGEP.

4.2.2 Diagrama de Despliegue

El Diagrama de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Además muestran la configuración en funcionamiento del sistema incluyendo su software y su hardware. Para cada componente de un diagrama es necesario documentar las características técnicas requeridas, el tráfico de red, el tiempo de respuesta, etc.
Ver Anexo 3 para los Diagramas de Despliegue de CRS y UTSO.

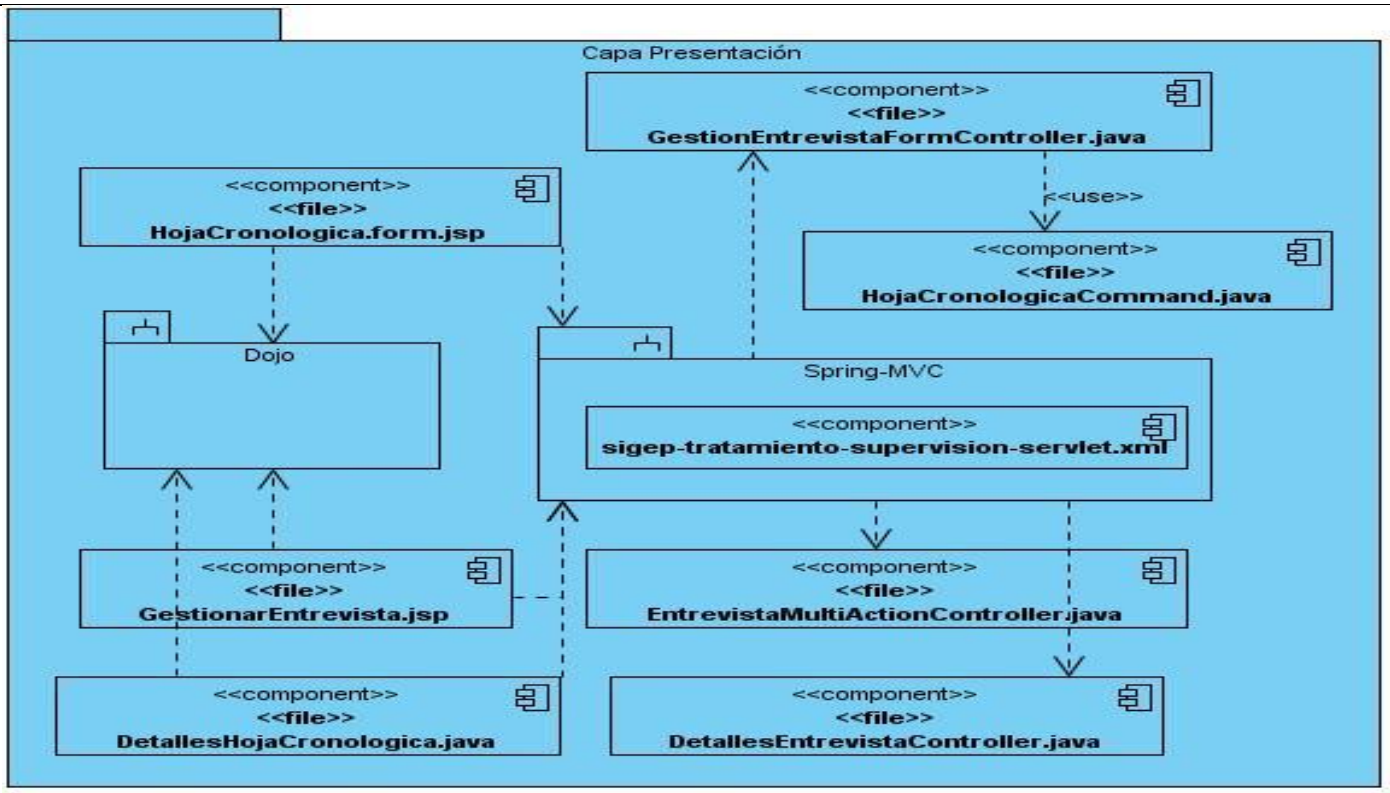


Figura 14 Capa Presentación

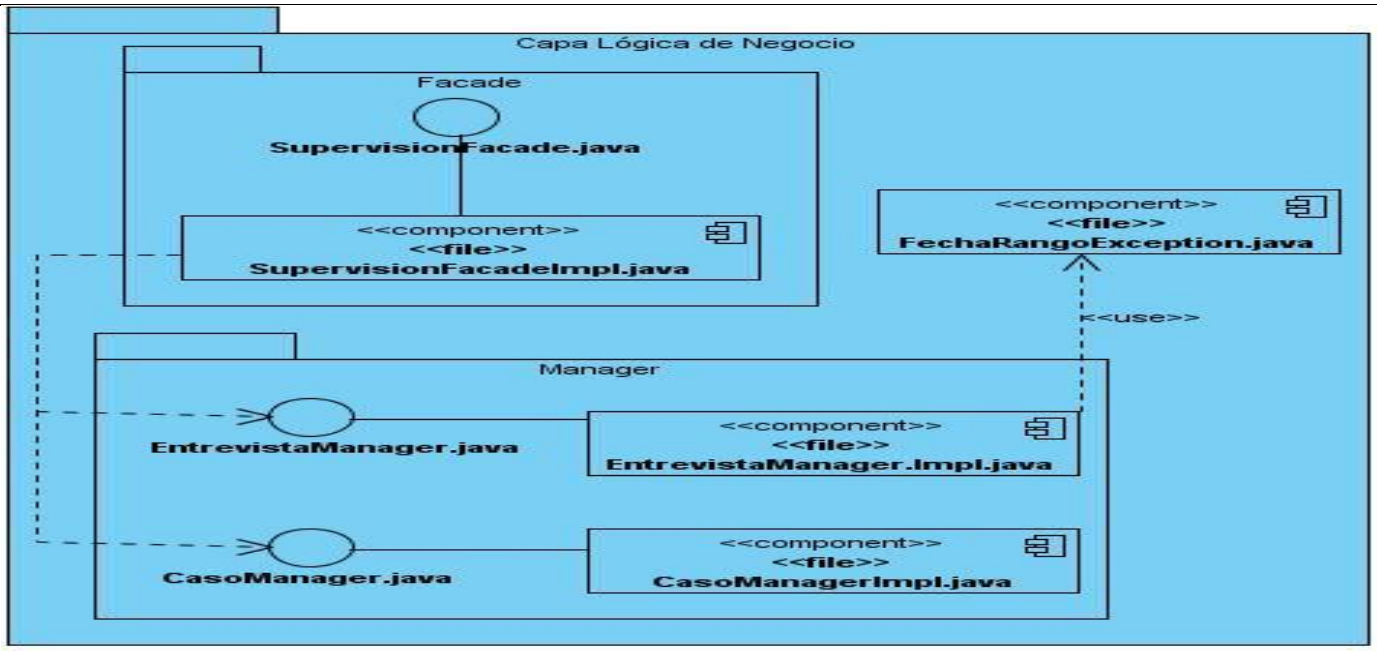


Figura 15 Capa Lógica de Negocio

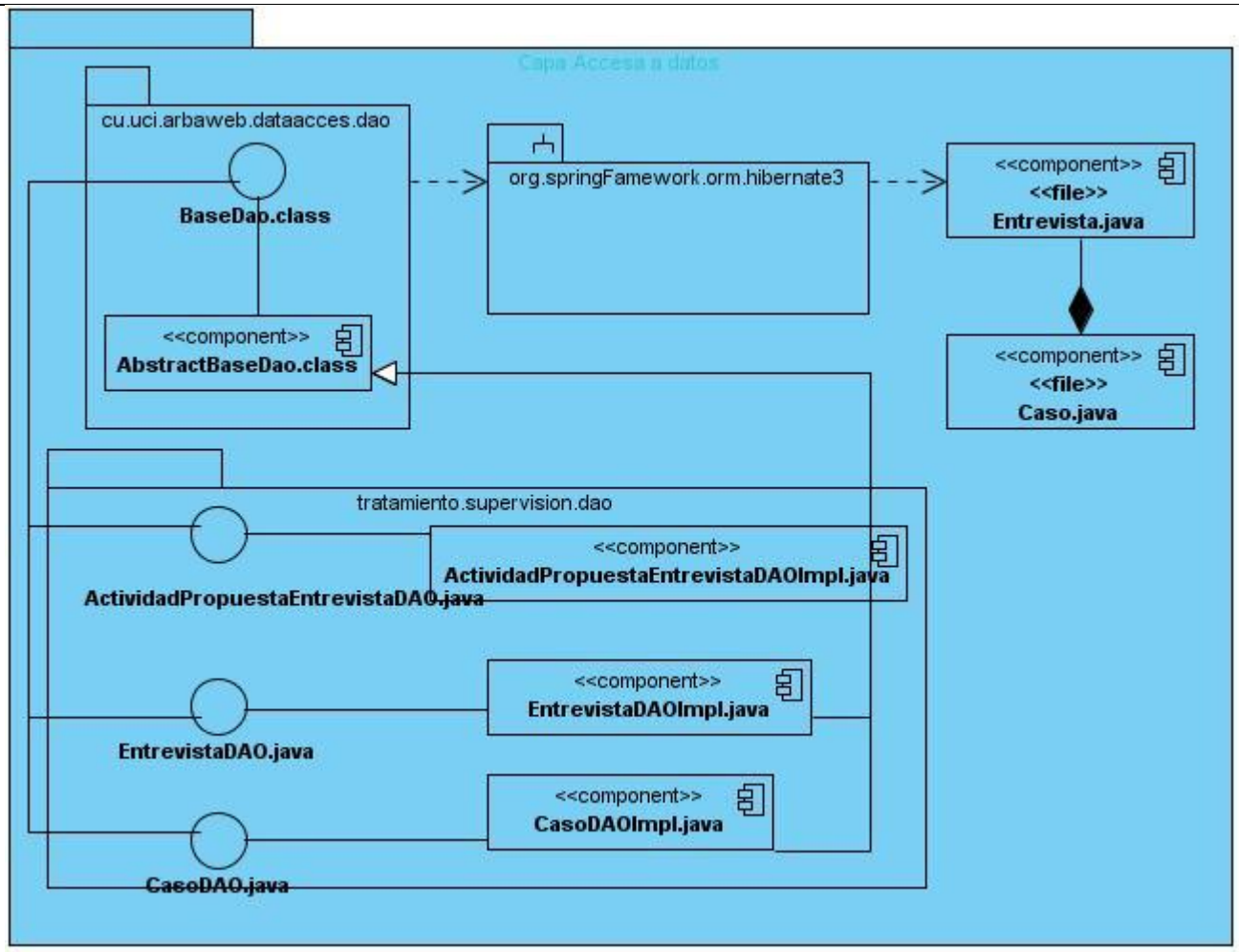


Figura 16 Capa de Acceso a Datos

4.3 Implementación de las Entidades del Dominio

La implementación de las Entidades del Dominio por lo general es muy básica, ya que estas suelen tener muy poco comportamiento. En ella además de los métodos comunes a todas las entidades, no suele hacerse énfasis en la programación de funcionalidades más especializadas que serán características de las capas a las que pertenezca cada entidad. 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.

4.4 Implementación de la Capa de Acceso a Datos

En la implementación de la Capa de Acceso a Datos se destacan dos tareas fundamentales: la creación de los ficheros de mapeo de Hibernate (hbm.xml) y la implementación de las funcionalidades de los DAOs.
El framework Hibernate permite asociar una tabla de la base de datos a un objeto Java, relacionando cada campo de la tabla con un atributo del objeto. Esta configuración se hace en ficheros XML. En la creación de estos ficheros juega un papel importante el plug-in de Eclipse Hibernate Tools, acelerando esta tarea al brindar la generación de código en algunos momentos durante la confección de estos ficheros.

```
hibernate-mapping package="vnz.sigep.tratamiento.supervision.domain">
      <class name="Entrevista" table="ENTREVISTA_EXTRAMURO">
            <id name="id" type="string">
                  <column name="ID_ENTREVISTA_EXTRAMURO" length="20"></column>
                  <generator

      class="vnz.sigep.common.global.util.dataaccess.id.PrefixPersistentGenerator">
                        <param name="sequence">SEQ_ENTREVISTA_EXTRAMURO</param>
                        <param name="maxlo">1000</param>
                  </generator>
            </id>
            <many-to-one name="caso" class="vnz.sigep.common.global.domain.Caso"
                  fetch="select">
                  <column name="ID_CASO" length="20"></column>
            </many-to-one>
            <property name="fechaPlanificacion" type="date">
                  <column name="FECHA_PLANIFICADA" not-null="true"></column>
```

Figura 17 Mapeo de Hibernate

En el código anterior se observan etiquetas y algunos de los atributos que estas contienen. El atributo **package** indica cuál es el paquete donde se encuentra la clase que va a ser mapeada. En la etiqueta **class** se especifican los nombres de la clase y de la tabla de la base de datos. Dentro de la etiqueta **id** se encuentra el nombre del atributo de la clase que va a almacenar el identificador, el tipo de dato, la columna de la tabla con la cual se hace corresponder y el **generador del identificador** (generator), en este caso, es la clase **PrefixPersistentGenerator**.
Por cada atributo hay que generar una etiqueta **property**. La etiqueta **many-to-one** significa que hay una relación de uno a muchos, donde se especifica el nombre del atributo, la clase de este atributo (el tipo de dato) y la columna de la tabla con la cual se mapea.

El uso de Hibernate en la Implementación de los DAOs permite alcanzar mayor productividad y flexibilidad en la implementación.
Para la implementación de las funcionalidades de los Objetos de Acceso a Datos se usa como primera opción los APIs Criteria y Example del Framework Hibernate, que permiten la creación de consultas al gestor de base de datos de gran complejidad en la que intervienen varias tablas, sin la necesidad de introducir código en lenguaje SQL. Cuando estas APIs no son suficientes para cumplir con las funcionalidades se utiliza también el lenguaje de consultas de Hibernate (HQL). Con la utilización de este lenguaje se obtiene independencia de la aplicación con respecto al gestor de base de datos utilizado, ya que las consultas se realizan de manera transparente, gestionadas directamente por el API provisto por Hibernate para ello. Los DAOs implementados se configuran en el fichero sigep-tratamiento-supervision-dataaccesscontext.xml.

4.5 Implementación de la Capa Lógica de Negocio

Durante esta actividad se implementan los métodos definidos para cada una de las interfaces de los managers y la fachada, los cuales deben ser ajustados a las funcionalidades previstas.

Cuando se implementa un método se debe verificar la integridad de los datos de entrada y de salida, y ante cualquier eventualidad se debe informar a la capa de interfaz de usuario a través de las excepciones definidas, las cuales serán capturadas por la Capa de Presentación y mostradas al usuario de una forma adecuada.

La fachada y los managers implementados se configuran en el fichero de configuración del contexto de Spring sigep-tratamiento-supervision-business-context.xml. En este fichero se inician los objetos del negocio en el contexto de Spring, se inyectan a cada uno de los managers a través de la inyección de dependencia de Spring los DAOs necesarios para realizar sus funcionalidades, así como la inyección a la fachada de los todos los manager que debe conocer para exponer las funcionalidades del módulo.

Los managers son los encargados de gestionar la información que fluya desde y hacia la capa de interfaz de usuario, así como de lanzar y capturar eventos del sistema cuando sea necesario.

La implementación de la interfaz de la fachada provee una simplificación de la relación entre la lógica del negocio y la interfaz de usuario, reduciendo al mínimo los objetos con los que esta última deberá interactuar. En la fachada no existe lógica de negocio, solamente se tendrá una representación de las

funcionalidades que se encuentran implementadas en los managers correspondientes, brindándole de este modo a la Capa de Presentación un punto de mínimo acceso a la información en cualquier momento.

A continuación se muestran ejemplos de implementación del método persitirEntrevista de la fachada y el manager con el mismo nombre.

```
public void persistirEntrevista(Entrevista entrevista,
                List<ActividadPropuestaEntrevista> actividadesPersistir,
                List<ActividadPropuestaEntrevista> actividadesEliminar)
                throws Exception {
            entrevistaManager.persistirEntrevista(entrevista, actividadesPersistir,
                        actividadesEliminar);   }
```

Figura 18 Ejemplo de la implementación de la fachada

```
public void persistirEntrevista(Entrevista entrevista,
                List<ActividadPropuestaEntrevista> actividadesPersistir,
                List<ActividadPropuestaEntrevista> actividadesEliminar)
                throws Exception {
            InputAssert.notNull(entrevista, "entrevista es requerido");
            InputAssert.notNull(entrevista.getCaso(), "el caso es requerido");
            InputAssert.hasText(entrevista.getCaso().getId(), "El id del caso es
requerido");

            Date fechaIngreso =
 casoManager.obtenerCasoPorId(entrevista.getCaso()).getFechaIngreso();
            if(entrevista.getFechaPlanificacion().before(fechaIngreso) ||
                        entrevista.getFechaEjecucion().before(fechaIngreso))
                throw new FechaRangoException("No puede registrar ninguna fecha
anterior a la fecha de ingreso del Caso");
                        // persistiendo
            entrevistaDAO.persist(entrevista);

            // persistiendo actividades
            if (actividadesPersistir != null) {
                for     (Iterator<ActividadPropuestaEntrevista>     iterator     =
actividadesPersistir
                            .iterator(); iterator.hasNext();) {
                        ActividadPropuestaEntrevista actividadPropuesta = iterator
                                    .next();
                        // se persisten actividades (SOLO) para esta entrevista
                        actividadPropuesta.setEntrevista(entrevista);
                        actividadPropuestaEntrevistaDAO.persist(actividadPropuesta);
                }
            }
            if  (StringUtils.hasText(entrevista.getId())  &&  actividadesEliminar  !=
null) {
                        for   (Iterator<ActividadPropuestaEntrevista>   iterator   =
```

```
actividadesEliminar
                                    .iterator(); iterator.hasNext();) {
                              ActividadPropuestaEntrevista    actividadPropuesta    =
iterator
                                          .next();
                              ActividadPropuestaEntrevista          apeBD          =
actividadPropuestaEntrevistaDAO

      .findById(actividadPropuesta.getId(), false);
                              actividadPropuestaEntrevistaDAO
                                          .delete(apeBD);
                        }
            }
      }
```

Figura 19 Ejemplo de implementación de un manager

4.5.1 Transacciones a Nivel de Negocio

Uno de los aspectos a tener en cuenta a la hora de desarrollar un sistema es que el mismo mantenga la integridad de la información. Uno de los procedimientos utilizados para cumplir con este requisito es el empleo de transacciones. Las mismas proporcionan cuatro garantías muy importantes: la consistencia, atomicidad, aislamiento y durabilidad.

En el SIGEP las transacciones se realizan a nivel de negocio, cada método de negocio que implique operaciones de escritura sobre la base de datos, se ejecuta como una transacción; por tanto si una de las operaciones falla, se cancelarán todas las demás.

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

La clase org.springframework.orm.hibernate3.HibernateTransactionManager une una sesión de Hibernate al hilo de ejecución e implementa los métodos getTransaction, commit y rollback definidos en la interface org.springframework.transaction.PlatformTransactionManager. A través de estos tres métodos se crea una transacción o se accede a la transacción en curso y se hace commit (aceptar) y rollback (fallar) a las transacciones. La instancia de HibernateTransactionManager requiere de una referencia al sessionFactory de Hibernate.

La Programación Orientada a Aspectos es uno de los pilares fundamentales de Spring. A continuación se muestran algunas definiciones importantes sobre la misma:

- **Aspect** (Aspecto) es una funcionalidad transversal que se va a implementar de forma modular y separada del resto del sistema como por ejemplo la aplicación de transacciones.

- **Join point** (Punto de Cruce o de Unión) es un punto de ejecución dentro del sistema donde un aspecto puede ser conectado, como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo. El código del aspecto será insertado en el flujo de ejecución de la aplicación para añadir su funcionalidad.
- **Advice** (Consejo) es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad. Se insertan en la aplicación en los Puntos de Cruce.
- **Pointcut** (Puntos de Corte) define los Consejos que se aplicarán a cada Punto de Cruce. Se especifica mediante Expresiones Regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.

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 asociado al manejo de transacciones en los métodos que contienen la lógica de negocio.

Con la etiqueta **<aop:advisor>** se declara el pointcut (punto en que se aplicará el aspecto). Con la etiqueta **<tx:advice>** se declara el aspecto específico, o sea, cómo se va a aplicar la transacción.

En la siguiente figura se muestra un ejemplo de transacción de negocio en el módulo Supervisión:

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

Figura 20 Aplicación de transacciones al objeto EntrevistaManagerImpl

Con esta configuración se logra que ante cualquier excepción que sea lanzada en la ejecución de los métodos de la clase EntrevistaManagerImpl se cancelen las operaciones, por tanto la base de datos

quedaría en el mismo estado en que estaba antes de iniciarse la transacción. De forma similar se configuran los restantes objetos de la capa de negocio.

La definición del punto donde se aplica la transacción está escrita utilizando la sintaxis de Aspectj (Lenguaje de programación orientado a aspectos, construido como una extensión del lenguaje Java). La expresión **execution** significa cuando el método sea ejecutado. La expresión entre paréntesis representa los métodos a los que se aplicará la transacción, en este caso: ****...* EntrevistaManagerImpl.*(...).**

El primer ***** significa cualquier tipo de retorno; la expresión ***...*** E**ntrevistaManagerImpl** significa cualquier clase cuyo nombre termine en EntrevistaManagerImpl y la expresión.***(...)** significa cualquier método con cualquier parámetro. Al no especificarse el nivel de aislamiento de la transacción mediante el atributo isolation de la etiqueta <tx:method>, se toma el nivel de aislamiento por defecto del medio de almacenamiento, que en el caso de Oracle es lectura confirmada **(read committed)** que evita la lectura de datos sucios. Con el atributo **rollback-for** se definen las excepciones que, en caso de ser lanzadas, se desea que hagan fallar (rollback) la transacción.

4.6 Implementación de la Capa de Presentación

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

Los controladores dentro del SIGEP implementan una interfaz genérica de Spring, en la que se definen funcionalidades comunes que estos deberán contener. Además serán los encargados de comunicar la Capa de Presentación con la Lógica de Negocio a través de la fachada que brinda esta última. Son también los responsables de realizar las validaciones de los datos de entrada en el lado del servidor y dar formato a los datos de salida.

Las páginas JSP construyen los documentos HTML que serán las páginas clientes que son mostradas al usuario a partir de la información recibida del controlador correspondiente. Por lo general cada página JSP construye una página cliente o, en ocasiones se juntan un número de ellas para formar una página cliente. Durante la implementación de las páginas JSP, se usan la tecnología JSTL (Java Standard Tag Library), conjunto de librerías de etiquetas simples y estándares muy útiles a la hora de realizar ciertas operaciones sobre los datos que son enviados junto con la página JSP desde el controlador y las etiquetas de Spring.

Para el diseño gráfico de las JSP usadas se utilizan las Hojas de Estilo en Cascada (Cascading Style Sheets, CSS) definidas para cada uno de los componentes dentro de SIGEP.

Aparejado a la implementación de las páginas JSP, se realiza la implementación de la lógica en el cliente. La misma se realizó con la utilización de JavaScript. Desde el lado del cliente se interactúa con la aplicación de manera asíncrona aprovechando las posibilidades que brindan AJAX y JSON con el envió de peticiones al servidor de una forma mucho más rápida. Se realizan las validaciones que sean necesarias y se programa el comportamiento de los componentes gráficos que se utilizan dentro de las páginas JSP, tales como: tablas, botones, pantallas emergentes de error o información al usuario y calendarios entre otros.

En la siguiente figura se muestra la implementación del método onSubmit del controlador GestionEntrevistaFormController.

Este método es el encargado de enviar los datos del formulario referente a la gestión de entrevista. Este formulario, cuenta con una tabla donde se encuentran todas las entrevistas realizadas hasta la fecha.

En el método se convierte cada uno de los elementos del arreglo JSON de la tabla de actividades nuevas a objetos actividades, las cuales serán añadidas a la lista de actividades a persistir o a eliminar.

```
protected ModelAndView onSubmit(HttpServletRequest request,
                  HttpServletResponse   response,   Object   command,   BindException
errors)
                  throws Exception {
            String error = "{error:0}";
            try {
                  Entrevista entrevista = new Entrevista();
                  HojaCronologicaCommand          hojaCronologicaCommand          =
(HojaCronologicaCommand) command;
entrevista = hojaCronologicaCommand.getEntrevista();
                  String idCasoInputId = request.getParameter("idCasoInputId");
                  JSONArray actividadesAddJSON = JSONArray.fromString(request
                              .getParameter("actividadesAdd"));
                  JSONArray actividadesEditJSON = JSONArray.fromString(request
                              .getParameter("actividadesEdit"));
                  JSONArray actividadesDelJSON = JSONArray.fromString(request
                              .getParameter("actividadesDel"));

                  //Se convierte cada elemento del arreglo JSON de actividades
nuevas a un objeto actividad y se añade a la lista de actividades a persistir
                  List<ActividadPropuestaEntrevista>     actividadesPersistir     =
convertirActividadesNuevas(
                              actividadesAddJSON);
                  //Se convierte cada elemento del arreglo JSON de actividades
```

```
modificadas a un objeto actividad y se añade a la lista de actividades a persistir
                  actividadesPersistir = convertirActividadesModificar(
                              actividadesPersistir, actividadesEditJSON);
                  //Se convierte cada elemento del arreglo JSON de actividades a
eliminar a un objeto actividad y se añade a la lista de actividades a eliminar
                  List<ActividadPropuestaEntrevista>      actividadesEliminar      =
convertirActividadesEliminar(
                              actividadesDelJSON);

                  Caso caso = new Caso();
                  caso.setId(idCasoInputId);
                  entrevista.setCaso(caso);
                  if (!StringUtils.hasText(entrevista.getId())) {
                        entrevista.setId(null);
                  }

      entrevista.setFechaEjecucion(hojaCronologicaCommand.getEntrevista()
                              .getFechaEjecucion());
                  entrevista.setHoraEjecucion(hojaCronologicaCommand.getEntrevista()
                              .getHoraEjecucion());
                  entrevista.setFechaPlanificacion(hojaCronologicaCommand
                              .getEntrevista().getFechaPlanificacion());
                  entrevista.setHoraPlanificacion(hojaCronologicaCommand
                              .getEntrevista().getHoraPlanificacion());

                  supervisionFacade.persistirEntrevista(entrevista,
                              actividadesPersistir, actividadesEliminar);
            } catch (FechaRangoException e) {
                  error = "{error:\"" + e.getMessage() + "\"}";
            } catch (Exception exception) {
                  exception.printStackTrace();
                  error = "{error:1}";
            }
            response.getWriter().write(error);
            return null;
      }
```

Figura 21 Implementación del controlador GestionEntrevistaFormController

4.7 Pruebas de software

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

Se puede definir prueba como: Una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. (18)

Las pruebas de software implican ejecutar una implementación del software con datos de prueba. Se examinan las salidas del software y su entorno operacional para comprobar que funcionan tal y como se requiere. Las pruebas son una técnica dinámica de verificación y validación. (14)

Las pruebas tienen dos objetivos fundamentales:

- Demostrar al desarrollador y al cliente que el software satisface sus requerimientos.
- Descubrir defectos en el software en que el comportamiento de este es incorrecto, no deseable o no cumple su especificación.

En la siguiente figura se muestra un modelo general del proceso de pruebas.

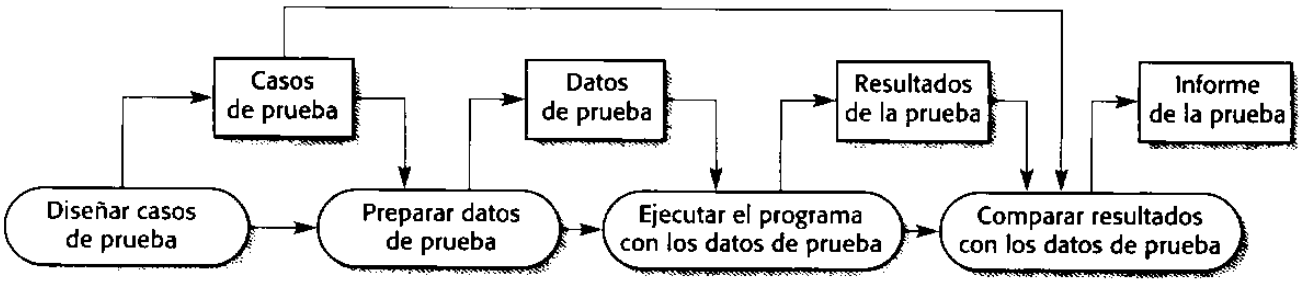


Figura 22 Modelo del proceso de pruebas del software

Los casos de prueba son especificaciones de las entradas para la prueba y la salida esperada del sistema, más una afirmación de lo que se está probando. Los datos de prueba son las entradas que han sido ideadas para probar el sistema. Los datos de prueba a veces pueden generarse automáticamente. La generación automática de casos de prueba es imposible. La salida de las pruebas sólo puede predecirse por personas que comprenden lo que debería hacer el sistema.

4.7.1 Herramientas para automatizar las pruebas

JUnit

Para la realización de pruebas unitarias se usa la herramienta JUnit**.** Esta es un conjunto de clases (*framework*) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

En la actualidad las herramientas de desarrollo como NetBeans y Eclipse cuentan con *plug-ins* que permiten que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se realice de manera automática, facilitando al programador enfocarse en la prueba y el resultado esperado, y dejando a la herramienta la creación de las clases que permiten coordinar las pruebas.

JMeter

Apache JMeter es una herramienta de carga diseñada para realizar Pruebas de Rendimiento y Pruebas Funcionales sobre Aplicaciones Web. (19)

A través de una batería de pruebas, **Jmeter** permite diagnosticar si la respuesta que ofrece el objeto sometido a estudio es la adecuada. Varias baterías de prueba pueden ejecutarse de forma simultánea gracias al carácter multihilo de la aplicación.

Utilizar **JMeter** en aplicaciones web para la comprobación de los recursos del sistema, supone una mayor efectividad en el proceso y en la fiabilidad de los resultados. (20)

4.7.2 Niveles de Prueba

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

- Prueba de Desarrollador
- Prueba Independiente
- Prueba de Unidad
- Prueba de Integración
- Prueba de Sistema
- Prueba de Aceptación

Para realizar las pruebas en el módulo Supervisión se desarrollarán las pruebas de unidad.
La prueba de unidad es la prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera.

Para que una prueba unitaria sea buena se deben cumplir los siguientes requisitos:

- **Automatizable:** no debería requerirse una intervención manual. Esto es especialmente útil para integración continua.
- **Completas:** deben cubrir la mayor cantidad de código.
- **Repetibles o Reutilizables:** no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.
- **Independientes:** la ejecución de una prueba no debe afectar la ejecución de otra.
- **Profesionales:** las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

Aunque estos requisitos no tienen que ser cumplidos al pie de la letra, se recomienda seguirlos o de lo contrario las pruebas pierden parte de su función.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

- **Fomentan el cambio:** Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (lo que se ha dado en llamar refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- **Simplifican la integración:** Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente. De esta manera se facilitan las pruebas de integración.
- **Documentan el código:** Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- **Separación de la interfaz y la implementación:** Dado que la única interacción entre los Casos de Prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro, a veces usando objetos mock (mock object) para simular el comportamiento de objetos complejos.
- **Los errores están más acotados y son más fáciles de localizar:** dado que existen pruebas unitarias que pueden desenmascararlos.

4.7.3 Técnicas de Prueba

Cada nivel de prueba engloba una técnica de prueba específica según los atributos de calidad que se deseen verificar con las pruebas al software.

Pruebas de Funcionalidad

Entre las técnicas de pruebas que se realizan en el sistema está la que evalúa la funcionalidad de éste. Según los parámetros de evaluación que abarca la técnica se pueden distinguir distintos tipos de pruebas:

- **Prueba Funcional:**

Objetivo: Asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

Metas:

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

Método de Prueba:

Caja Negra: Se refiere a las pruebas que se llevan a cabo sobre la interfaz del software, por lo que los Casos de Prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo fundamentalmente del sistema sin tener mucho en cuenta la estructura interna del software. (21)

Se ejecuta cada caso de uso, flujo de caso de uso, o función, usando datos válidos e inválidos, para verificar lo siguiente:

- Que se aplique apropiadamente cada regla de negocio.
- Que los resultados esperados ocurran cuando se usen datos válidos.
- Que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.

Caso de Prueba: Los Casos de Pruebas serían entonces un conjunto de entradas con datos de prueba, unas condiciones de ejecución, y unos resultados esperados cuyo propósito es identificar y comunicar las condiciones que se llevarán a cabo en la prueba. Los Casos de la Prueba son necesarios para verificar la aplicación exitosa y aceptable de los requisitos del producto (casos de uso). (22)

Los Casos de Prueba del módulo Supervisión se encuentran en el Expediente de proyecto del Sistema de Gestión Penitenciaria.

No Conformidades: La plantilla de No Conformidades recoge los errores que son detectados durante la revisión de la documentación del sistema. Se elabora un documento por cada revisión que se haga y se controlan a través de versiones según se vayan eliminando los errores, hasta que finalmente se hayan erradicado todos los defectos que posea el elemento que se prueba. Además de estar inmerso en la planilla de diseño de Casos de Prueba, estas No Conformidades se van registrando en un documento aparte para luego enviarlo al equipo de desarrolladores.

En la siguiente tabla se expone un resumen de los resultados de las No Conformidades obtenidos para contabilizar el total de No Conformidades del módulo Supervisión, distinguiendo de ellas cuáles proceden, cuáles no proceden y las que ya fueron resueltas.

Módulo	Total de No Conformidades	# de No Conformidades que proceden	# de No Conformidades que no proceden	# de No Conformidades resueltas
Supervisión	10	7	3	7

Tabla 1 Resumen de NC del proceso de liberación interno.

- **Pruebas de Seguridad**

Objetivo:

- Nivel de Seguridad de la Aplicación: Verificar que un actor solo pueda acceder a las funciones y datos que su usuario tiene permitido.
- Nivel de Seguridad del Sistema: Verificar que solo los actores con acceso al sistema y a la aplicación están habilitados para accederla.

Garantiza:

- Que los usuarios están restringidos a funciones específicas o su acceso está limitado únicamente a los datos que están autorizados a acceder.
- Que solo aquellos usuarios autorizados a acceder al sistema son capaces de ejecutar las funciones del sistema.

Técnica:

- Identificar cada tipo de usuario y las funciones y datos a los que se debe autorizar.
- Crear pruebas para cada tipo de usuario y verificar cada permiso, creando transacciones específicas para cada tipo de usuario.
- Modificar tipos de usuarios y volver a ejecutar las pruebas.

Criterio de Completitud:

Para cada tipo de usuario conocido, las funciones, datos apropiados y todas las transacciones funcionan como se esperaba.

Pruebas de Confiabilidad

La confiabilidad del software se refiere a la precisión con la que una aplicación proporciona, sin errores, los servicios que se establecieron en las especificaciones originales. (23)

- **Pruebas de Stress**

Objetivo:

Consisten en la simulación de grandes cargas de trabajo para observar de qué forma se comporta la aplicación ante situaciones de uso intenso.
Está enfocada a evaluar cómo el sistema responde bajo condiciones anormales (extrema sobrecarga, insuficiente memoria, servicios y hardware no disponible, recursos compartidos no disponible).

Prueba de Rendimiento

Las pruebas de rendimiento son las pruebas que se realizan, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. Están diseñadas para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

- **Pruebas de Carga**

Objetivo:

Usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante.
La variación en carga es simular la carga de trabajo promedio y con picos que ocurre dentro de tolerancias operacionales normales.
Para medir el rendimiento de la aplicación se utilizó la herramienta JMeter, que fue anteriormente explicada.

Simulación de 10 usuarios con un periodo de subida de 30 segundos

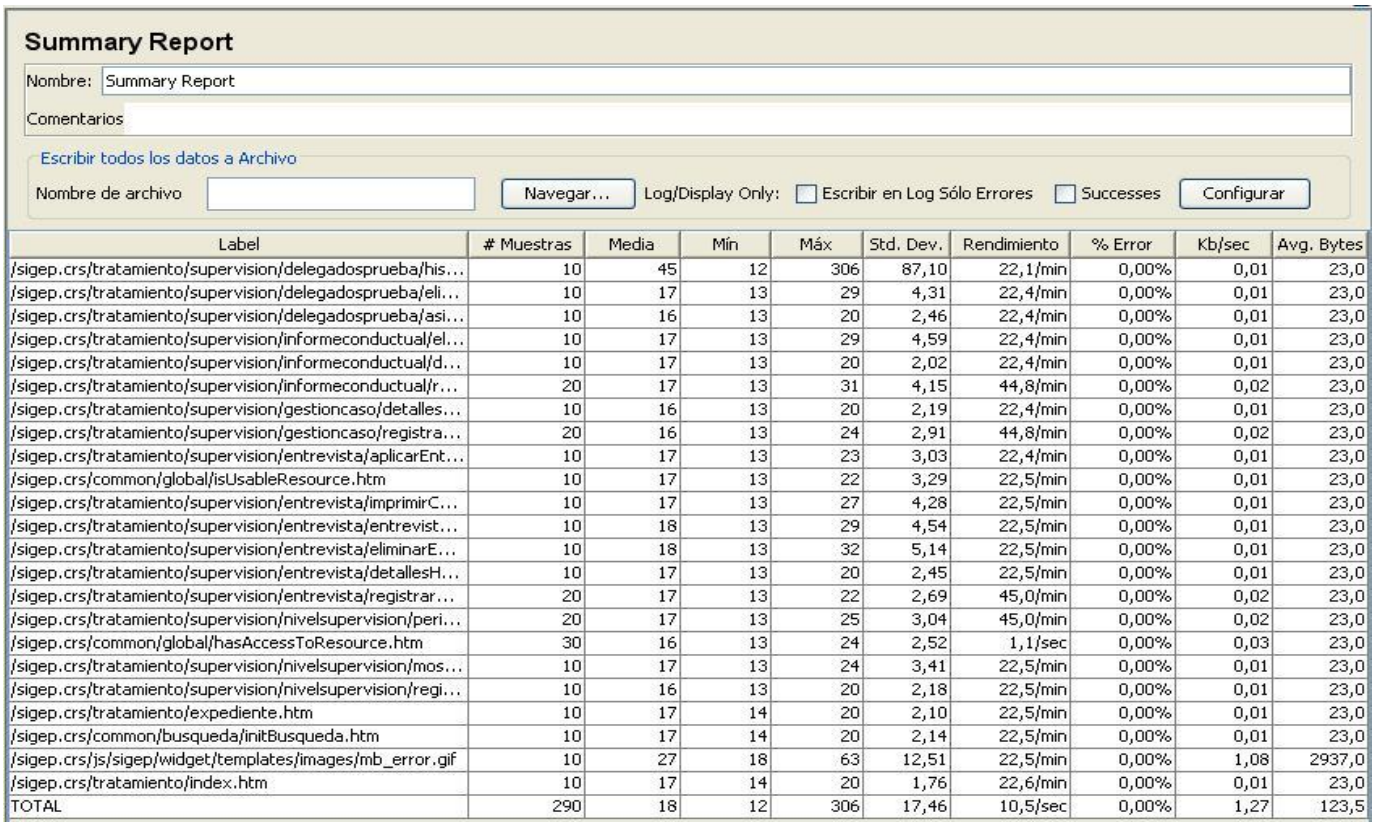
Para la primera prueba, se introducen 10 hilos (número de usuarios a simular) y 30 como período de subida (tiempo que debiera llevarle a JMeter lanzar todos los hilos). Si se seleccionan 10 hilos y el periodo

de subida es de 30 segundos, entonces cada hilo comenzará 3 segundos después de que el hilo anterior haya sido lanzado.

El “Summary Report” (Informe Resumen) crea una fila por cada petición en el test. Por cada una de estas filas se muestra la siguiente información:

- Label: El nombre de la muestra (conjunto de muestras).
- # Muestras: El número de muestras para cada URL.
- Media: El tiempo medio transcurrido para un conjunto de resultados.
- Mín: El tiempo mínimo transcurrido para las muestras de la URL dada.
- Máx: El tiempo máximo transcurrido para las muestras de la URL dada.
- Error %: Porcentaje de las peticiones con errores.
- Rendimiento: Rendimiento medido en base a peticiones por segundo/minuto/hora.
- Kb/sec: Rendimiento medido en Kilobytes por segundo.
- Avg. Bytes: Tamaño medio de la respuesta de la muestra medido en bytes.

Para este primer intento el elemento Summary Report mostrará el siguiente aspecto:



Summary Report

Nombre: Summary Report

Comentarios

Escribir todos los datos a Archivo

Nombre de archivo | Navegar... | Log/Display Only: | Escribir en Log Sólo Errores | Successes | Configurar

Label	# Muestras	Media	Mín	Máx	Std. Dev.	Rendimiento	% Error	Kb/sec	Avg. Bytes
/sigep.crs/tratamiento/supervision/delegadosprueba/his...	10	45	12	306	87,10	22,1/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/delegadosprueba/eli...	10	17	13	29	4,31	22,4/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/delegadosprueba/asi...	10	16	13	20	2,46	22,4/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/informeconductual/el...	10	17	13	29	4,59	22,4/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/informeconductual/d...	10	17	13	20	2,02	22,4/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/informeconductual/r...	20	17	13	31	4,15	44,8/min	0,00%	0,02	23,0
/sigep.crs/tratamiento/supervision/gestioncaso/detalles...	10	16	13	20	2,19	22,4/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/gestioncaso/registra...	20	16	13	24	2,91	44,8/min	0,00%	0,02	23,0
/sigep.crs/tratamiento/supervision/entrevista/aplicarEnt...	10	17	13	23	3,03	22,4/min	0,00%	0,01	23,0
/sigep.crs/common/global/isUsableResource.htm	10	17	13	22	3,29	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/entrevista/imprimirC...	10	17	13	27	4,28	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/entrevista/entrevist...	10	18	13	29	4,54	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/entrevista/eliminarE...	10	18	13	32	5,14	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/entrevista/detallesH...	10	17	13	20	2,45	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/entrevista/registrar...	20	17	13	22	2,69	45,0/min	0,00%	0,02	23,0
/sigep.crs/tratamiento/supervision/nivelsupervision/peri...	20	17	13	25	3,04	45,0/min	0,00%	0,02	23,0
/sigep.crs/common/global/hasAccessToResource.htm	30	16	13	24	2,52	1,1/sec	0,00%	0,03	23,0
/sigep.crs/tratamiento/supervision/nivelsupervision/mos...	10	17	13	24	3,41	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/supervision/nivelsupervision/regi...	10	16	13	20	2,18	22,5/min	0,00%	0,01	23,0
/sigep.crs/tratamiento/expediente.htm	10	17	14	20	2,10	22,5/min	0,00%	0,01	23,0
/sigep.crs/common/busqueda/initBusqueda.htm	10	17	14	20	2,14	22,5/min	0,00%	0,01	23,0
/sigep.crs/js/sigep/widget/templates/images/mb_error.gif	10	27	18	63	12,51	22,5/min	0,00%	1,08	2937,0
/sigep.crs/tratamiento/index.htm	10	17	14	20	1,76	22,6/min	0,00%	0,01	23,0
TOTAL	290	18	12	306	17,46	10,5/sec	0,00%	1,27	123,5

Figura 23 Prueba para 10 hilos en un periodo de subida de 30 segundos

Se puede observar que las pruebas se han realizado sin errores. Esto se deduce de la columna representativa del tanto por ciento de errores para cada una de las peticiones asociadas a cada conjunto de muestras. El rendimiento muestra que para una simulación de 10 usuarios junto a un periodo de subida de 30 segundos el servidor es capaz de aceptar una media de 10.5 peticiones por segundo.

Simulación de 60 usuarios con un periodo de subida de 180 segundos

Si se realiza la simulación con 60 usuarios considerando un periodo de subida de 180 segundos (nuevamente 3 segundos entre el lanzamiento de cada hilo) los resultados serán los siguientes, teniendo en cuenta que dichos resultando se irán solapando a los ya obtenidos en la simulación anterior.

En este caso se observa que las pruebas se han realizado sin errores nuevamente. El rendimiento muestra que para una simulación de 60 usuarios junto a un periodo de subida de 180 segundos el servidor es capaz de aceptar una media de 3.5 peticiones por minuto.

En el Anexo 4 se muestra como queda el elemento "Summary Report" para este intento.

A medida que se va aumentando el número de hilos y el periodo de subida entre hilos, aumenta considerablemente el porciento de error y el rendimiento va disminuyendo.

En las pruebas realizadas al módulo no se pudo continuar debido a la carencia de memoria de la PC. En este caso se puede hablar de limitaciones de memoria. El sistema solo aguantó 120 usuarios con un periodo de subida de 120 segundos.

4.8 Conclusiones parciales

En este capítulo se presentaron los elementos más representativos de la implementación del módulo Supervisión, teniendo en cuenta las pautas de la arquitectura del sistema y el diseño definido para la realización de las funcionalidades descritas. El desarrollo de este módulo no provocó ningún cambio en otros módulos de la aplicación.

Se debe destacar que con la utilización de librerías y facilidades que ofrecen los frameworks durante el proceso de implementación se agilizó el proceso, ahorrando código y tiempo.

Para comprobar la calidad del producto se realizaron algunas pruebas y para ello se utilizaron herramientas que favorecieron la agilidad y confiabilidad del proceso. Se detectaron algunas no conformidades, las cuales fueron resueltas inmediatamente, logrando correspondencia entre las funcionalidades y lo que el sistema hace.

Conclusiones

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

- Se definió el marco teórico de la investigación, definiéndose todos los elementos necesarios para buscar la respuesta al problema de investigación, dígase objeto de estudio, los objetivos y la idea a defender.

- Se realizó un refinamiento de los caso de uso definidos en el proyecto, obteniéndose el Diagrama de Casos de Uso del Sistema y la Descripción Textual de cada uno de ellos.

- Se diseñó del módulo Supervisión, con el cual se obtuvo el Modelo de Diseño y en particular los Diagramas de Clases del Diseño para cada uno de los casos de uso identificados, divididos en capas cumpliendo con la arquitectura definida por el proyecto.

- Se implementaron todas las funcionalidades definidas para el módulo Supervisión durante la fase de inicio del proyecto con la utilización de las herramientas y tecnologías definidas, lográndose una aplicación funcional integrada al SIGEP v2.1.

- Se validó y verificó la calidad de la solución propuesta con la realización de pruebas, demostrándose que cada una de las funcionalidades responde apropiadamente a los requisitos funcionales definidos, incluyendo la navegación, la entrada de datos, el procesamiento y la obtención de resultados. De manera general los resultados obtenidos en estas pruebas fueron buenos.

Recomendaciones

Se recomienda la realización de un monitoreo constante del rendimiento del módulo en la medida en que la cantidad de datos que el sistema maneja vaya creciendo, puesto que actualmente el sistema fue probado en máquinas clientes que no cumplen los requisitos del servidor donde debería probarse la aplicación, por lo que no se pudo verificar el límite al cual colapsa la solución propuesta.

Referencias Bibliográficas

1. **BARATTA, Alessandro.** *Resocialización o Control Social. Por un Concepto Crítico de Reintegración Social del Condenado.* Universidad del Saarlan, R.F.A. : s.n., 1990.
2. Biblioteca.com. [En línea] [Citado el: 2010 de Diciembre de 8.] http://www.analitica.com/bitblio/anc/constitucion1999.asp.
3. **Osorio, Manuel.** *Diccionario de Ciencias Jurídicas, Políticas y Sociales.* s.l. : Guatemala.
4. **Juan Carlos Gomez, Yadira Venavides Zaila.** *Diseño e implementación de los módulos Decisiones.* UCI : s.n., 2008.
5. [En línea] [Citado el: 18 de 12 de 2010.] http://www.rational.com.ar/herramientas/rup.html.
6. Ecured. [En línea] [Citado el: 2010 de 10 de 10.] http://www.ecured.cu/index.php/RUP.
7. [aut. libro] Ivar Jacobson, Grady Booch James Rumbaugh. *El Lenguaje Unificado de Modelado. Manual de Referencia.* pág. 56.
8. **Granda, Roberto.** *TRABAJO DE DIPLOMA Diseño e Implementación del módulo Requisas y Decomisos.* Ciudad de la Habana : s.n., 2010.
9. **H., Stephen Kaisler.** *Software Paradigms.* 2005.
10. **BREINDENBACH, CRAIG WALLS Y RYAN.** *Spring in Action, Second Edition.* s.l. : Manning Publications Co, 2008.
11. **King, Christian Bauer y Gavin.** *Hibernate in action.* s.l. : Manning Publications Co, 2005.
12. **Kaisser, S. H.** *Softweare Paradigms.* 2005.
13. **Pressman, Roger.** *Ingeniería del Softweare, un enfoque práctico.* 2002.
14. **Somerville, Ian.** *Softweare Engineering.* 2006.
15. *wikilearning.* [En línea] [Citado el: 30 de Marzo de 2011.] http://www.wikilearning.com/tutorial/desarrollo_orientado_a_objetos_con_uml-diagrama_de_casos_de_uso/6321-5.
16. **González, Luis Alberto Pimentel.** *Documento de Arquitectura de Software.* Ciudad de La Habana : s.n.
17. **Viacava, Daniel Perovich y Leonardo Rodríguez.** Gestión de Transacciones en la Plataforma J2EE.
18. **Pecos, D.** PostGreSQL. 2005.
19. **Basco, Sociedad Informatica del Gobierno.** *JMeter. Manual de usuario.*
20. uptodown. [En línea] [Citado el: 2011 de 04 de 10.] http://jmeter.uptodown.com/.
21. EVA. [En línea] [Citado el: 2011 de 04 de 12.] http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_9/Conferencia_7/Materiales_Complementarios/Material_de_caja_b_y_caja_n.pdf.
22. **Jorrín, Ing. Violena Hernández Aguilar y Ing. Michael González.** *Proceso de pruebas de caja negra basado en la descripción de casos de uso. .* UCI : s.n.
23. **Boucchechter, Carolina Zibert van Gricken Israel.** [En línea] 30 de mayo de 2005. [Citado el: 2011 de 04 de 17.] http://carolina.terna.net/ingsw3/datos/Pruebas_de_Confiabilidad.pdf.