Universidad de las Ciencias Informáticas

Facultad 2

Título: Diseño e Implementación del módulo Control de Capacidades del SIGDATI.

Trabajo de Diploma por el título

de Ingeniero en Ciencias Informáticas.

Autor: Alejandro Miguel Tellez Pereira

Tutor: Msc. Julio Omar Prieto Entenza

Ciudad de La Habana, 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 _____.

_____

Alejandro Miguel Tellez Pereira

_____

Msc. Julio Omar Prieto Entenza

Tutor: *MSc. Julio Omar Prieto Entenza*

Correo electrónico: jprieto@uci.cu

Categoría Científica: Máster

Categoría Docente: Instructor

- Graduado del 2007 de Ingeniero en Ciencias Informáticas
- Ha participado en eventos como Uciencia e Informática 2009
- Ha dirigido varias tesis de diploma
- Vinculado a proyectos productivos y de investigación como desarrollador y arquitecto.

"El genio es uno por ciento de inspiración y un noventa y nueve por ciento de dedicación."

Thomas Alva Edison

AGRADECIMIENTOS

Quisiera agradecerle a todo el equipo de desarrollo del proyecto que sin la colaboración de uds hoy no estaría realizando la defensa de mi tesis en especial: a mí tutor Julio por ayudarme en todos los aspectos y si hay alguien a quien debo dar gracias es a él, a Yanet la jefa por su comprensión y su apoyo en todo, a Leandro, a Otto, a los dos Reinier, a Maikel David, al Javico y a los demás por hacerme sentir parte de su grupo.

A mis padres por todo el apoyo y la confianza que han depositado en mí y siempre fue mi objetivo regalarles este título.

A mi hermano Ariel sobran las palabras por estar ahí y querer ser un ejemplo para ti.

A mi familia completa, todos mis abuelos, mis tías (que son mis madres), mis primos y a todos en fin muchas gracias.

A mis hermanos Ramón y Kmilo por ser unos amigos incomparables.

A mi Bebé por hacer que estos dos últimos años en la escuela se viera lleno de amor y alegría junto a ti…Espero sea el comienzo de una vida entera a tu lado. Te quiero mucho.

A todas mis amistades del Pre, del servicio y de la UCI decir nombres sería dejar fuera algunos, lo importante es saber que siempre que he tenido que contar con uds han estado ahí y les agradezco por su apoyo y su amistad de verdad sin uds estos años no hubieran tenido alegrías, ni disgustos hubiera sido un tiempo null.

Este trabajo es dedicado a mi abuelita Ita, daría todo lo que tengo en este mundo porque pudieras verme hoy haciendo realidad tu sueño. Sé que desde el cielo estas dándome ese aliento que siempre me distes y leyéndome tus oraciones para que salga bien. Te quiero mucho Ita.

A mi mama por ser la mami más linda del mundo y espero te sientas orgullosa de tu nené que una vez mas no te defraudé. No hay palabras para describir ahora mismo todo lo que a lo largo de mi vida me has enseñado y lo que siento hacia ti. Te quiero mucho.

A mi papá. Papi mejor no lo quiero porque mejor que tú no hay nada gracias a ti hoy soy un hombre que lleva siempre la frase "Dale que tú sabes pa´ eso y mucho más" esas palabras de fuerza que me dices siempre que tengo malos momentos. Te quiero.

A mi hermano Ariel. Espero que estés orgulloso de mi y que sigas mi ejemplo y te mantengas como vas por el camino correcto para que te conviertas en un hombre de bien. Te quiero.

RESUMEN

El Sistema Penitenciario Nacional en la actualidad cuenta con un Sistema Automatizado para el Control de Recluso (SACORE) para gestionar todo la información penitenciaria de los individuos que se encuentran cumpliendo sanción en dichos centros. Dicho sistema a pesar de que se ha ido perfeccionando a partir de los requerimientos y solicitudes de los usuarios durante sus ocho años de explotación actualmente continua presentando problemas al no estar informatizados en su totalidad muchos de los proceso priorizados, entre los que se encuentran los procesos referentes al Control de Capacidades. Esta situación además de originar un trabajo riguroso y arcaico ocasiona graves problemas no solo para los centros sino para todo aquel que de una forma u otra tiene el deber de controlar y gestionar todas las actividades en los centros penitenciarios.

En el siguiente trabajo se realiza el Diseño e Implementación del módulo Control de Capacidades del Sistema de Gestión de Datos del Interno (SIGDATI). Dando cumplimiento durante el desarrollo del mismo a las tareas y objetivos planteados a partir del análisis de los requerimientos de software establecidos en acuerdo con el cliente y haciendo uso de las tecnologías y herramientas definidas para el proyecto.

Índice de Contenido

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

INTRODUCCIÓN

Antecedentes

Luego del triunfo de la revolución el 1ro de enero de 1959 el gobierno cubano toma un sin número de medidas con el objetivo de revertir muchas de las políticas y leyes que había dejado como legado la tiranía. Una de las medidas que tomó en aras de hacer un ordenamiento jurídico y crear nuevas formas para enfrentar los delitos en correspondencia con los perjuicios que éstos producían fue la de renovar el sistema penitenciario existente en aquel entonces. Muchas de las antiguas prisiones heredadas de la tiranía, donde el abuso, las torturas, los tratos inhumanos a los internos, así como los maltratos(los cuales constituían los métodos y procedimientos que caracterizaban la estancia de los detenidos en la prisión), fueron desactivadas y se construyó un Sistema Penitenciario totalmente humano, basado en el respeto y en una gestión de control inspirada en la regulación de normas y leyes internacionales en cuanto al tratamiento a los reclusos. Durante estos 52 años de revolución el sistema penitenciario actual ha sufrido un sin número de transformaciones que no solo han posibilitado cumplir con los fines de la sanción penal sino que también ha ayudado a situar a nuestro sistema de justicia entre los de mayor carácter humanista del mundo, por contar entre sus fortalezas con el perfeccionamiento de la legislación penitenciaria y de su base reglamentaria.

En el año 1989 comienza en Cuba el sistema de informatización de los centros penitenciarios, con la automatización de los datos principales del recluso y ciertos aspectos de control penal. Pero no es hasta algunos años más tarde que a raíz del cumplimiento de la orden 43/99 del Vice Ministro Primero se crea un Sistema Automatizado para el Control del Recluso, SACORE por sus siglas, el cual comenzó a dar respuesta en gran medida a muchos de los problemas descritos anteriormente. Dicho sistema culminó su desarrollo a finales del 2002, poniéndose en marcha a principios del 2003 y cuenta en la actualidad con tres módulos principales: Control Penal, Reeducación Penal y el Orden Interior, los cuales aumentan las especificaciones de la automatización de los datos principales del recluso y los aspectos de control penal existentes, además de adicionar un mayor número de facilidades. En los 8 años de explotación del sistema, a pesar de sus facilidades y ayuda brindada a dichos centros aún cuenta con requisitos incompletos o pendientes.

La Universidad de Ciencias Informáticas, como vanguardia en el desarrollo de las tecnologías de la información, ha asumido la modernización informática del Sistema Penitenciario Cubano a través del proyecto SIGDATI[1], siguiendo al pie de la letra todas las disposiciones legales del MININT, los Órganos de Justicia y del Estado. Una de las áreas aún no implementadas es el módulo de Control

[1] Sistema de Gestión de Datos del Interno

de Capacidades el cual, por su importancia; debe permitir definir la estructura física del área de reclusión penal y mantener un control sobre las capacidades penitenciarias, los niveles de ocupación, la disponibilidad y las afectaciones. Además debe ser capaz de registrar las capacidades asignadas por la DEP (Dirección de Establecimientos Penitenciarios) y la jefatura provincial, de modo que se conozca con antelación los nuevos ingresos que recibirá el centro penitenciario.

Debido a lo anterior se plantea como ***problema a resolver*** ¿Cómo gestionar las capacidades de la DEP cumpliendo con los parámetros definidos para la arquitectura y los requisitos del proyecto SIGDATI?

Definiendo a su vez como ***objeto de estudio:***

Los procesos de desarrollo de software del módulo Control de Capacidades del Sistema Penitenciario Cubano.

Y quedando definido como ***campo de acción:***

Los modelos de diseño e implementación del módulo Control de Capacidades del Sistema Penitenciario Cubano.

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

Objetivos específicos:

- Diseñar el módulo Control de Capacidades
- Implementar el módulo diseñado
- Integrar el módulo a la plataforma SIGDATI

En aras de dar cumplimiento al objetivo planteado se traza el siguiente conjunto de ***tareas generales:***

- Elaboración del modelo conceptual que cumpla con los requisitos relacionados con el Control de Capacidades.
- Desarrollo del diagrama de clases del diseño.
- Implementación del diseño realizado
- Diseño del diagrama de componentes

El documento consta de 3 capítulos además de los Anexos, a continuación se brinda un resumen de los contenidos que abordarán los mismos.

Capítulo 1: Fundamentación Teórica

Aborda conceptos relacionados con los Sistemas Penitenciarios y el tratamiento del Control de Capacidades en los mismos. Se analizaran las principales herramientas y tecnologías a utilizar para el desarrollo de los procesos relacionados con el diseño e implementación del módulo Control de Capacidades del Sistema Penitenciario Cubano.

Capítulo 2: Arquitectura del Sistema

Aborda todo lo referente al diseño propuesto para la solución del software así como la descripción de la arquitectura a utilizar. Se especifican además el conjunto de funcionalidades a desarrollar dando un breve resumen de las mismas.

Capítulo 3: Diseño e Implementación

Se detallan las actividades definidas para el diseño e implementación del módulo Control de Capacidades las cuales darán cumplimiento a las funcionalidades propuestas en el capítulo anterior.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA.

1. Introducción

En el presente capítulo se abordan algunos conceptos relacionados con los Sistemas Penitenciarios y el tratamiento de Control de Capacidades en los mismos. Se analizan además las principales herramientas y tecnologías a utilizar para el desarrollo de los procesos relacionados con el diseño e implementación del Control de Capacidades de los Centros Penitenciarios del Sistema Penitenciario Cubano.

1.1. Sistema Penitenciario

Se define Sistema Penitenciario como el órgano encargado de garantizar el proceso de ejecución de la sanción de privación de libertad, de la sanción de trabajo correccional con internamiento, la medida de seguridad reeducativa de internamiento y la medida cautelar de prisión provisional.[1]

Cada estado estipula dichas organizaciones basándose en sus concepciones políticas, y adecuándose a las condiciones sociales y económicas de su desarrollo. En nuestro país este sistema está dirigido por la Dirección de Establecimientos Penitenciarios del Ministerio del Interior y se sustenta en la integración de principios, conceptos, procedimientos, fuerzas y medios que garantizan el funcionamiento de los centros destinados al internamiento y el tratamiento a los internos. A su vez los fundamentos de la política penitenciaria están determinados en la Constitución de la República, La Ley 62, Código Penal, además la organización y las condiciones de ejecución de las sanciones privativas de libertad se corresponden con lo establecido en las "Reglas Mínimas Clásicas para el Tratamiento a los Reclusos" aprobadas el 30 de agosto de 1955 por la Organización de las Naciones Unidas (ONU) y la Declaración Universal de los Derechos del Hombre del 10 de diciembre de 1948.[2]

1.2. Sistemas Informáticos en Sistemas Penitenciarios

En la actualidad los productos informáticos evolucionan dinámicamente para poder adaptarse a las necesidades tecnológicas del momento y los Centros Penitenciarios al igual que disímiles organizaciones se han debido modernizar. Aspectos como la seguridad, control y evaluación se han visto día a día amenazados por la evolución constante de los sistemas informáticos y el rápido incremento de las tecnologías. Viejos y rigurosos mecanismo de trabajo, así como los arduos procesos de gestión existentes durante décadas en las prisiones hoy en día han sido sustituidos por eficientes soluciones informáticas capaces de gestionar dichos procesos con la máxima seguridad y mínimas dificultades.

Existen diversos sistemas para la gestión penitenciaria, muchos países tienen el suyo propio. Estos sistemas tienen como objetivo facilitar el control de los reclusos y gestionar los procesos esenciales en los establecimientos penitenciarios relacionados con la población penal. También cuentan de una manera u otra, con un sistema de reportes, que les permite saber del estado de ciertos indicadores.

1.2.1. Sistema Automatizado para el control de Capacidades de Cuba

El sistema SACDEP (Sistema Automatizado de Capacidades de la Dirección de Establecimientos Penitenciarios), se crea como apoyo a SACORE (Sistema Automatizado para el Control de Reclusos) con el objetivo de poseer información sobre las capacidades de los establecimientos penitenciarios, permitiendo registrar información de todos los locales del centro y dar soporte a una mejor ubicación de los internos. Además controla datos desde las áreas de reclusión hasta las áreas fuera de la misma, como son: las redes técnicas eléctricas, sanitarias, hidráulicas y de seguridad. Entre sus principales funcionalidades se encuentran:

- Diagnóstico de la infraestructura de las edificaciones
- Capacidad de dormitorios en el área de reclusión
- Clasificación, donde registra información importante sobre la construcción del establecimiento, datos generales sobre el estado del centro y sobre su estructura interna.
- Órganos que tienen vínculo con el establecimiento.
- Registrar información de las redes técnicas, como el consumo eléctrico, plantas eléctricas, abasto de agua, depósitos de agua, garitas de vigilancia, alcantarillado, entre otros.
- Aunque es un sistema independiente, es decir, que no está integrado al SACORE, cubre las necesidades de controlar las capacidades de los establecimientos penitenciarios en el país.[3]

A pesar de sus funcionalidades presenta varios problemas:

- La usabilidad es pobre debido fundamentalmente que la interfaz es poco amigable, compleja y un tanto desorganizada. Esto conlleva a que los usuarios no se sientan cómodos al trabajar con él.
- Al no formar parte del SACORE, debe instalarse aparte, lo que provoca que los usuarios se deban intercambiar constantemente de un entorno hacia otro, conllevando a una pérdida importante de tiempo.
- Debido a los cambios en el Sistema Penitenciario Cubano los tipos de edificaciones han aumentado, algo que este sistema no contempló. Por eso, al no existir la posibilidad de incorporar nuevas edificaciones se hace necesario volver a implementar el sistema.

La gestión de estas informaciones en estos momentos no se realiza de forma efectiva y no se tiene un control exacto de las capacidades con que cuenta el sistema, conllevando a que en ocasiones se

asignen internos a lugares donde haya exceso de personal, a lugares inhabitables o en una agrupación a la que no pertenece.

1.3. Herramientas y Tecnologías a utilizar.

Como metodología de referencia para guiar el trabajo de implementación, se utilizó ***RUP***[2]***.***

RUP es un proceso que se caracteriza por ser:

- **Dirigido por casos de uso.** Los casos de uso describen los requisitos funcionales del sistema desde la perspectiva del usuario y se usan para determinar el alcance de cada iteración y el contenido de trabajo de cada persona del equipo de desarrollo.
- **Centrado en la arquitectura**. La arquitectura permite ganar control sobre el proyecto para manejar su complejidad y controlar su integridad. Hace posible la reutilización a gran escala y provee una base para la gestión del proyecto.
- **Iterativo e incremental.** Se divide en 4 fases: Inicio, Elaboración, Construcción y Transición, y cada una de ellas se divide en iteraciones. En cada iteración se trabaja en un número de disciplinas haciendo énfasis en algunas de ellas. Las disciplinas propuestas por RUP son: Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, entre otras.

Cada iteración añade funcionalidades al producto de software o mejora las existentes. [4]

1.3.1. Herramienta de modelado

La herramienta de modelado de software utilizada para generar y representar los artefactos del sistema a realizar es:

- **Visual Paradigm Suite 3.1:** Visual Paradigm es una herramienta UML[3] profesional que soporta el ciclo de vida completo del desarrollo de software. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste, ya que por un lado el uso de lenguajes visuales facilitan su asimilación y entendimiento por parte del equipo de desarrollo, el tiempo invertido en el desarrollo de la arquitectura se minimiza, la detección y resolución de errores se agiliza siempre y cuando se haga uso de herramientas adecuadas de diagnóstico y depuración; y la trazabilidad y documentación del proyecto se realiza de una forma ordenada y guiada por los casos de uso. Proporciona además abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.[5]

1.3.2. Herramientas de desarrollo

Las herramientas de desarrollo utilizadas para llevar a cabo el desarrollo de las funcionalidades son:

---

[2] ***P****roceso* ***U****nificado de* ***D****esarrollo,* **R**ational **U**nified **P**rocess

[3] ***L****enguaje* ***U****nificado de* ***M****odelado*, **U**nified **M**odeling **L**anguage

- **NetBeans 6.8 Milestone2:** El NetBeans IDE[4] es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring. Cuenta con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente, emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad. Puede extenderse usando otros lenguajes de programación como son C/C++ y Phyton y trae incluido entre sus novedades más destacadas el uso de Groovy.[6]
- **Contenedor Web (Apache Tomcat 6.025):** Apache Tomcat es un contenedor Web escrito en Java, por lo que funciona en cualquier sistema operativo que disponga de una máquina virtual Java y desarrollado en un ambiente participativo y abierto. Apache Tomcat es usado en numerosas aplicaciones web de gran escala y críticas en diversas industrias y organizaciones que se referencian en su sitio oficial.[7]
- **Subversión:** Subversión es un software de sistema de control de versiones. Esta desarrollado sobre software libre bajo una licencia de tipo Apache/BSD y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversión es que los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. El acceso al repositorio es mediante la red, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. Su principal importancia radica en que varias personas pueden modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomentando así la colaboración. Permite además integrar varias interfaces a entornos de desarrollo, un ejemplo de esto son el TortoiseSVN el cual provee integración con el explorador de Windows. Y como carencia o desventaja tenemos que el manejo de cambio de nombres de archivos no es completo, pues este es manejado como la suma de una operación de copia y una de borrado. [12]

1.3.3. Sistema Gestor de Base de Datos

- **Oracle Database 11g:** Es un Sistema Gestor de Bases de Datos con características objeto-relacionales, utilizando tecnología cliente/servidor. Permite la gestión de grandes bases de datos,

[4] ***E**ntorno de **D**esarrollo **I**ntegrado*, **I**ntegrated **D**evelopment **E**nvironment

un alto rendimiento en transacciones, disponibilidad controlada de los datos de las aplicaciones, la gestión de la seguridad y la autogestión de la integridad de los datos. Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando: soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma. Además se encuentra muy difundido sobre todo en las grandes compañías, transnacionales o instituciones gubernamentales por sus elevados precios de licencias de software, soporte técnico y sus elevados requisitos de hardware. [8]

1.3.4. Tecnologías

Las tecnologías utilizadas para el desarrollo de las funcionalidades son:

- **JSON: (Javascript Object Notation)** formato ligero para intercambio de datos. JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX[5]. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador semántico de JSON. [9]
- **Javascript:** Lenguaje script utilizado para unir el conjunto de tecnologías usados en la web. JavaScript es un lenguaje de scripting basado en objetos, utilizado para acceder a objetos en aplicaciones. Principalmente, se utiliza integrado en un navegador web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas. JavaScript se caracteriza por ser un lenguaje basado en prototipos, con entrada dinámica y con funciones de primera clase. Todos los navegadores modernos interpretan el código JavaScript integrado dentro de las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM[6]. Su principal importancia radica en que tradicionalmente, se venía utilizando en páginas web HTML[7], para realizar operaciones y en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se ejecuta en el agente de usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML. [10]
- **Dojo Toolkit:** Librería de clases Javascript. Dojo es un frameworks que contiene Apis y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. 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.

[5] ***J**avaScript **A**síncrono **Y** **X**ML*, **A**synchronous **J**avaScript**A**nd **X**ML
[6] ***M**odelo de **O**bjeto del **D**ocumento*, **D**ocument **O**bject **M**odel
[7] ***L**enguaje de **M**arcas de **H**ipertextos*, **H**ypertext **M**arkup **L**anguage

Resuelve asuntos de usabilidad comunes como pueden ser la navegación y detección del navegador, soportar cambios de URL[8] en la barra de URLs para luego regresar a ellas, y la habilidad de degradar cuando AJAX/JavaScript no es completamente soportado en el cliente. Es conocido como "la navaja suiza del ejército de las bibliotecas Javascript". Proporciona una gama más amplia de opciones en una sola biblioteca JavaScript y es compatible con navegadores antiguos. [11]

- **Grails:** Framework para aplicaciones web libre, desarrollado sobre el lenguaje de programación Groovy (el cual a su vez se basa en la Java Plataform). Grails pretende ser un marco de trabajo altamente productivo siguiendo paradigmas tales como convención sobre configuración o no te repitas (DRY), proporcionando un entorno de desarrollo estandarizado y ocultando gran parte de los detalles de configuración al programador. [13]

Sus principales características son:

- **Alta productividad:** Grails tiene tres características que intentan incrementar su productividad comparándolo con los Framework Java tradicionales:
  1. Inexistencia de configuración XML
  2. Entorno de desarrollo preparada para funcionar desde el primer momento
  3. Funcionalidad disponible mediante métodos dinámicos.
- **Integración con la plataforma Java:** Al estar construido sobre la plataforma Java, con lo que es muy fácil integrarlo con librerías Java, Framework y código existente. La mejor característica que Grails ofrece en este ámbito es una integración transparente con clases mapeada mediante el Framework Hibernate ORM[9]. Esto significa que aplicaciones existentes que utilicen Hibernate pueden utilizar Grails sin recompilar el código o reconfigurar las clases Hibernate, aprovechando los métodos de persistencia que se mencionan anteriormente. Una consecuencia es que se puede utilizar scaffolding con las clases Java mapeadas con Hibernate. Otra consecuencia es que las capacidades de Grails están totalmente disponibles para estas clases y las aplicaciones que las usan.
- **Persistencia:** El modelo de datos en Grails se graba en la base de datos utilizando GORM (Grails Object Relational Mapping). Las clases de dominio se guardan en el directorio *grails-app/domain.*

Grails, como frameworks de aplicaciones web y con la visión de convertirse en un marco de trabajo altamente productivo no puede estar ajeno al uso de patrones por lo que utiliza como

[8] ***L****ocalizador* ***U****niversal de* ***R****ecursos,***U**niversal **R**esource **L**ocation
[9] ***M****apeo de* ***O****bjeto* ***R****elacional*, **O**bject **R**elational **M**apping

principales paradigmas en este sector dos patrones fundamentales: Convención sobre Configuración o también conocidos por sus siglas en Ingles como Convention over Configuration y DRY, más conocido también como Don't Repeat Yourself.

Convention over Configuration es un paradigma de programación de software que busca decrementar el número de decisiones que un desarrollador necesita hacer, ganando así en simplicidad pero no perdiendo flexibilidad por ello y DRY es una filosofía de definición de procesos que promueve la reducción de la duplicación. Ambos patrones en general proporcionan un entorno de desarrollo estandarizado y ocultan en gran parte detalles de configuración. [18]

Con la utilización del frameworks Spring como componente de Grails se utilizan diferentes patrones de dicho frameworks que aportan un mejor diseño e implementación al sistema. Un ejemplo de esto es la utilización del patrón Modelo Vista Controlador. Spring-MVC es uno de los módulos del Framework de Spring, y como su nombre indica implementa una arquitectura Modelo - Vista - Controlador, explicada anteriormente, que se utilizará como base para desarrollar la capa de presentación de la aplicación. [17]

La inversión de control(IoC) es otro patrón utilizado en Grails, según el cual las dependencias de un componente no deben gestionarse desde el propio componente para que este solo contenga la lógica necesaria para hacer su trabajo.[18]

1.3.5. Arquitectura en Capas según Grails.

La arquitectura de Grails está conformada por 3 capas lógicas principales: Web Layer, Service Layer, y Data Layer, donde cada capa está separada de la siguiente e interactúan mediante interfaces que definen funcionalidades que la misma debe brindar o también llamadas fachadas las cuales aseguran que el acoplamiento sea el más bajo posible y la abstracción del funcionamiento de la capa inferior, sea casi total. Cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Esta arquitectura en capas por su diseño proporciona la facilidad de modificar cada capa todo lo posible sin infligir daños o alteraciones a la capa inmediata, a continuación se brinda una descripción de las mismas:

Web Layer: En esta capa se encuentran las Vistas y la Lógica de Presentación, las cuales según la arquitectura que propone Grails estarán enmarcadas sus clases en los paquetes Controllers, el cual contendrá las clases controladoras y View and Layouts donde se encontrarán las Groovy Server Pages o más conocidas como GSP. En la Lógica de Presentación se manejará todo el flujo web utilizando la implementación del patrón Modelo Vista Controlador (MVC) que nos brinda Grails

mediante Spring MVC. Grails implementa el patrón MVC, en el que la lógica del negocio se separa de la presentación de la aplicación. Esto permite cambiar fácilmente el aspecto de la aplicación, sin modificar su comportamiento. La capa de presentación se compone principalmente de: modelo, vistas, controladores.

- **Controlador:** Un controlador de Grails es una clase responsable por el manejo de los pedidos provenientes de la aplicación. El controlador recibe la petición, realiza algún trabajo potencial con la misma y finalmente decide que sucederá a continuación, lo cual puede incluir algunos de los siguientes flujos.
  - Ejecutar otra función de controlador.
  - Mostrar una vista.
  - Mostrar información directamente con la respuesta de la petición.

Un controlador es prototipado, lo cual significa que una nueva instancia es creada por cada petición, por tanto los desarrolladores no necesitan manejarlos en modo "singleton[10]". Proveen la entrada principal para cualquier aplicación de Grails, coordinando los pedidos entrantes, delegando hacia los servicios o clases de dominio para la lógica de negocios y renderizando las vistas.

- **Modelo:** Una de las actividades fundamentales llevadas a cabo por los controladores es obtener los datos que serán mostrados en la vista. El controlador puede recoger esta información directamente, delegarla a algún servicio u otro componente comprendido en la capa de acceso a datos esta información es pasada a la vista en forma de un mapa u objeto de información. Dicho objeto representa el modelo.
- **Vista**: Grails utiliza para la interacción con el usuario la tecnología JSP[11]. Pero basada en una implementación mediante GSP, que es una extensión de JSP y puede incluir Groovy. El mismo permite a los desarrolladores mezclar etiquetas de lenguajes de marcas tradicionales como HTML con código Java para producir vistas dinámicas. Las Vistas son los recursos que junto al modelo generado por los controladores le permiten al cliente visualizar la información, estos pueden ser páginas HTML, documentos en formato PDF, hojas de cálculo, entre otras. Las misma están representadas en el paquete de clase View and Layouts.

Service Layer: En esta capa se encapsula toda la lógica de la aplicación en fachadas de negocio que son utilizadas por los controladores en la capa de presentación y se exponen algunos procesos de negocio a través de interfaces de servicios. Sus clases radicarán según la arquitectura propuesta por Grails en el paquete Services. A estas fachadas de negocio se le aplican la seguridad a nivel de métodos y de objetos de negocio, auditorías, cache, política de transacciones, entre otros.

[10] *Patrón de diseño Instancia Única.*
[11] ***P****ágina* ***S****ervidora de* ***J****ava*, **J**ava**S**erver **P**age.

Data Layer: Maneja los objetos de acceso a datos abstrayéndolos del mecanismo de persistencia usado; a través de interfaces que exponen las operaciones de persistencia. Grails para evitar trabajar directamente con un gestor de base de datos y sus tablas y permitir trabajar con objetos en su lugar utiliza Hibernate 3 como una herramienta ORM pero esta vez dada la naturaleza dinámica de Grails y la adopción del convenio sobre la configuración crea sobre una versión superior de una nueva implementación de Hibernate llamado Grails objeto mapeo relacional (GORM) que simplifica el trabajo con Hibernate y elimina cualquier configuración externa. [18]

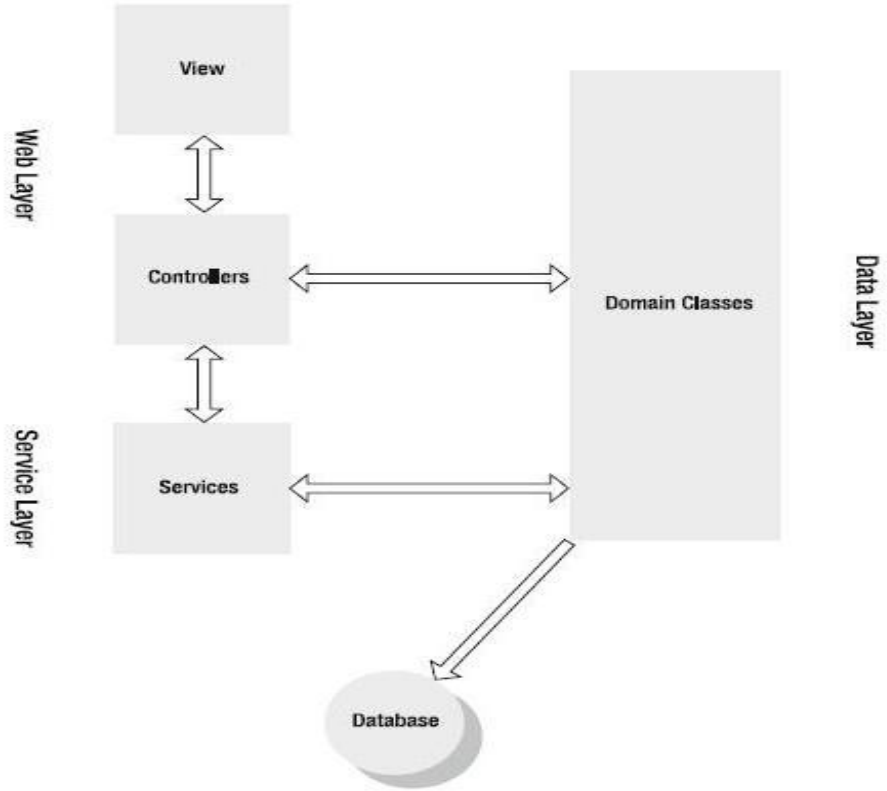


Figura 1 Arquitectura en Capas según Grails.

1.3.6. Arquitectura Cliente-Servidor

La arquitectura cliente servidor se encuentra dentro de la clasificación de estilo de llamada y retorno, donde dos tipos de aplicaciones se encuentran ejecutándose de forma independiente, una de estas aplicaciones se ejecuta como cliente y la otra como servidor. El cliente y el servidor generalmente están localizados en diferentes sistemas, sin embargo pueden encontrarse en el mismo. El cliente es la entidad que hace la petición por un servicio. El servidor es la entidad que provee el servicio correspondiente a la petición. El servicio debe procurar el resultado, el cual es retornado. Esta arquitectura posibilita no solo una mayor escalabilidad y una manera fácil de hacerla sino que además proporciona un reparto de carga adecuado para dichos sistemas. Por parte de la aplicación

cliente hay una menor complejidad en los procesos así como una menor ocupación de la memoria y la aplicación Servidor permite a su vez un menor tráfico en la red.

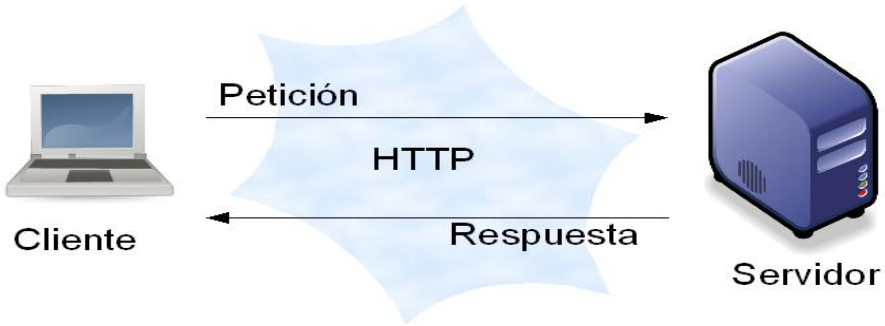


Figura 2 Arquitectura Cliente-Servidor.

1.4. Conclusiones Parciales.

En este capítulo se realizó un estudio más detallado de todo lo concerniente con el Sistema Penitenciario Nacional lo que permitió comprobar y analizar el estado actual del mismo en todo lo referente al Control de Capacidades Centros Penitenciarios. Se estudió el sistema SACDEP (Sistema Automatizado de Capacidades de la Dirección de Establecimientos Penitenciarios) demostrando las insuficiencias que este presenta. Se abordaron además todas las metodologías, herramientas y tecnologías propuestas por el proyecto SIGDATI para la implementación de la solución del módulo Control de Capacidades.

CAPÍTULO II: ARQUITECTURA DEL SISTEMA

2. Introducción

El presente capítulo abordará principalmente el diseño propuesto para la solución del software así como la descripción de la arquitectura a usar en el proyecto SIGDATI, sobre la cual se va a enmarcar a su vez el desarrollo del módulo Control de Capacidades. Se dará una breve descripción de sus componentes así como un análisis de los mismos.

2.1. Requisitos Funcionales del Sistema.

Tabla 1 Requisitos Funcionales del Sistema.

Nº	Funcionalidad	Descripción
RF1.1.1	Registrar estructura	Permite registrar los datos de un centro asociado a una provincia, definiendo el tipo de centro (centro, CTEM o asentamiento).
RF1.1.2	Actualizar estructura	Permite actualizar los datos de un centro pudiendo deshabilitarse o habilitarse.
RF1.1.3	Eliminar estructura	Permite eliminar un centro penitenciario.
RF1.1.4	Consultar designación del centro	Permite visualizar qué tipos de internos van para ese centro, la designación definida.
RF1.1.5	Registrar órgano cooperante	Permite registrar los órganos cooperantes con un centro.
RF1.1.6	Actualizar órgano cooperante	Permite actualizar los datos de un órgano cooperante del centro.
RF1.1.7	Eliminar órgano cooperante	Permite eliminar un órgano cooperante del centro.
RF1.1.8	Registrar local	Permite registrar todos los locales de un centro penitenciario, especificando al tipo de área que pertenece (área de reclusión, área de servicios, área administrativa y de servicios a funcionarios penitenciarios).
RF1.1.9	Actualizar local	Permite actualizar los datos de un local de un centro penitenciario pudiéndose habilitar o deshabilitar y registrar las

		capacidades afectadas o fuera de uso.
RF1.1.10	Eliminar local	Permite eliminar un local creado en un centro penitenciario.
RF1.1.11	Registrar afectación en un centro	Permite registrar una afectación detectada en un centro penitenciario o local, dando una descripción de la misma.
RF1.1.12	Eliminar afectación	Permite eliminar una afectación en un centro penitenciario.
RF1.1.13	Actualizar afectación	Permite actualizar los datos de una afectación en un centro o local específico.
RF1.1.14	Registrar teléfono del centro	Permite registrar por cada local los teléfonos existentes.
RF1.1.15	Eliminar teléfono del centro	Permite eliminar un teléfono existente en el centro.
RF1.1.16	Actualizar teléfono del centro	Permite actualizar un teléfono del centro.

2.2. Diagramas de Casos de Uso.

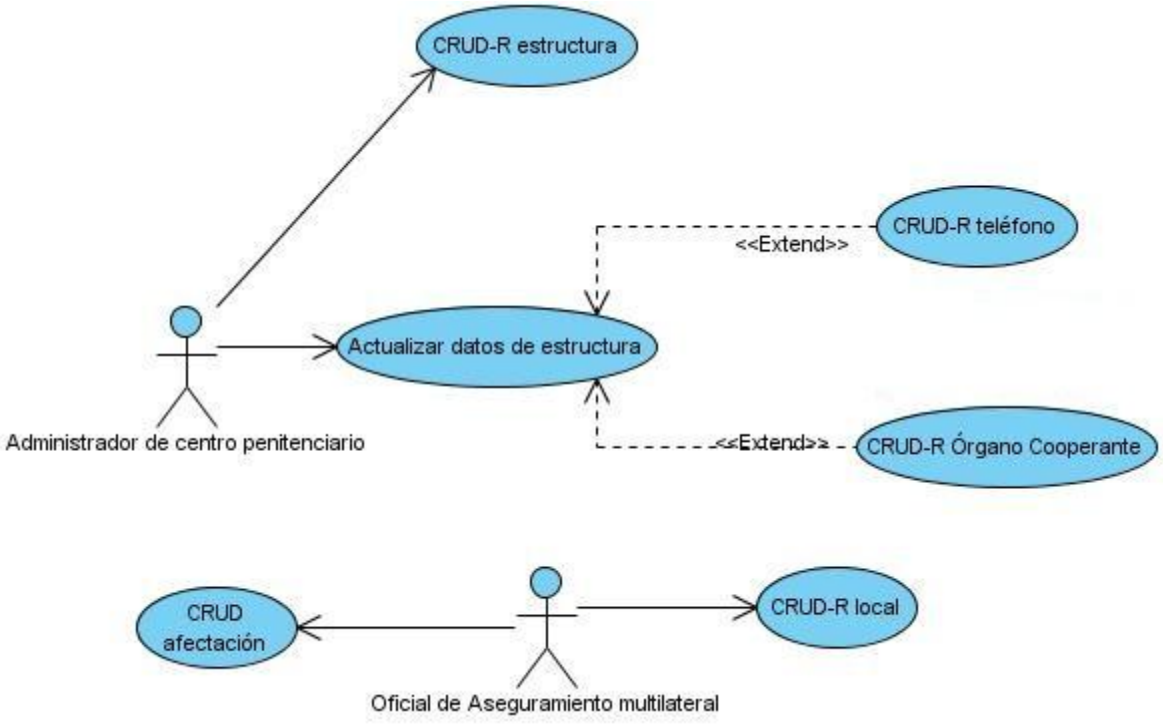


Figura 3 Diagrama de Casos de Uso Control de Capacidades.

El Administrador de centro penitenciario es el encargado de: Registrar y eliminar los centros, Actualizar datos del centro, Registrar, eliminar o actualizar los teléfonos y órganos cooperantes de un centro penitenciario.

Estos administradores pueden ser: Sección de Dirección Provincial, Dpto. de Dirección Nacional y Jefe de unidad.

Por otra parte el oficial de Aseguramiento Multilateral es el encargado de: Registrar, actualizar, eliminar locales y Registrar, actualizar, eliminar afectaciones.

2.3. Arquitectura del Sistema.

La arquitectura de software se define como “la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”. O como la organización o estructura (en un punto dado en el tiempo) de los componentes importantes que interactúan a través de interfaces, los componentes que se compone de sucesivos componentes más pequeños y las interfaces. Una arquitectura puede ser descompuesta recursivamente en partes que interactúan a través de interfaces, las relaciones que conectan a las partes, y las limitaciones para el montaje de piezas. Partes que interactúan a través de interfaces incluyen clases, componentes y subsistemas. [14]

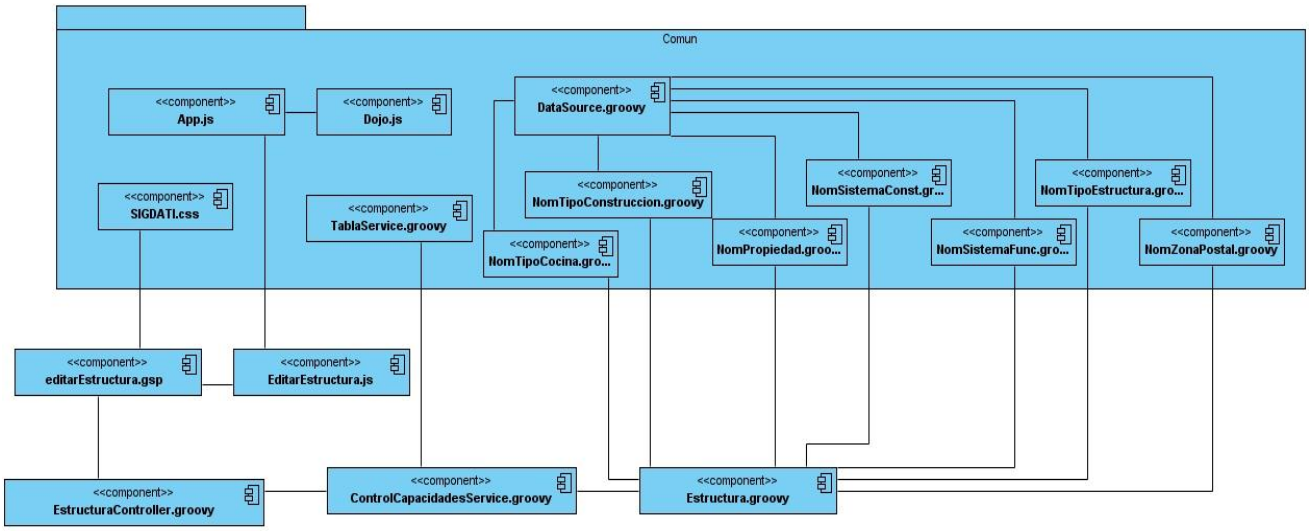


Figura 4 Diagrama de Componentes CU Actualizar Estructura.

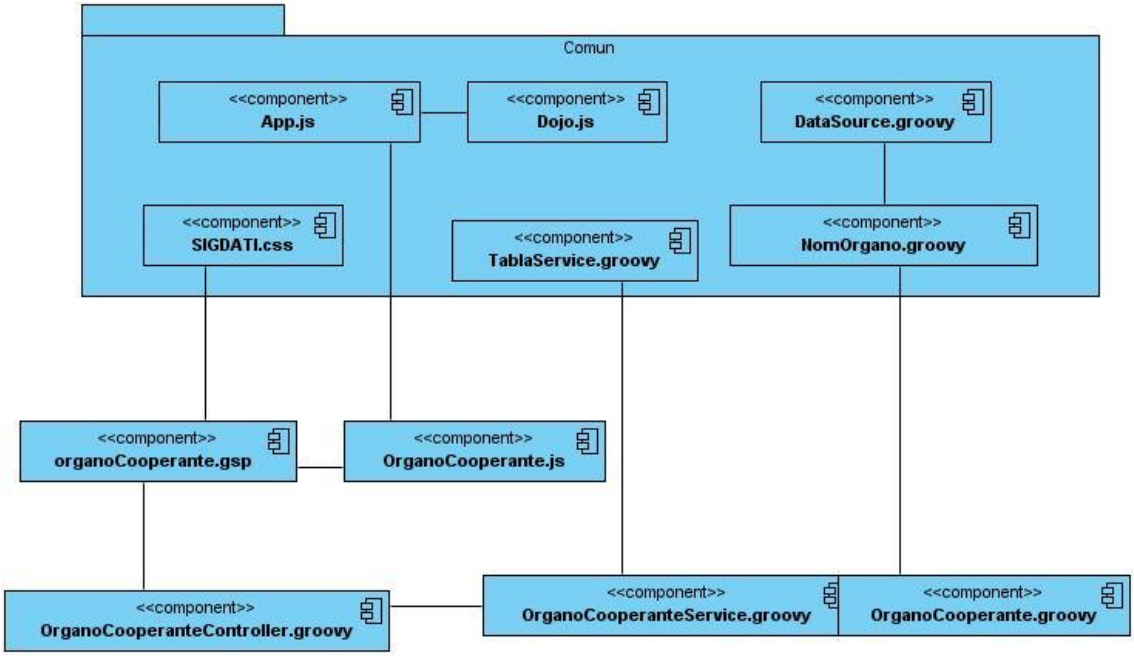


Figura 5 Diagrama de Componentes CU CRUD-R Órgano Cooperante

La arquitectura definida para la construcción del Sistema de Gestión de Datos del Interno de la República de Cuba (SIGDATI) se basará a su vez en el estilo arquitectónico Cliente–Servidor y la propuesta de Arquitectura en Capas que propone Grails.

2.4. Patrones de Diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Estos se caracterizan por estar conformados por un conjunto de elementos, como: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios). [15]

De los patrones que propone el grupo de los patrones GoF o también conocidos Gang of Four se utilizará de forma general el Singleton y quedando como propuesta a su vez el uso de patrón Facade (Fachada). Este último está comprendido entre los patrones de creación y se propone utilizar para separar las clases relacionadas con la lógica de negocio con las clases de la lógica de presentación, evitando así las constantes peticiones al servidor y permitiendo a su vez estructurar las llamadas o peticiones que se realizaran desde el cliente. En el caso de que se decida cambiar la implementación de las clases no es necesario que la capa superior se entere de tal situación, sencillamente la fachada no permitiría que esto ocurra, al no existir una dependencia directa entre las capas.

De los patrones Grasp se utilizarán de forma general los patrones Alta Cohesión, Bajo Acoplamiento, Controller y Front-Controller.

El patrón Controller propone asignar la responsabilidad de controlar el flujo de eventos de un sistema, a clases específicas llamadas controladores. Los controladores no ejecutan las tareas sino que las delegan en otras clases, con las que mantiene un modelo de alta cohesión. Cada clase de Grails promueve el uso de este patrón como parte de los convenios del frameworks. A su vez el patrón Front-Controllers propone utilizar un controlador como el punto inicial de contacto para manejar las peticiones del usuario en una aplicación.

2.5. Diseño de la solución

El Sistema de Gestión de Datos del Interno (SIGDATI) está constituido por un conjunto de sistemas y subsistemas estructurados según las áreas que conforman la actividad penitenciaria. En los subsistemas se agruparan todos los módulos relacionados con un objetivo a fin, permitiendo un mejor aprovechamiento, reutilización y acomodamiento de los mismos. Los módulos serán por su parte los encargados de agrupar todas las funcionalidades comunes, dando cumplimiento en ellos a los requisitos planteados. El Control de Capacidades estará comprendida en un módulo del mismo nombre: “módulo Control de Capacidades” y estará comprendido dentro del Subsistema Registro Legal el cual permitirá administrar los aspectos relacionados con el Control de Capacidades en los Centros Penitenciarios. Dicho módulo está estrechamente relacionado con los módulos Traslado y Egreso pertenecientes al mismo Subsistema, así como también se encuentra relacionado con los módulos Sistema de Colectivos y Ubicación perteneciente a los Subsistemas Sistema Educativo y Área de Ingreso, Observación, evaluación y diagnóstico respectivamente.(Figura. 6)

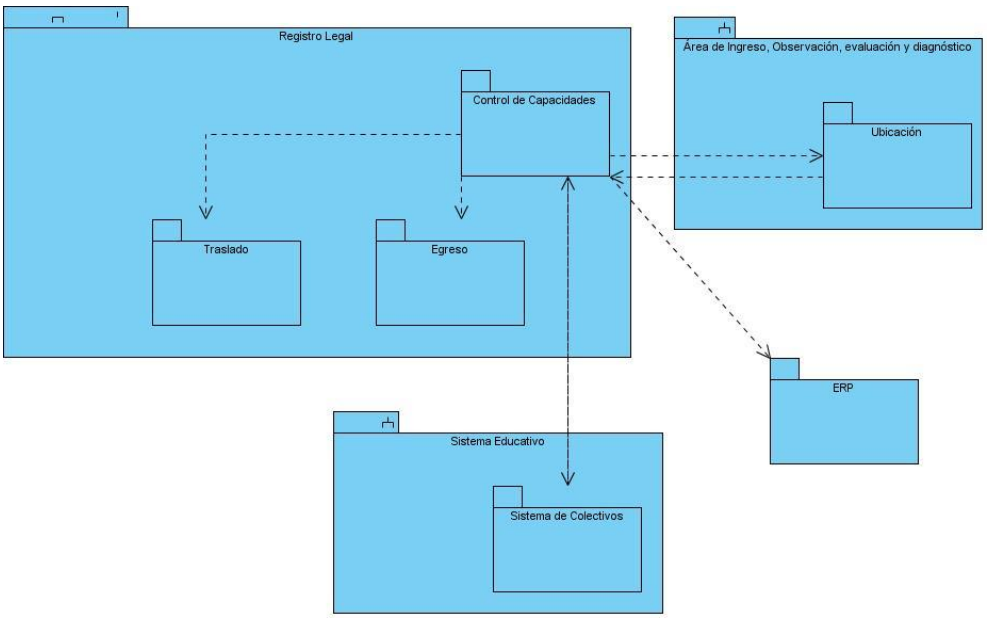


Figura 6 Diagrama de Módulos.

Para una mejor comprensión de la arquitectura propuesta y de la implementación el módulo Control de Capacidades se dividió en varios paquetes los cuales organizan los diferentes casos de uso según las funcionalidades propuestas para lograr un correcto funcionamiento del sistema como se muestra en la **Figura 4**. Dicha estructura por las características de Grails es común para la jerarquía de paquetes que el mismo propone, teniendo en cada paquete de Grails, dígase: Controllers, Views and Layouts, y Domain Class, etc. una estructura semejante. Entre los paquetes definidos se encuentran:

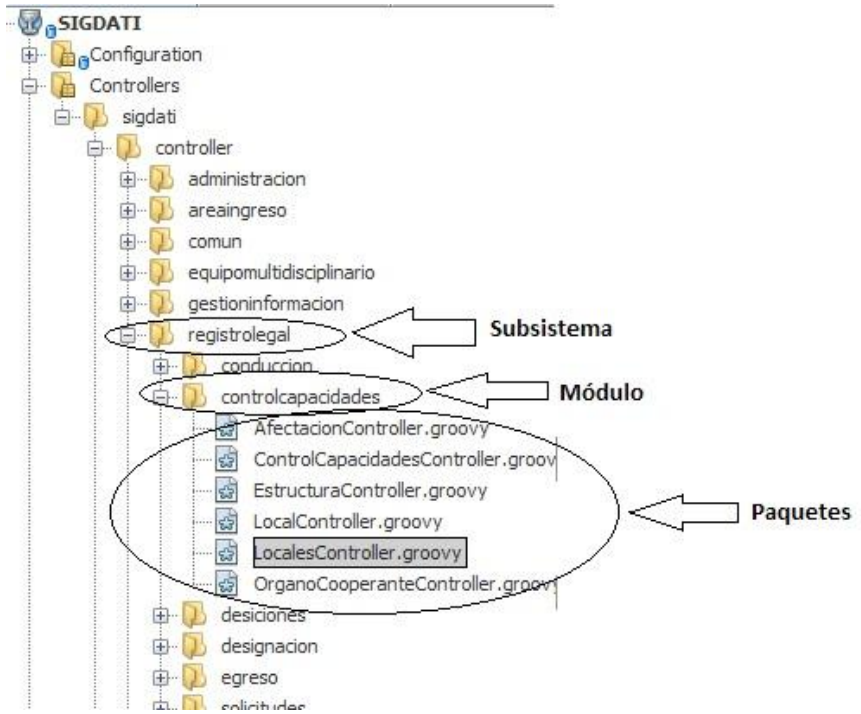


Figura 7 Estructura Módulo Control de Capacidades.

El paquete **Afectación** es el encargado de Gestionar las afectaciones en los Centros penitenciarios.

El paquete **Local** es el encargado de Gestionar los locales existentes en las estructuras pertenecientes al Sistema Penitenciario Nacional.

El paquete **Estructura** es el que contiene todas las funcionalidades referentes a las estructuras asi como tambien debe establecer las relaciones entre las estructuras existentes y los órganos cooperantes con ella.

El paquete **OrganoCooperante** se encarga de gestionar los organos cooperantes que se relacionan con las estructuras ya existentes.

Se definieron a su vez las clases y componentes que estarán comprendidas en los paquetes **comun** y que serán las encargadas de gestionar todas aquellas funcionalidades comunes tanto para los módulos como subsistemas lo que hace más flexible y reutilizable el código, tanto para las clases del dominio como para las de acceso a datos y las que gestionan la lógica de negocio.

Tabla 2 Descripción de funcionalidades.

Nombre	Descripción

registrarEstructura	Permite registrar una estructura en el Sistema Penitenciario.
eliminarEstructura	Permite eliminar una estructura del Sistema Penitenciario.
updateEstructura	Permite actualizar los datos de una estructura del Sistema Penitenciario.
registrarNumeroTelef	Permite registrar un número telefónico.
updateNumeroTelef	Permite actualizar un número telefónico.
eliminarNumeroTelef	Permite eliminar un número telefónico.
registrarLocal	Permite registrar un local perteneciente.
updateLocal	Permite actualizar un local perteneciente a una estructura.
eliminarLocal	Permite eliminar un local perteneciente a una estructura.
guardarOrganoCooperante	Permite registrar un nuevo órgano cooperante o actualizar uno ya existente perteneciente al centro penitenciario.
eliminarOrganoCooperante	Permite eliminar un órgano cooperante del centro penitenciario.
registrarAfectacion	Permite registrar una afectación.
updateAfectacion	Permite actualizar una afectación.
eliminarAfectacion	Permite eliminar una afectación.

2.6. Conclusiones Parciales.

En este capítulo se trató de forma general la arquitectura propuesta para la implementación del módulo Control de Capacidades enfatizando en los estilos arquitectónicos propuestos Arquitectura en Capas y Arquitectura Cliente-Servidor. Se analizó el diseño realizado para la confección de ambos estilos arquitectónicos e integrando estos al frameworks de desarrollo propuesto lo que facilita una mejor comprensión de la solución propuesta para la realización del diseño e implementación del módulo Control de Capacidades del Sistema Penitenciario Nacional.

CAPITULO III: DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN

3. Introducción

En el presente capítulo se abordará todo lo referente al diseño e implementación del módulo Control de Capacidades tomando como base los estudios realizados en capítulos anteriores. Se detallarán las actividades definidas para darle cumplimiento a las funcionalidades propuestas en el capítulo anterior.

3.1. Actividades de Diseño e Implementación del módulo Control de Capacidades.

A continuación se detallan el conjunto de actividades definidas por el SIGDATI, para la construcción de la solución. Las mismas fueron desarrolladas para el diseño e implementación del Módulo Control de Capacidades.

3.1.1. Diseño del Dominio

En el modelo de dominio se captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. El diseño del dominio constituye la entrada principal a las restantes actividades de diseño pues en él se identifican los nomencladores y las clases del dominio, las entidades que serán gestionadas por la capa de lógica de negocio, las que persistirán en la capa de acceso a datos y las que se mostraran en la capa de presentación. La definición del dominio, en especial de las entidades persistentes sirve como una primera aproximación al diseño definitivo del modelo de datos. Grails utiliza en este sector de persistencia potentes patrones como Table Data Gateway y Row Data Gateway los cuales aseguran que existirá una tabla en la base de datos por cada entidad del dominio que se posea y que por cada atributo de la misma que se posea va a haber una fila en la tabla de la base de datos.

El diseño de la solución se enmarca específicamente en los cuatro casos de usos principales los cuales son CRUD[12]-R Estructura, CRUD Afectación, CRUD-R Local y finalmente el CU[13] Actualizar datos de estructura el cual cuenta con dos CU extendidos CRUD-R Teléfono y CRUD-R Órgano Cooperante.

En la **Figura 8** se muestra el modelo del dominio para el caso de uso Actualizar Datos de Estructura.

[12] ***C****rear, **O**btener, **A**ctualizar y **B**orrar*; **C**reate, **R**ead, **U**pdate and **D**elete
[13] ***C****aso de **U**so,***U**se **C**ase

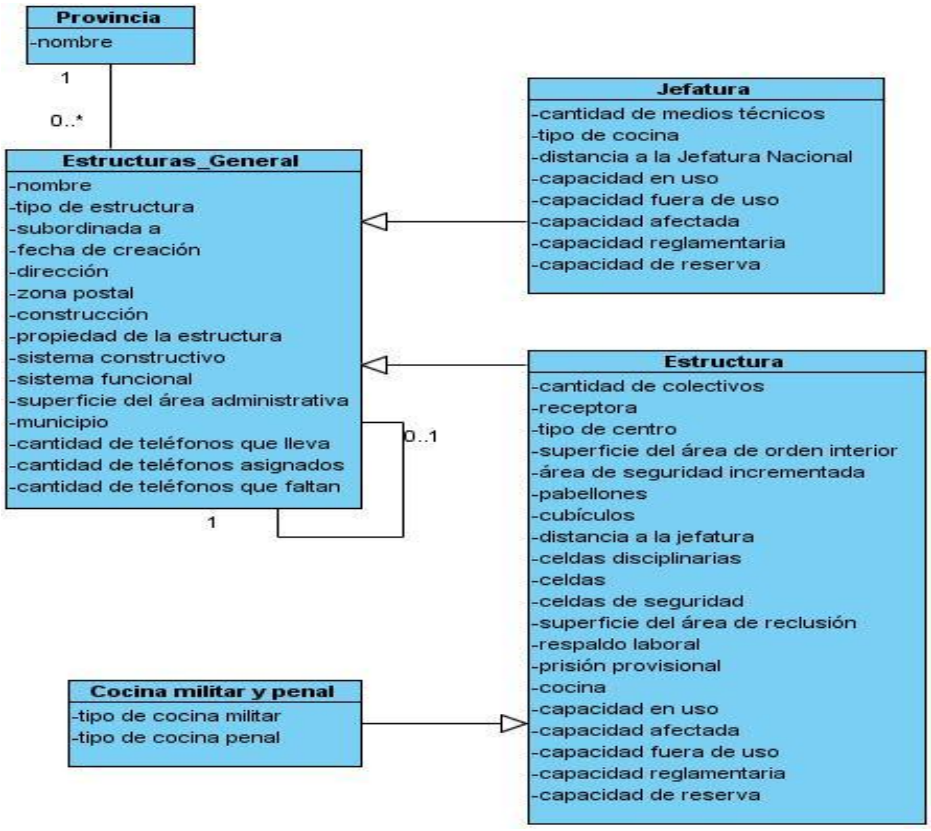


Figura 8 CU Actualizar Datos de Estructura.

3.1.2. Diseño del Modelo de Datos

El diseño de la base de datos debe almacenar toda la información referente a los procesos descritos anteriormente, y permitir a los funcionarios de control penal recuperarla y actualizarla en base a sus peticiones. La utilización de estos objetos de la base de datos es analizada detenidamente y se decide siempre que implique mayor rendimiento en el acceso a los datos y un acoplamiento mínimo al gestor de base de datos. Grails en lo relacionado al acceso a datos propone 3 estructuras fundamentales para el trabajo con la base de datos, create, update and drop. Estas funcionalidades permiten un mejor manejo de la misma además de garantizar que el almacenamiento y recuperación de la información ocurra de manera adecuada y que se cumplan las restricciones identificadas, a través de la integridad referencial.

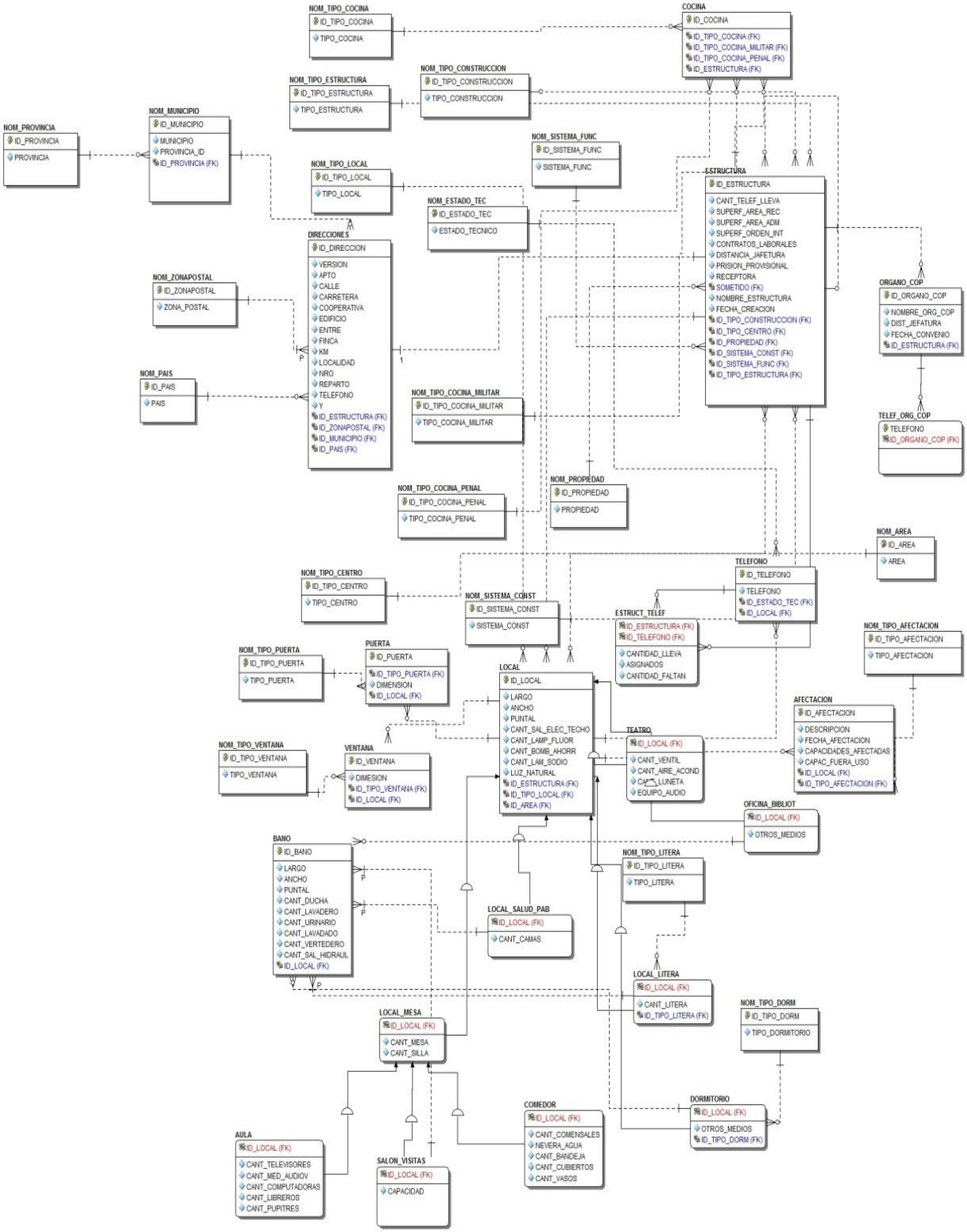


Figura 9 Modelo de Datos.

3.1.3. Descripción de las entidades de dominio significativas.

Tabla 3 Entidad Estructura

Entidad	Descripción
Estructura	Representa los datos de las Jefaturas, los CTEM[14], Asentamientos y centros penitenciarios del Órgano de Prisiones.

Tabla 4 Atributos de Estructura.

Atributos	Tipo	Puede ser nulo
cantidadTelefonosLleva	int	NO
superficieAreaAdministra	float	NO
distanciaJefatura	float	NO
nombreEstructura	String	NO
fechaCreacion	Date	NO
tipoConstruccion	NomTipoConstruccion	NO
propiedad	NomPropiedad	NO
sistemaConstructivo	NomSistemaConst	NO
construcción	NomTipoConstruccion	NO
sistemaFuncional	NomSistemaFunc	NO
tipoEstructura	NomTipoEstructura	NO
direccion	Direccion	NO
zonaPostal	NomZonaPostal	NO
tipoCocina	NomTipoCocina	NO

[14] Campamento de Trabajo y Estudio Municipal

Tabla 5 Entidad Afectación.

Entidad	Descripción
Afectación	Representa las afectaciones de un local o de la estructura.

Tabla 6 Atributos de Afectación.

Atributos	Tipo	Puede ser nulo
tipoAfectacion	NomTipoAfectacion	NO
fechaDeAfectacion	Date	NO
descripcion	String	SI
capacidadesAfectadas	int	SI
capacidadesFueraUso	int	SI

3.1.4. Diseño de la Interfaz de Usuario

En esta actividad se definen las dos principales partes de la capa Web de Grails que van a estar presentes en el módulo: las vistas y los controladores así como el flujo de navegación de las mismas y su relación con los controladores. Las vistas son implementadas mediante GSP o JSP, y por el diseño y arquitectura de Grails estarán compuestas por los Tag Libs y la Templates que propone SiteMesh. Las mismas van a ser las encargadas de mostrar los datos obtenidos en las peticiones del usuario a los controladores así como de enviarles a estos los parámetros obtenidos. Los controladores por su parte gestionan la aplicación mediante la recepción de las acciones emitidas por el usuario a los GSP, interactuando con las mismas o simplemente con las clases de Dominio o delegando las acciones a otro controlador o capa diferente.

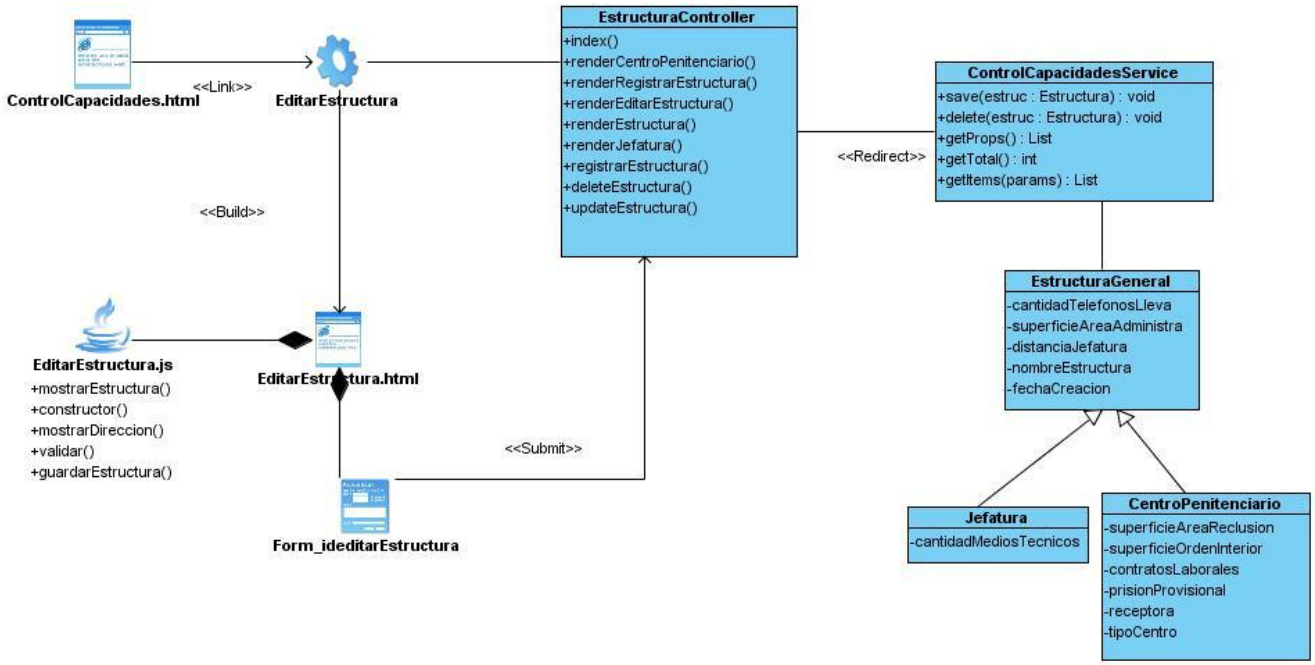


Figura 10 Diagrama de Clases del Diseño para el CU Actualizar Datos de Estructura.

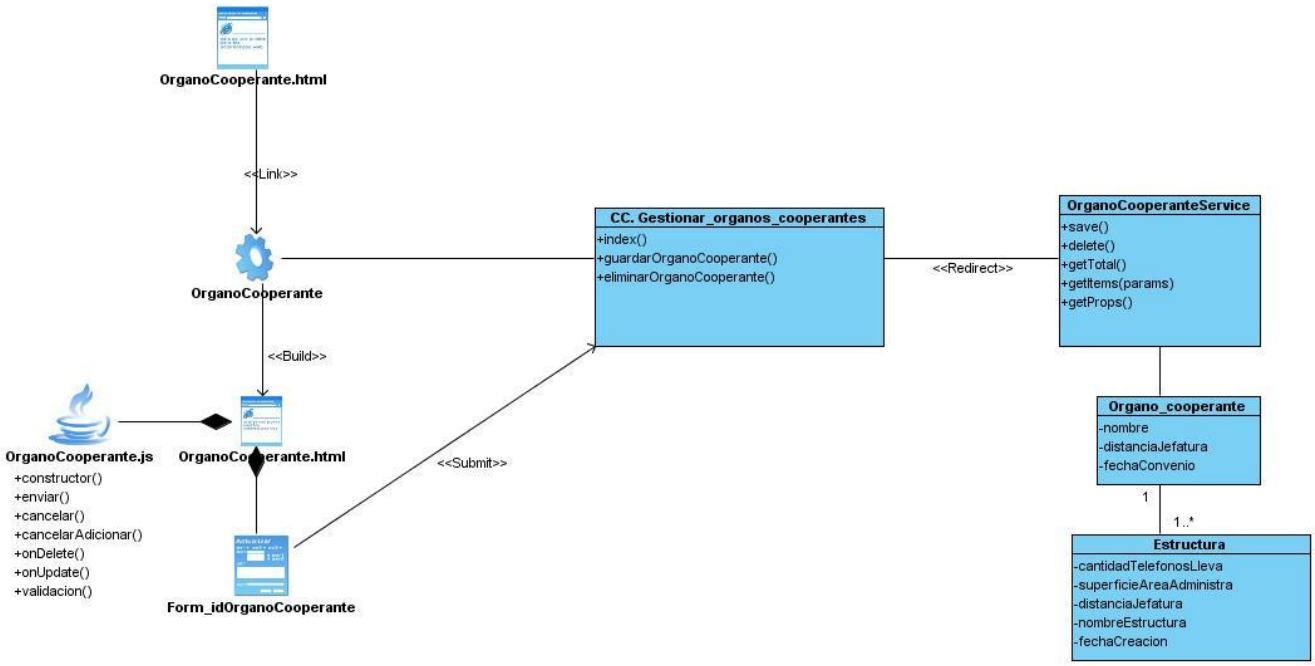


Figura 11 Diagrama de Clases del Diseño CU CRUD-R Órgano Cooperante.

3.1.5. Implementación de las clases del dominio.

Las clases del dominio generalmente no presentan un gran comportamiento en las aplicaciones. Grails por su parte fortalece su estructura haciendo mayor la dependencia de controladores y vistas. Las clases de dominio contendrán la validación de sus atributos mediante expresiones y restricciones propias de Grails, las que facilitan en gran medida el comportamiento de dichos datos tanto en las

vistas como en los métodos a los que puedan estar vinculados en los controladores. Las clases de dominio, las cuales estarán contenidas en ***SIGDATI/domain/sigdati/domain***, contendrán a su vez las relaciones existentes entre ellas siguiendo la estructura del modelo de relaciones que propone Grails.

A continuación se muestra la implementación de una clase de dominio con la validación de sus atributos así como las relaciones existentes con otras clases.

```
class Estructura {
    int cantidadTelefonosLleva
    float superficieAreaAdministra
    float distanciaJefatura
    String nombreEstructura
    Date fechaCreacion
    NomTipoConstruccion tipoConstruccion
    NomPropiedad propiedad
    NomSistemaConst sistemaConstructivo
    NomSistemaFunc sistemaFuncional
    NomTipoEstructura tipoEstructura
    Direcciones direccion
    NomZonaPostal zonaPostal
    NomTipoCocina tipoCocina

static hasMany = [sometidoA: Estructura, locales: Local, organosCooperantes: OrganoCooperante]
    static constraints = {
cantidadTelefonosLleva  nullable:true
        superficieAreaAdministra nullable:true
        distanciaJefatura nullable:true
        tipoConstruccion nullable:true
        propiedad nullable:true
        sistemaConstructivo nullable:true
        sistemaFuncional nullable:true
        direccion nullable:true
        zonaPostal nullable:true
        tipoCocina nullable:true
    }
```

```
}
```

3.1.6. Implementación de las vistas.

Las vistas son responsables de hacer la interfaz de usuario y se implementan mediante GSP, que es la tecnología de vistas que utiliza Grails y es una extensión de JSP que puede incluir código Groovy. Las GSP tienen extensión .gsp y se encuentran dentro del directorio ***SIGDATI/views/registrolegal/controlcapacidades***. Las vistas van a ser las responsables de contener todas la funciones de Dojo, las cuales se crean y se definen en cada GSP, las mismas son las encargadas a su vez de invocar las llamadas a las funciones de JavaScript las cuales van a estar contenidas en los archivos JS, los cuales se definieron según los métodos y funcionalidades comunes que agrupan. Su relación con los controladores se representa de forma predeterminada empleando las definiciones de Grails al cada GSP poder contener una clase controladora que la respalde.

3.1.7. Implementación de los controladores.

Luego de la implementación y creación de las clases de dominio se implementan los controladores mediante el uso de Scaffolding. El Scaffolding no es más que la creación de artefactos como los controladores y vistas que satisfacen una serie de requisitos, como por ejemplo la realización de operaciones CRUD a una clase de dominio, la autenticación, búsqueda y las pruebas unitarias. Los controladores son los encargados de gestionar y coordinar el flujo lógico de la aplicación mediante la recepción de las acciones del usuario. Reciben peticiones de los usuarios y actúan sobre ellas. Pueden interactuar directamente con una clase de dominio para realizar una operación de CRUD, re direccionar al usuario a un GSP diferente, delegar una acción a un actor diferente (otro controlador o una clase de servicio), y preparar y enviar la respuesta de nuevo a la vista, siempre creando un nuevo controlador para cada solicitud. Su creación va a estar definida por la sintaxis ***grails create-controller***, se ubica en ***SIGDATI/controllers/sigdati/controller/registrolegal/controlcapacidades*** manejando todo el flujo de trabajo de la capa Web Layers junto a las vistas.

3.1.8. Implementación de la capa de acceso a datos.

La implementación de la capa de acceso a datos en Grails se basa principalmente en la configuración del archivo ***DataSource.groovy***, en el cual se define el gestor de base de datos a usar así como su configuración. El uso de GORM, como una herramienta ORM evita el trabajo directo con el gestor de base de datos y sus entidades, trabajando a su vez con objetos en su lugar, lo que simplifica en gran medida el trabajo con Hibernate y elimina cualquier configuración externa. Grails es compatible con tres entornos deforma predeterminada: desarrollo, prueba y producción. Dichos

entornos no solo facilitan el manejo de los datos sino que además posibilitan tratar a cada entorno de manera diferente: create-drop, create, update, blank, posibilitando hacer el trabajo de forma controlada y segura.

```
dataSource {
    pooled = true
    driverClassName = "oracle.jdbc.driver.OracleDriver"
    dialect = org.hibernate.dialect.OracleDialect
    username = "proyecto"
    password = "proyecto"
}
hibernate {
    cache.use_second_level_cache=true
    cache.use_query_cache=true
    cache.provider_class='net.sf.ehcache.hibernate.EhCacheProvider'
    //config.location = "file:/hibernate.cfg.xml"
}
// environment specific settings
environments {
    development {
        dataSource {
            //dbCreate = "update" // one of 'create', 'create-drop','update'
            //url = "jdbc:hsqldb:mem:devDB"
            url = "jdbc:oracle:thin:@10.36.13.154:1521:SIGI"
            //jdbc:oracle:thin:@<host>:<port>:<sid>
        }
    }
    test {
        dataSource {
            //dbCreate = "update"
            //url = "jdbc:hsqldb:mem:testDb"
            url = "jdbc:oracle:thin:@10.36.13.154:1521:SIGI"
        }
    }
    production {
        dataSource {
            //dbCreate = "update"
            // url = "jdbc:hsqldb:file:prodDb;shutdown=true"
            url = "jdbc:oracle:thin:@10.36.13.154:1521:SIGI"
        }
    }
}
```

Figura 12 Archivo de configuración de la base de datos.

GORM trae implementado a su vez métodos dinámicos como ***save(), delete(), refresh() e ident()*** así como métodos estáticos como ***count(), exist(), find(), findAll(), findBy*() y findWhere(),*** los cuales facilitan en gran medida el trabajo con la base de datos.

3.1.9. Implementación de la lógica en el cliente.

Para esta actividad se implementan en los archivos JS contenidos en ***SIGDATI/Web-application/js/sigdati/registrolegal/controldecapacidades*** las validaciones existentes del lado del cliente, así como el comportamiento de cada componente Dojo para los diálogos de información o error. El lanzamiento de peticiones desde el cliente al servidor también está incluido en esta actividad mediante el uso de tecnología AJAX, JSON y JSON-RPC y para el uso de componentes de bibliotecas Javascript Dojo Toolkit. Un ejemplo de validaciones hechas del lado del cliente contenidas en el fichero **editarEstructura.js** es la correspondiente a la funcionalidad Registrar Estructura en el Sistema Penitenciario como se muestra a continuación:

```
if(!dojo._hasResource["sigdati.registrolegal.controldecapacidades.EditarEstructura"]) {
dojo._hasResource["sigdati.registrolegal.controldecapacidades.EditarEstructura"] = true;
   dojo.provide("sigdati.registrolegal.controldecapacidades.EditarEstructura");
   dojo.declare("sigdati.registrolegal.controldecapacidades.EditarEstructura", null, {
constructor: function() {
 dojo.connect(dojo.byId("estructura.aceptar.id"),"onclick",this, ostrarEstructura");
            dojo.connect(dojo.byId("OrganoCooperante.direccion.id"), "onclick", this,
"mostrarDireccion");
      },
      mostrarEstructura : function(){
         if (this.validar()){
           dijit.byId("estructura.estructuraPrincipal.id").href =
"/SIGDATI/controlCapacidades/renderEstructura";
           dijit.byId("estructura.estructuraPrincipal.id").refresh();
}
       },
       mostrarDireccion:function(){
          sigdati.app.mostrarDialog({
             title: "Dirección",
             href: "/SIGDATI/organoCooperante/mostrarDireccion",
method: "post",
             onLoad: function(){
 dojo.connect(dojo.byId("OrganoCooperante.direccion.aceptar.button.id"), "onclick",
function(){
//validar la ventana emergente. Si pasa algo return
dojo.xhrPost({
                        sync : true,
                        handleAs:"json",
                        form: dojo.byId("OrganoCooperante.direccion.form.id"),
                        content : {
                        },
                        load : function(data) {
document.getElementById("OrganoCooperante.direccion.id").value= data.direccion;
document.getElementById("tiene_direccion.id").value = "yes";
document.getElementById("OrganoCooperante.direccion.id").disabled = true;
                           sigdati.app.closeDialog();
                        },
                        error: function(data){
alert ("Ha ocurrido un error interno")
                        }
                     });
                });
             }
          });
       },
```

3.1.10. Diseño de la funcionalidad Actualizar Estructura

La funcionalidad Actualizar Estructura es la encargada de regir el trabajo con las estructuras en el módulo Control de Capacidades. La solución permite inicialmente crear una estructura con los campos básicos, que son los de la interfaz Registrar Estructura y los demás campos se llenan en el actualizar estructura. (Figura. 13)

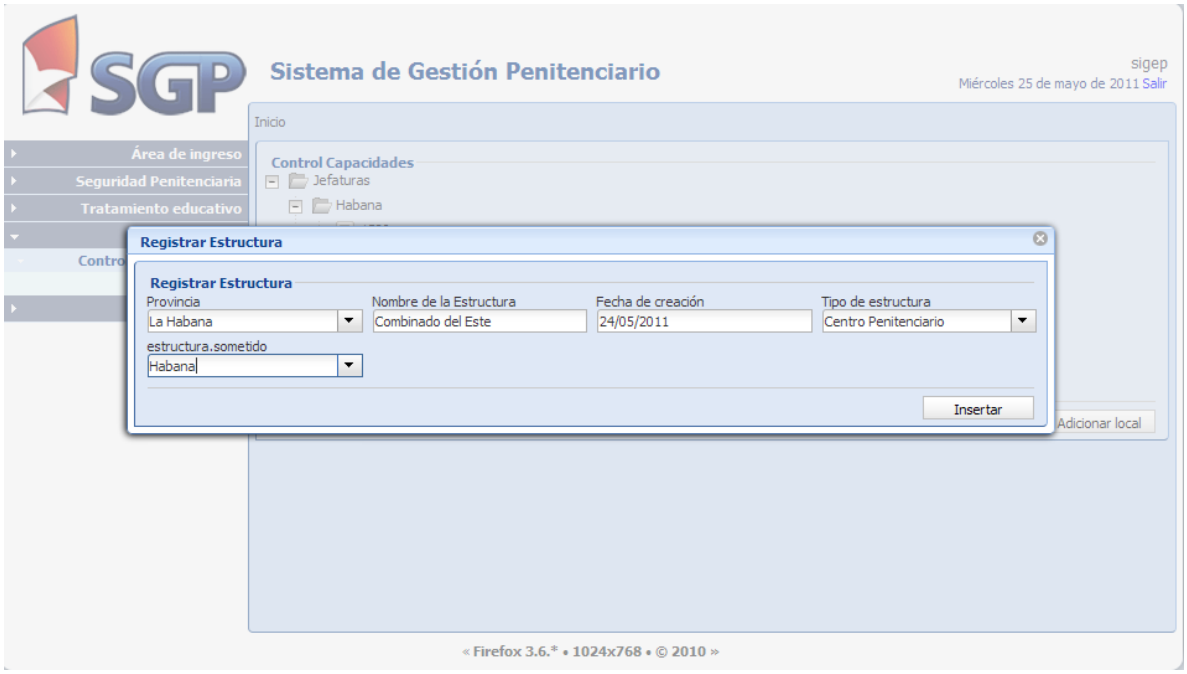


Figura 13 Interfaz Registrar Estructura.

Una vez creada se procede a seleccionar las jefaturas y sus centros subordinados mostrando así en el Árbol jerárquico las estructuras pertenecientes al centro y a la jefatura seleccionada. (Figura.14)

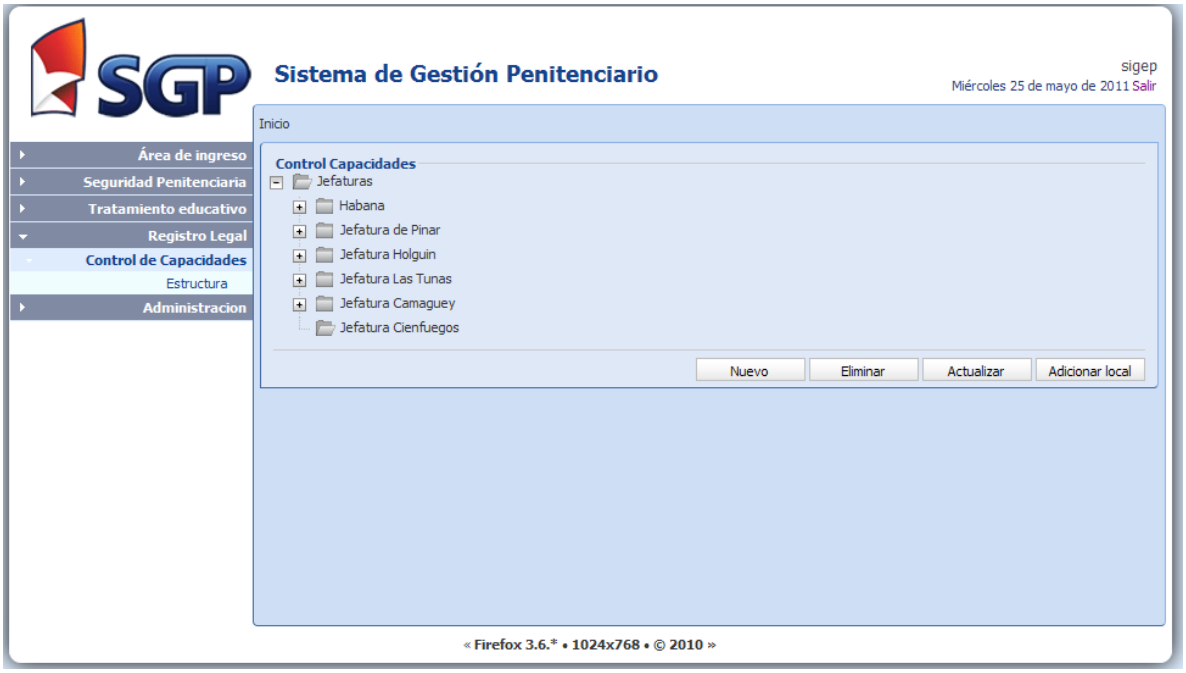


Figura 14 Interfaz Principal para seleccionar una estructura.

Dichas estructuras pueden ser de diferentes tipos: Jefaturas Provinciales y Centros Penitenciarios, en los que se incluyen los CTEM, Asentamientos. Los tipos de estructura no deben variar según concesiones que se han hecho con el cliente, y aunque las mismas pueden estar sujetas a cambios en un futuro tanto la flexibilidad del sistema como la arquitectura de Grails aseguran que dichos cambios no afecten de forma directa la ejecución de la solución. La existencia de algún nuevo tipo de estructura en la capa de acceso a datos solo conllevará la inserción de la misma en el gestor así como leves cambios en las clases de dominio, al Grails tenerlas estrechamente relacionadas en la capa Domain Layers. Al seleccionar la estructura a actualizar el sistema procede a mostrar la interfaz Actualizar Datos de Estructura en el cual se pueden actualizar datos en el caso que ya estén registrados y en caso que no se procede a registrarlos en la estructura ya creada (Figura. 10). En esta Actualización están contenidos en caso de que sea necesario el Caso de Uso extendido CRUD-R Órgano Cooperante. (Figura. 15)

Figura 15 Interfaz Actualizar Datos de Estructura.

Figura 16 Interfaz Registrar Órgano Cooperante.

En el controlador el Método **updateEstructura** es el encargado de guardar los datos actualizados de la estructura.

3.2. Conclusiones Parciales.

Al concluir el presente capítulo se detallaron de forma más específica todas las actividades definidas por el SIGDATI para el diseño e implementación del Módulo Control de Capacidades del Sistema Penitenciario Nacional, mediante el uso de diagramas, fragmentos de códigos e imágenes.

CONCLUSIONES

- Como resultado del trabajo realizado se diseñó e implementó el módulo Control de Capacidades del Sistema Penitenciario Nacional a partir de los requerimientos de software establecidos con el cliente y haciendo uso de la arquitectura y las tecnologías definidas por el proyecto.
- Se logró la integración del módulo Control de Capacidades a la plataforma SIGDATI
- Se realizó un estudio de las tecnologías y herramientas a utilizar en el diseño e implementación del módulo así como del análisis de los requisitos de software y del modelo de negocio correspondientes.
- Las actividades de diseño e implementación fueron ejemplificadas a través de diagramas de clase y fragmentos de código fuente cumpliendo así con los objetivos propuestos para este trabajo.
- Se logró erradicar en gran medida los problemas de usabilidad que presenta SACDEP.

RECOMENDACIONES.

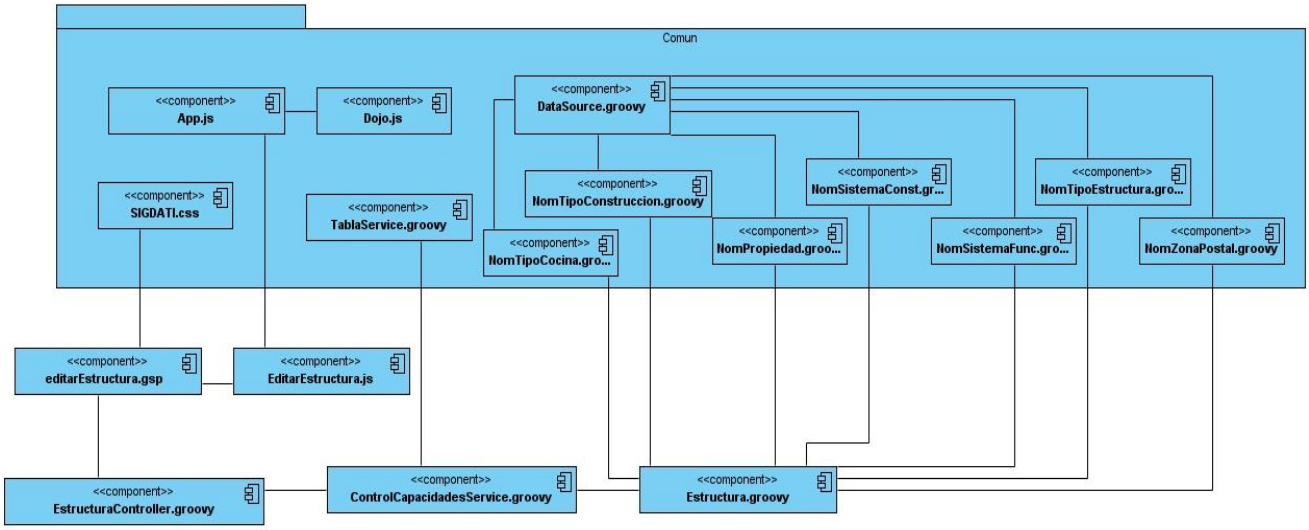
- Comprobar el rendimiento del módulo frente a bases de datos de grandes volúmenes de información para validar que el diseño propuesto satisface tales condiciones, puesto que actualmente el sistema está probado para una cantidad limitada de datos que no ponen al límite la solución propuesta.

BIBLIOGRAFÍA.

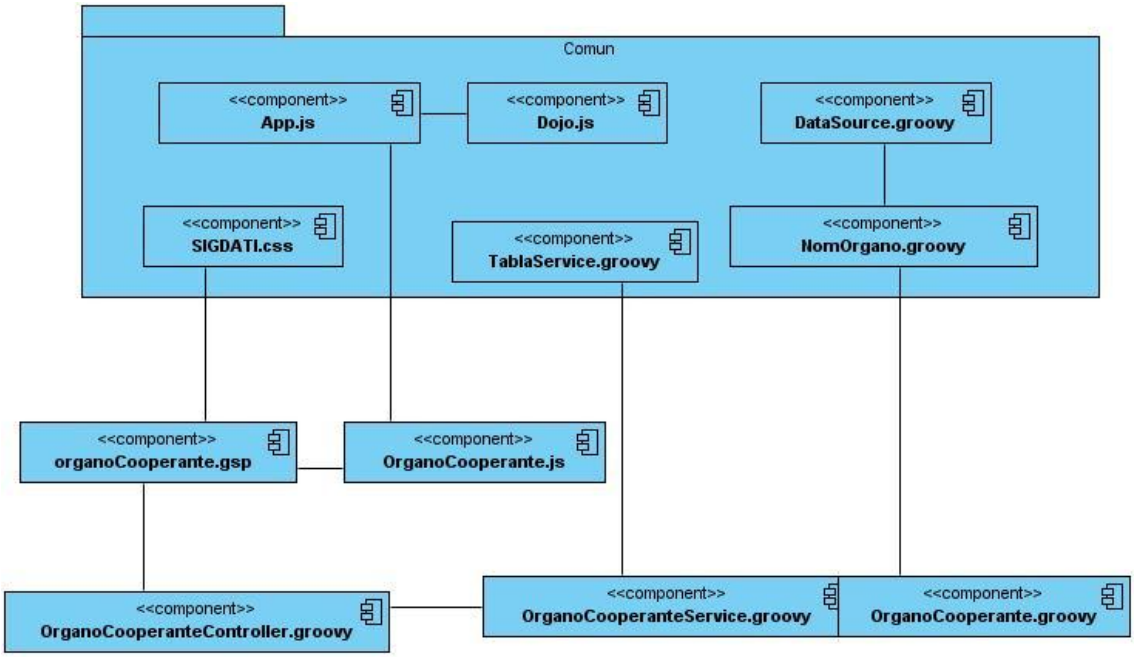
1. **Zequeira Peña, Dr. Tte.Cnel. Alfonso y Céspedes Quesada, Lic. 1erTte. Aimeé.** *Vocabulario Jurídico Penitenciario.* Ciudad de la Habana : MININT, 2005.
2. **Código Penal Cubano.** *Ley nro. 62.* República de Cuba : s.n.
3. **Osvaldo Vergara Castañeda y Alejandro Rojas Santana**, Análisis del Módulo de Control de Capacidades del proyecto Sistema de Gestión Penitenciaria en Cuba, Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, 2010
4. **Rational Software Corporation.** *RUP. "Rational Unified Process".* 2003.
5. **Visual Paradigm International.** Visual Paradigm. [En línea] http://www.visual-paradigm.com..
6. **Oracle Corporation and/or its affiliates.** NetBeans. [En línea] http://netbeans.org/index.html.
7. Sitio Web Apache Tomcat. [En línea] http://tomcat.apache.org/
8. **ORACLE, CORPORATION.** *Oracle Database Documentation Library.* 2005.
9. **Json.org.** JSON. *Java Script Object Notation.* [En línea] http://www.json.org/json-es.html.
10. **Sun Microsystms Inc.** Javascript. [En línea] http://www.javascript.com/.
11. **The Dojo Foundation.** DojoTolkit. [En línea] http://www.dojotoolkit.org/.
12. **Open Source Software Engineering Tools.**Subversion. [En línea] http://subversion.tigris.org/
13. **SpringSource.** Grails. *Grails.* [En línea] SpringSource, 2009. http://www.grails.org.
14. **Jacobson, Ivar, Booch, Grady y Rumbauch, James. 2000.** *El Proceso Unificado de Desarrollo del Software. .* Madrid : s.n., 2000.
15. **Larman., Craig.** *UML Y Patrones. Introducción al análisis y diseño orientado a objeto. .* Mexico: s.n.
16. **Glen Smith, Peter Ledbrook.***Grails in Action.* United States of America : Manning Publications Co., 2009. ISBN 978-1-933988-93-1.
17. **LADD., SETH.** *Expert Spring MVC and Web Flow. .* 2006.
18. **Nacho Brito.** Manual de desarrollo web con Grails. [En línea] http://www.manual-de-grails.es

ANEXOS

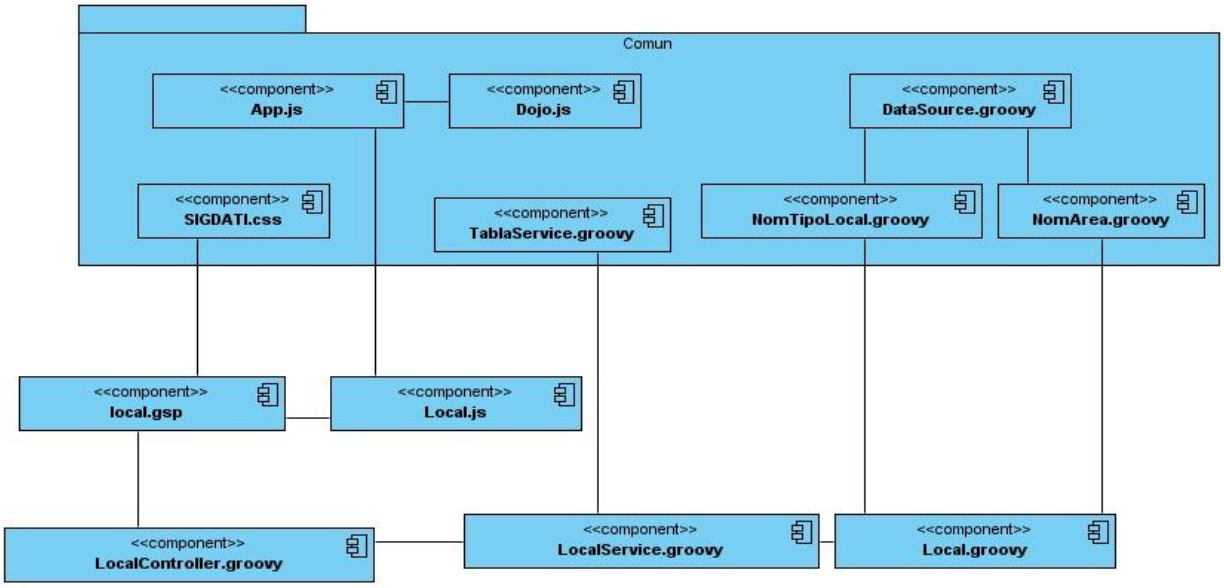
Anexo 1 Diagrama de Componentes CU Actualizar Estructura.



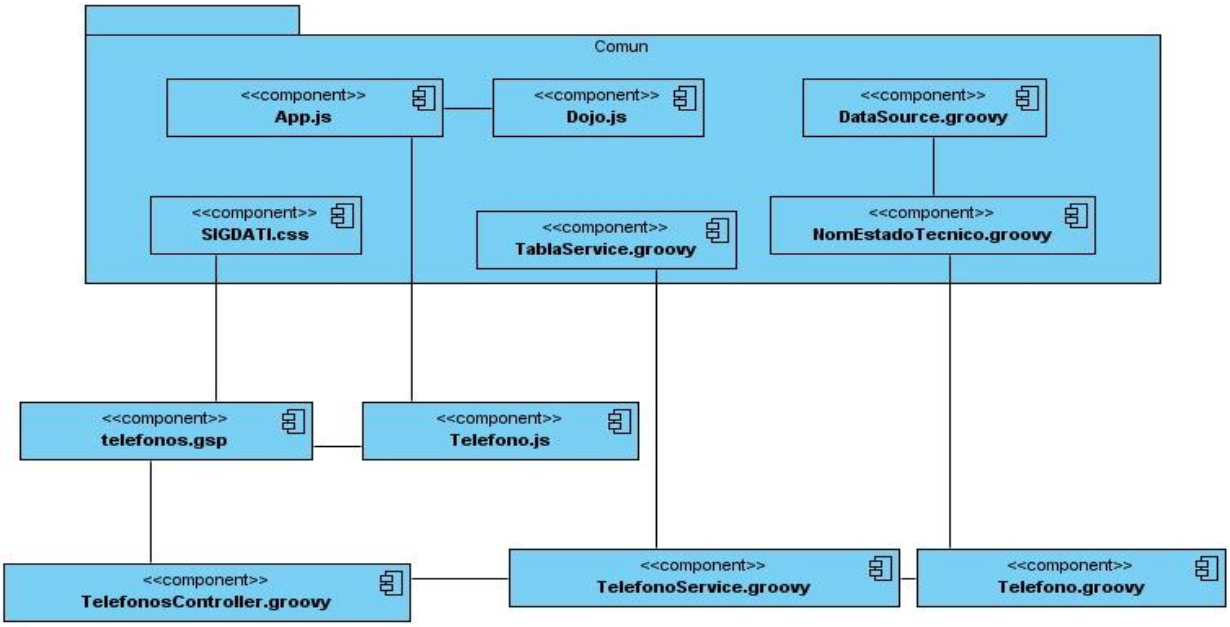
Anexo 2 Diagrama de Componentes CU CRUD-R Órgano Cooperante.



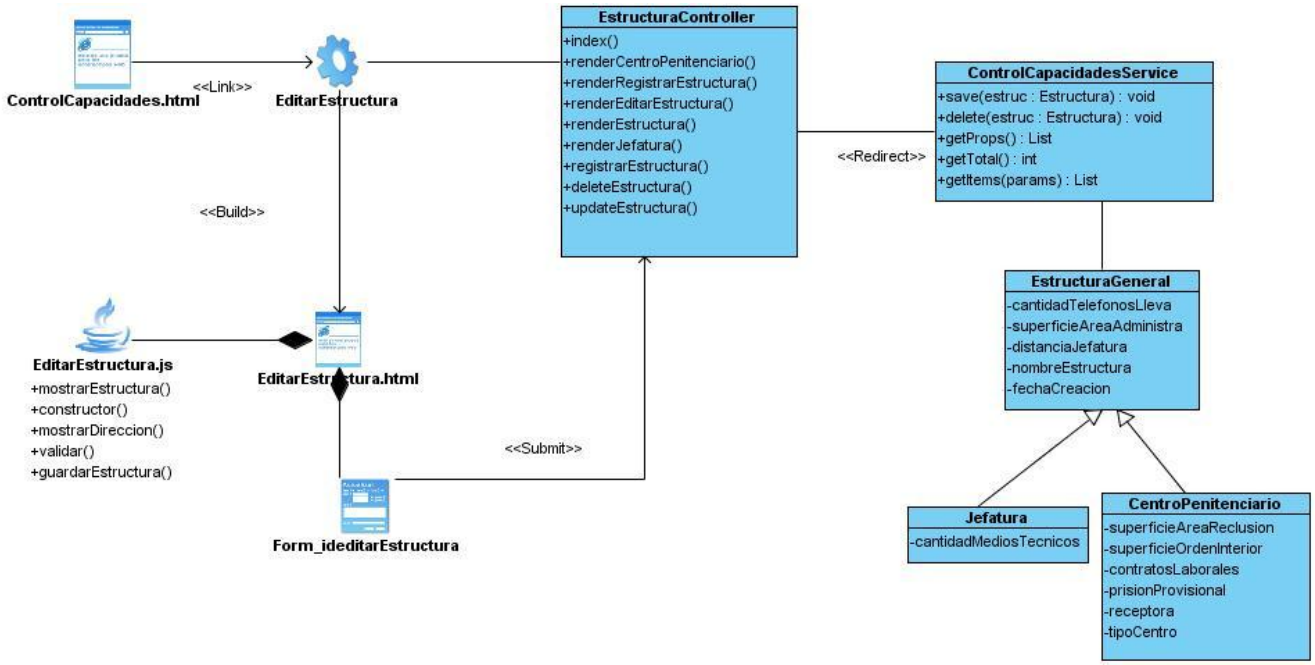
Anexo 3 Diagrama de Componentes CU CRUD-R Local.



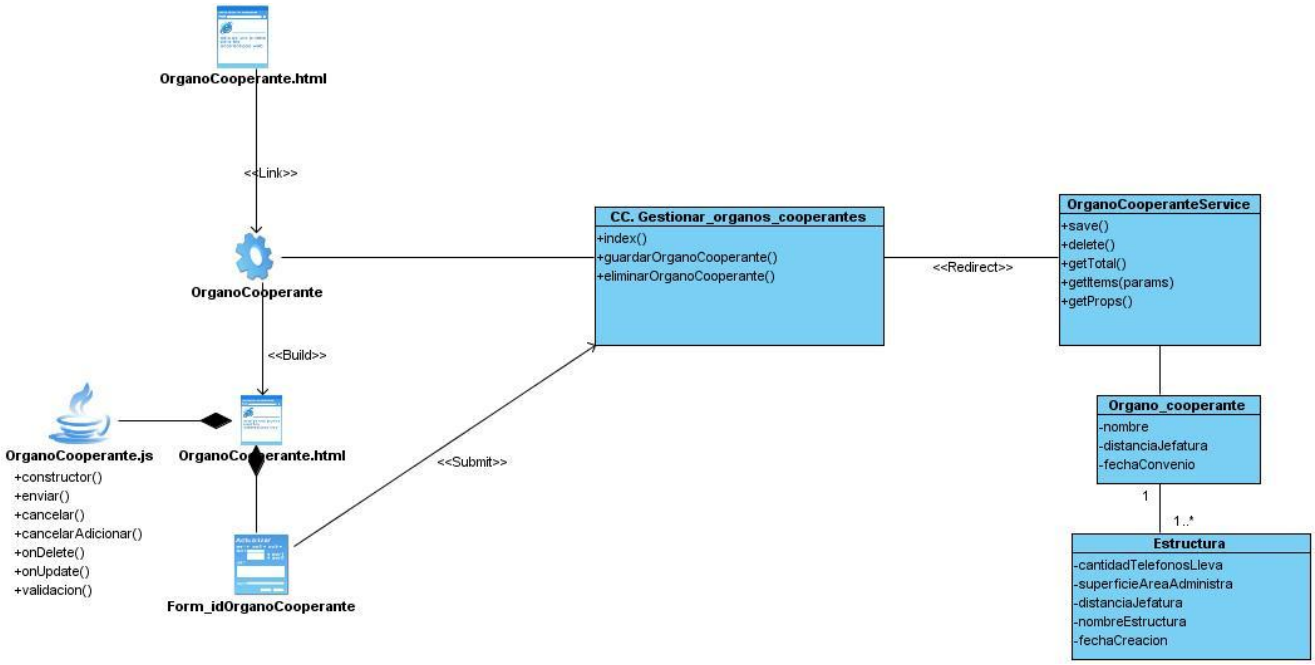
Anexo 4 Diagrama de Componentes CU CRUD-R Teléfono.



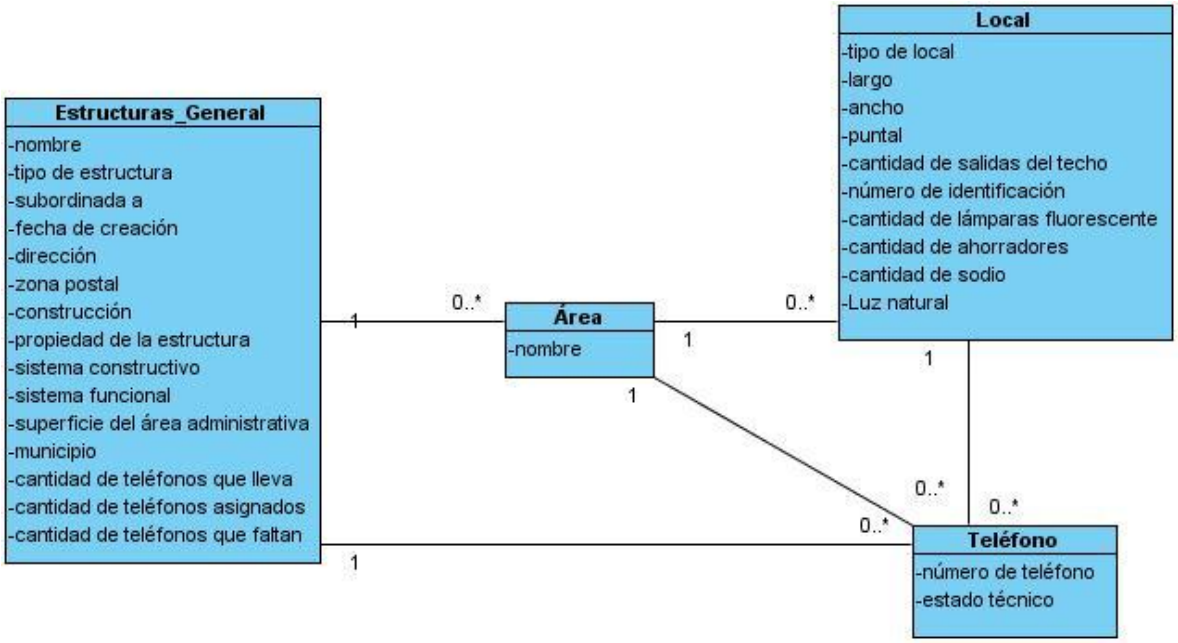
Anexo 5 Diagrama de Clases del Diseño CU Actualizar Estructura



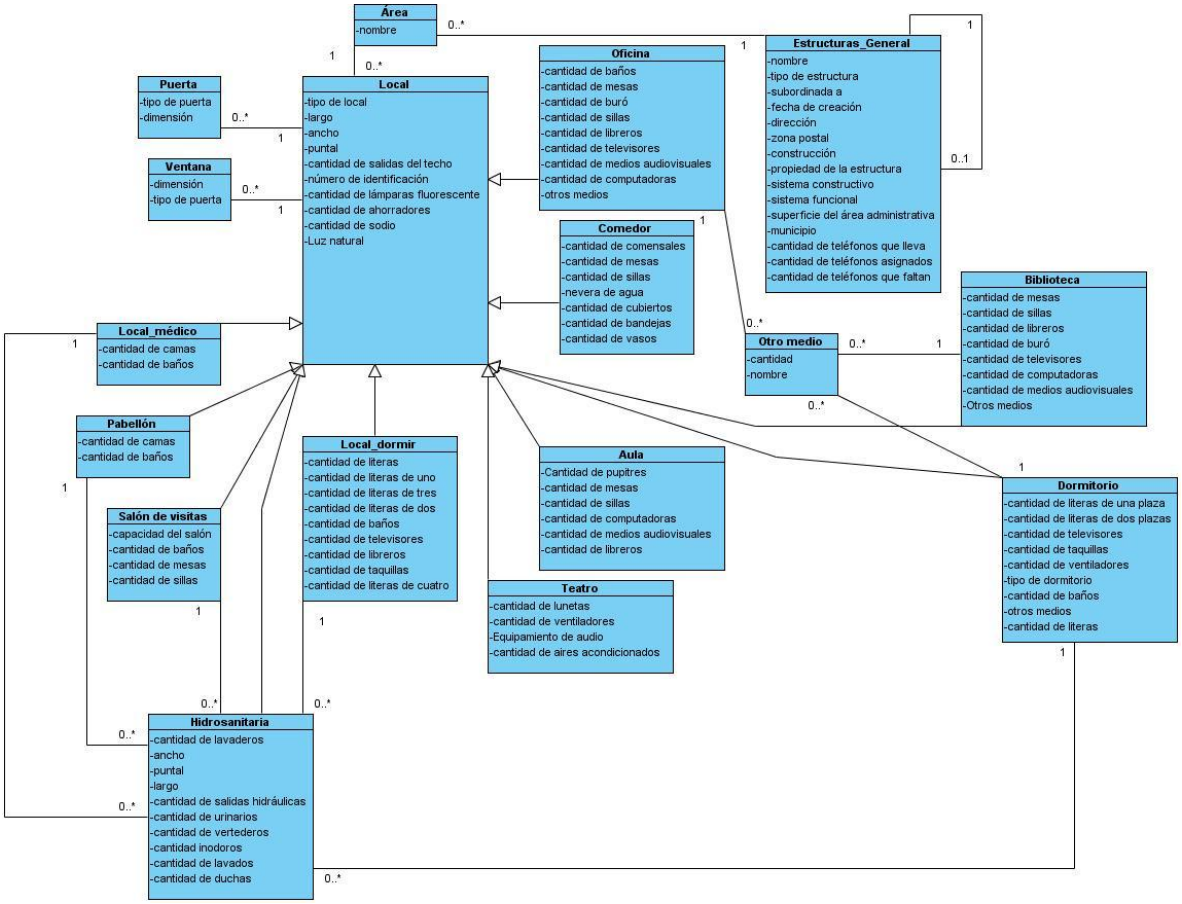
Anexo 6 Diagrama de Clases del Diseño CU CRUD-R Órgano Cooperante.



Anexo 7 Modelo de dominio CU CRUD-R Teléfono.



Anexo 8 Modelo de dominio CU CRUD-R Local.



Anexo 8 Modelo de dominio CU CRUD-R Local.