Universidad de las Ciencias Informáticas

TESIS EN OPCIÓN AL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Sistema de Gestión Policial. Especificación de la Arquitectura

Autor: Frank Ernesto Verdecia Rodríguez.

Tutor: Lic. Karel Osorio Ramírez.

Ciudad de la Habana, febrero de 2009.

"Año del 50 aniversario del triunfo de la Revolución"

Declaración de Autoría

Declaro ser autor de la presente tesis y se le reconoce a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma con carácter exclusivo.

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

_____ _____

Frank E. Verdecia Rodríguez. Lic. Karel Osorio Ramírez.

Dedicado a mi mamá, mi papá y mí hermana Rumy.

Por siempre apoyarme y sentirse orgullosos de mí.

Agradecimientos

En primer lugar agradecer a mi tutor Karel, que sin el no hubiese sido posible que se terminara todo esto.

A Yaneisy por siempre estar apoyándome y ser una buena amiga.

A mi familia por siempre apoyarme en todo.

A mis amigos los cuales no voy a poner nombres para que no se me quede ninguno.

A mis profes de la primaria, de la secundaria, del pre y de la universidad, de los cuales aprendí mucho y les debo todo lo que se.

A la revolución por darme la posibilidad de estudiar en esta universidad y a Fidel Castro por su genial idea de crear esta escuela.

Resumen

El Sistema de Gestión Policial (SIGEPOL) es un sistema informático para la automatización de los distintos procesos que ocurren en una dependencia policial de la República Bolivariana de Venezuela.

En el presente trabajo se especifica la arquitectura base sobre la cual se desarrolló el sistema SIGEPOL, esta define la organización lógica y física, la definición de los mecanismos de seguridad y auditoría, las distintas tecnologías y herramientas de desarrollo a utilizar, además de brindar un flujo de trabajo permitiendo una óptima interacción entre los desarrolladores y realizar una implementación en el menor tiempo posible.

Índice

Introducción

El MPPRIJ[1] de la República Bolivariana de Venezuela, atendiendo a su misión institucional de garantizar la seguridad ciudadana, el 6 de noviembre del 2001, crea la Ley de Coordinación de Seguridad Ciudadana en la que establece las bases para el proyecto Sistema Nacional de Registro Delictivo, Emergencias y Desastres (SINARDED), con la finalidad de almacenar y procesar la información de los distintos sucesos policiales que ocurren a nivel nacional diariamente (1); sin embargo este sistema no fue implementado en su totalidad, por lo que todos sus objetivos no fueron cumplidos y en estos momentos es empleado únicamente en la Dirección de Coordinación Policial.

Actualmente las áreas de las dependencias policiales no disponen de un proceso unificado a nivel nacional para la actuación policial. Existen diferencias en el proceso de levantamiento de denuncias, reseña del ciudadano y planificación de operativos policiales de las distintas dependencias policiales.

El MPPRIJ promueve la formulación y puesta en marcha del Sistema de Gestión Policial (SIGEPOL), que pretende unificar y compartir la información relacionada con denuncias, reseñas y operativos policiales que se registran en las dependencias de las policías estadales y municipales, así como intercambiar información con el CICPC[2] y servir de fuente de información al CTAISC[3]. Todo esto con la finalidad de ayudar a combatir el delito en la República Bolivariana de Venezuela y alcanzar la tranquilidad ciudadana del pueblo en el marco del estricto cumplimiento de las regulaciones legales.

Uno de los principales mecanismos que ha adoptado la industria del software en los últimos años para combatir la complejidad inherente a los grandes sistemas informáticos; consiste en definir y construir sus cimientos, o sea su arquitectura, con el objetivo de aportar elementos que ayuden a la toma de decisiones y a la vez proporcionar conceptos y un lenguaje común que permita la comunicación entre todos los desarrolladores del sistema.

La ausencia de patrones de arquitectura o arquitecturas base dificulta la construcción y mantenimiento de sistemas informáticos de gran complejidad, que sean capaces de

[1] Ministerio del Poder Popular para Relaciones Interiores y Justicia.
[2] Cuerpo de Investigaciones Científicas Penales y Criminalística.
[3] Centro de Tratamiento y Análisis de la Información de Seguridad Ciudadana.

brindar servicios eficientemente, a partir de grandes volúmenes de datos, y que necesitan interactuar con otros sistemas.

Ante esta situación surge el siguiente **problema científico**: ¿Cuáles serían las especificaciones arquitectónicas del SIGEPOL, de manera tal que proporcione mecanismos necesarios para su desarrollo?

En el desarrollo de la investigación que aborda este documento se tomó como **objeto de estudio** los modelos y patrones arquitectónicos, y como **campo de acción** los aplicables en la construcción de una arquitectura web para un sistema de gestión.

Este trabajo tiene como **objetivo general** definir e implementar una arquitectura base que permita desarrollar el Sistema de Gestión Policial.

Para su cumplimiento se definen los siguientes **objetivos específicos**:

- ✓ Identificar y seleccionar las tecnologías y herramientas para el desarrollo del sistema.
- ✓ Seleccionar patrones arquitectónicos a utilizar en el sistema.
- ✓ Definir la estructura lógica y física de SIGEPOL.
- ✓ Definir e implementar los mecanismos de seguridad y auditoria.
- ✓ Implementar las clases arquitectónicamente significativas.
- ✓ Definir los mecanismos de comunicación entre los componentes del sistema.
- ✓ Definir mecanismos de comunicación con otros sistemas.
- ✓ Definir convenciones y estándares de codificación.
- ✓ Definir un flujo de trabajo para el desarrollo del sistema.

Las **tareas de investigación** propuestas para dar cumplimiento a los objetivos trazados son:

- ✓ Estudiar e identificar patrones de arquitectura aplicables en el sistema.
- ✓ Investigar los diferentes frameworks de desarrollo web.
- ✓ Investigar los mecanismos y herramientas para la seguridad de aplicaciones web.
- ✓ Investigar las principales herramientas de desarrollo web.
- ✓ Identificar patrones de diseño aplicables en el sistema.
- ✓ Seleccionar la infraestructura base de desarrollo.

Capítulo 1

Arquitectura de Software

Los orígenes de la arquitectura de software se remontan a la década de los 60 cuando en 1968 Edsger Dijkstra de la Universidad Tecnológica en Holanda propuso que se hiciera una estructuración correcta de los sistemas de software antes de lanzarse a programar. Pero no es hasta la década de los 90 que los aspectos arquitectónicos del desarrollo de software comienzan a recibir mayor interés, precisamente por la complejidad creciente de los sistemas informáticos y la imperiosa necesidad de reutilizar y analizar las arquitecturas ya existentes. (2) (3) (4)

En la conferencia de la NATO[4] Science Committee de 1969, un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp formuló estas sorprendentes apreciaciones comentando las ideas de Dijkstra:

"Pienso que tenemos algo, aparte de la ingeniería de software: algo de lo que hemos hablado muy poco pero que deberíamos poner sobre el tapete y concentrar la atención en ello. Es la cuestión de la arquitectura de software. La arquitectura es diferente de la ingeniería. Como ejemplo de lo que quiero decir, echemos una mirada a OS/360[5]. Partes de OS/360 están extremadamente bien codificadas. Partes de OS, si vamos al detalle, han utilizado técnicas que hemos acordado constituyen buena práctica de programación. La razón de que OS sea un amontonamiento amorfo de programas es que no tuvo arquitecto. Su diseño fue delegado a series de grupos de ingenieros, cada uno de los cuales inventó su propia arquitectura. Y cuando esos pedazos se clavaron todos juntos no produjeron una tersa y bella pieza de software." (5)

Sharp continúa su alegación afirmando que con el tiempo probablemente llegue a hablarse de *"la escuela de arquitectura de software de Dijkstra"* y se lamenta que en la industria de su tiempo se preste tan poca o ninguna atención a la arquitectura. La frase siguiente también es extremadamente visionaria:

"Lo que sucede es que las especificaciones de software se consideran especificaciones funcionales. Sólo hablamos sobre lo que queremos que haga el programa. Es mi creencia que cualquiera que sea responsable de la implementación de una pieza de software debe

[4] Organización del Tratado del Atlántico Norte (OTAN), NATO por sus siglas en inglés.
[5] Sistema operativo producido por IBM entre 1965 y 1972.

especificar más que esto. Debe especificar el diseño, la forma; y dentro de ese marco de referencia, los programadores e ingenieros deben crear algo. Ningún ingeniero o programador, ninguna herramienta de programación, nos ayudará, o ayudará al negocio del software, a maquillar un diseño feo. El control, la administración, la educación y todas las cosas buenas de las que hablamos son importantes; pero la gente que implementa debe entender lo que el arquitecto tiene en mente." (5)

Existen diversas definiciones de arquitectura de software. Perry y Wolf en 1992 la definieron como un conjunto de elementos que tienen una forma común. Garlan and Shaw en 1994 y 1996 la conceptualizaron como una colección de componentes y conectores unidos con una descripción de la interacción entre esos componentes y conectores. Bass, Clements, y Kazman en 1998 la definen como la estructura de las estructuras de un sistema, la cual comprende componentes de software, las propiedades visibles externas de estos componentes, y las relaciones entre ellos. La definición que brinda el documento de IEEE Std 1471-2000 que se formula así: "La Arquitectura de Software es 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". (3) (6) (7) (8) (9)

La arquitectura es un medio para alcanzar un fin, es una técnica de trabajo, donde influyen los conocimientos básicos y la experiencia del arquitecto, o del equipo encargado de su implementación, así como el ambiente técnico. Es decir, esta depende del contexto donde es aplicada. La misma es impactada por las metas funcionales y de calidad del cliente, y de la organización que la desarrolla; pero a su vez influye sobre estos elementos y sobre el sistema que está siendo desarrollado, formando de esta manera un sistema que se retroalimenta. (10)

1.1 Cimientos

La arquitectura es el resultado de un conjunto de decisiones referentes al negocio y a la tecnología, es decir, surge de un análisis donde se tienen en cuenta restricciones del negocio y restricciones desde el punto de vista tecnológico. (10)

La arquitectura depende del dominio de la aplicación, por lo que el arquitecto no tiene libertad absoluta para realizar su trabajo, si no que debe atarse al dominio en el cual está trabajando en las condiciones que le impone el ambiente externo. En muchas ocasiones la arquitectura está ceñida a una serie de requisitos o restricciones, como pueden ser herramientas y tecnologías de desarrollo, impuestas por el ambiente externo. (10)

1.2 Los stakeholders

Un stakeholder, desde el punto de vista informático, es toda aquella persona que tiene algún interés en el sistema a desarrollar, o forme parte de alguna manera del mismo; pueden ser inversores, programadores, gerentes del proyecto, clientes, usuarios, en fin, cualquiera que tenga una determinada participación en el sistema. (3) (10) (11)

Cada stakeholder tiene objetivos y preocupaciones propias. Un paso importante es lograr que todos los stakeholders tengan la misma visión sobre el proyecto a desarrollar. Cada parte interesada emitirá su criterio y pondrá restricciones o requisitos, que pueden ser funcionales, como las características, capacidades o casos de uso que debe tener el sistema; o no funcionales, como el rendimiento, la seguridad o la disponibilidad de la aplicación. El arquitecto debe lidiar con los conflictos que puedan surgir de opiniones divergentes, así como solucionar deficiencias técnicas que puedan surgir de los puntos de vistas de los stakeholders, dándole prioridades a los objetivos o restricciones impuestos por los mismos; actuando de integrador y director del proyecto en sus aspectos tecnológicos, siempre teniendo en cuenta el contexto en que se está desarrollando. (3) (6) (10)

1.3 Impacto de la organización

La arquitectura es influenciada por la estructura o naturaleza de una organización. Por lo que hay que tener en cuenta las características de la empresa para la cual se desarrollará el software. Por otro lado la arquitectura también depende de la estructura o naturaleza de la empresa de software, estando condicionada por los conocimientos o experiencia previa que tengan los desarrolladores; por ejemplo, si la empresa ya ha realizado proyectos utilizando una solución “cliente - servidor”, es muy probable que la solución propuesta sea también de este tipo, por que será el tipo de solución en la que los desarrolladores se sentirán más cómodos. (10)

Otro elemento importarte que influye directamente en arquitectura son las habilidades del equipo de desarrollo, es decir, el conjunto de conocimientos previos, el nivel y el tipo de conocimiento que poseen los integrantes del equipo que realizará el proyecto. El arquitecto, o el equipo de desarrollo de la arquitectura, seleccionarán una determinada solución o patrón de solución en base a los conocimientos previos que posean. (10)

El cronograma del proyecto también juega su papel determinante en el desarrollo de la arquitectura, teniendo en cuenta que no es lo mismo desarrollar en un cronograma

apretado que sobre un cronograma libre. La arquitectura, y principalmente sus aspectos tecnológicos pueden estar condicionados por el presupuesto asignado para el desarrollo del proyecto. (10)

1.4 Impacto de la arquitectura

La arquitectura determina una estructura para el desarrollo del sistema, estableciendo equipos, con diversas actividades de desarrollo, para cada unidad del sistema. Adicionalmente se determina un cronograma para cada unidad del sistema, que la organización debe asumir y adaptarse, de ser necesario, para su cumplimiento. La estructura organizacional de un proyecto quedará como ejemplo de trabajo en equipo, e impactará en futuros proyectos a desarrollar por la empresa. Del éxito que tenga la arquitectura propuesta dependerá en gran medida el éxito del sistema, influyendo de esta manera en el impacto de la organización en el mercado. Una vez utilizada una solución, quedará como experiencia dentro de la empresa, permitiendo reutilizar las estructuras creadas y disminuir tiempos y costos de desarrollo. (10)

1.5 Características fundamentales

Algunas de las principales virtudes consensuadas de la arquitectura de software son:

Comunicación mutua: La arquitectura de software representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales. (3) (4) (10)

Decisiones tempranas de diseño: La arquitectura de software representa la encarnación de las decisiones de diseño más tempranas sobre un sistema, y esos vínculos tempranos tienen un peso fuera de toda proporción en su gravedad individual con respecto al desarrollo restante del sistema, su servicio en el despliegue y su mantenimiento. (3) (4)

Restricciones constructivas: Una descripción arquitectónica proporciona vistas parciales para el desarrollo, indicando los componentes y las dependencias entre ellos. Por ejemplo, una vista en capas de una arquitectura documenta típicamente los límites de abstracción entre las partes, identificando las principales interfaces y estableciendo las formas en que unas partes pueden interactuar con otras. (4) (12)

Reutilización: La arquitectura de software representa un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de frameworks[6] en el que se pueden integrar componentes. (3) (4) (12)

Evolución: La arquitectura de software puede exponer las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Haciendo explícitas estas "paredes" perdurables, quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad y reutilización. (4) (12)

Análisis: Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios. (4) (12)

Administración: La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales. (4) (10) (12)

1.6 Niveles de abstracción arquitectónicos

El estudio de los aspectos arquitectónicos del desarrollo de software está estructurado en niveles jerárquicos, partiendo del más general al más particular. Cada nivel tiene una profundidad bien definida lo que permite un análisis más detallado del mismo. De este modo podemos distinguir varios niveles. (13)

1.6.1 Estilos arquitectónicos

Desde sus inicios, en la arquitectura de software, se observó que en la práctica del diseño y la implementación ciertas regularidades que aparecían una y otra vez como respuesta a similares demandas. El número de esas formas no parecía ser muy grande. Muy pronto se les llamó estilos, por analogía con el uso del término en arquitectura de edificios. (14)

[6] Conjunto de librerías, componentes y herramientas de apoyo que son utilizados en la organización y desarrollo de proyectos.

Un estilo arquitectónico define una familia de sistemas que siguen un mismo patrón estructural, estableciendo un vocabulario de tipos de componentes y conectores y un conjunto de restricciones sobre cómo combinar esos componentes y conectores. (9)

Los componentes y conectores que forman el estilo arquitectónico representan unidades computacionales y de datos y los mecanismos de interacción entre ellos, las invariantes del estilo se representan a través de restricciones de interconexión. Asociados a cada estilo hay una serie de propiedades que lo caracterizan, determinando sus ventajas e inconvenientes, condicionando la elección de uno u otro estilo. (15)

Los principales estilos arquitectónicos son:

- ✓ Sistemas de flujo de datos
  - Secuencial en lote
  - Tubos y filtros
- ✓ Sistemas centrados en los datos
  - Arquitecturas basadas en pizarras o repositorios
- ✓ Sistemas de llamada y retorno
  - Modelo – Vista – Controlador (MVC[7])
  - Arquitecturas en Capas
  - Arquitecturas Orientadas a Objetos
  - Arquitecturas Basadas en Componentes
- ✓ Peer to Peer
  - Procesos comunicativos
  - Arquitecturas Basadas en Eventos
  - Arquitecturas Orientadas a Servicios
  - Arquitecturas Basadas en Recursos
- ✓ Máquinas virtuales
  - Intérpretes
  - Sistemas basados en reglas

1.6.2 Modelos y arquitecturas de referencia

Constituyen la particularización de un estilo determinado, estandarizando el conjunto de componentes y estableciendo un método definido de desarrollo. (15)

Por ejemplo, el modelo OSI[8], modelo de referencia para la definición de arquitecturas de interconexión de sistemas de comunicaciones, particulariza el estilo de organización en capas, con 7 niveles.

[7] Siglas del inglés Model - View - Controller.
[8] Siglas del inglés Open System Interconnection

1.6.3 Marcos de trabajo o frameworks

Definen una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, definiendo de forma abstracta una serie de componentes y sus interfaces y estableciendo las reglas y mecanismos de interacción entre ellos. Suele incluirse la implementación de algunos de los componentes e incluso varias implementaciones alternativas.

El usuario selecciona, instancia, extiende y reutiliza los componentes del marco, completando la arquitectura con nuevos componentes específicos dentro de las relaciones estructurales del marco.

Básicamente se presentan como un diseño reutilizable de todo o parte de un sistema, representado por un conjunto de componentes abstractos y la forma en que los componentes interactúan. Una alternativa es verlos como un esqueleto de una aplicación que debe ser adaptado por el desarrollador según sus necesidades concretas. Un marco de trabajo define el patrón arquitectónico que relaciona los componentes de un sistema.

Algunos frameworks de propósitos generales son:

- ✓ Spring: es un framework de código abierto para el desarrollo de aplicaciones sobre la plataforma Java. También existe una versión para la plataforma .NET denominada Spring.net.
- ✓ Symfony: es un framework diseñado para el desarrollo de aplicaciones web escritas en PHP, siguiendo el patrón de diseño arquitectónico Modelo – Vista – Controlador (MVC).
- ✓ Ruby on Rails: es un framework de aplicaciones web de código abierto, escrito en el lenguaje de programación Ruby, siguiendo el paradigma de arquitectura MVC.

1.6.4 Familias y líneas de productos

Las familias y líneas de productos se refieren a técnicas de ingeniería para crear un conjunto de sistemas de software similares, a partir de un conjunto compartido de componentes de software, usando un medio común de producción. (16)

La creación de una línea de productos de software posibilita la reducción del tiempo promedio de creación y entrega de nuevos productos, la disminución del número promedio de defectos por producto, del esfuerzo promedio requerido para desarrollar y mantener los productos y del costo promedio de producción de los productos. Además,

incrementa el número total de productos que pueden ser efectivamente desplegados y mantenidos. (17)

Por otro lado también se reportan beneficios estratégicos de negocios como la reducción en el tiempo de entrega (*time-to-market*) y el tiempo de retorno (*time-to-revenue*) de nuevos productos, mejoras en el valor competitivo del producto, márgenes mayores de ganancias, mejor calidad de los productos, mejoras en la reputación de la empresa, mayor escalabilidad del modelo de negocios en términos de productos y mercados, mayor agilidad para expandir el negocio a nuevos mercados y la reducción de riesgos en la entrega de productos. (17)

1.6.5 Instancias

Se le llama instancia arquitectónica a la arquitectura de una aplicación concreta. Esta instancia es construida tomando como base una arquitectura de referencia, que es creada a partir de un sistema de patrones arquitectónicos y de un modelo de referencia proyectado en elementos de software y los flujos de datos que existen entre ellos. Las características no funcionales y los requerimientos funcionales son elementos básicos para el diseño de la arquitectura de software.

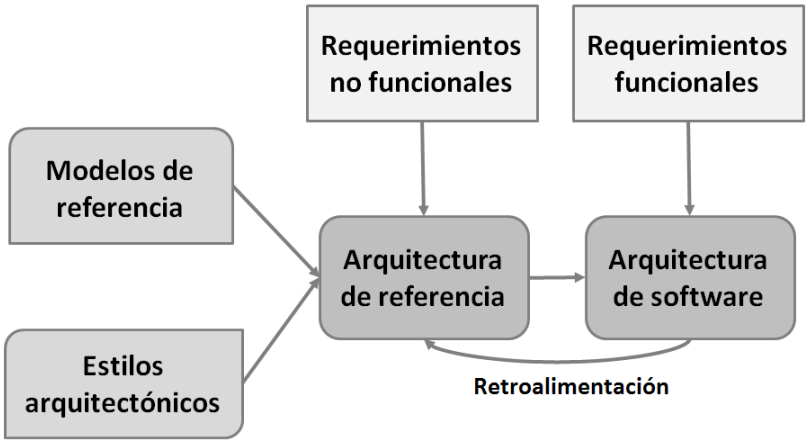


Figura 1 Elementos para crear una arquitectura de software

1.7 Arquitectura Cliente – Servidor

Esta arquitectura se encuentra dentro de la clasificación de estilo de llamada y retorno. El cliente y el servidor generalmente están localizados en diferentes sistemas, sin embargo pueden encontrarse en el mismo sistema. 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 al cliente. (14)

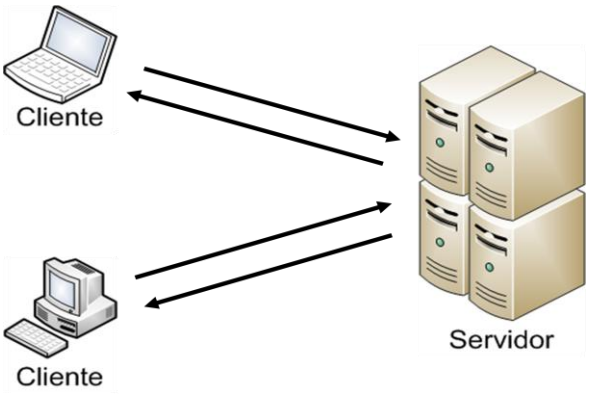


Figura 2 Arquitectura Cliente - Servidor

Los clientes pueden ser clasificados como clientes ligeros o pesados. Lo clientes pesados típicamente contienen, además de la lógica de presentación, gran parte de la lógica de negocio de la aplicación. Lo clientes ligeros manejan usualmente sólo la lógica de presentación, permitiendo que futuros cambios de negocio en la aplicación no afecten al cliente. (14)

1.8 Arquitectura en capas

La arquitectura en capas es una organización jerárquica, donde cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. (9)

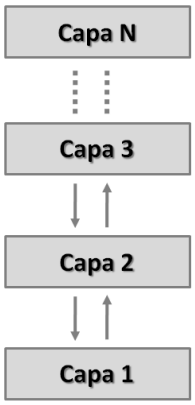


Figura 3 Flujo de información entre las capas

El estilo soporta un diseño basado en niveles de abstracción crecientes lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. Permite realizar optimizaciones y refinamientos enfocando los cambios en un solo lugar. Proporciona amplia reutilización dada la división bien definida de responsabilidades. Al igual que los tipos de datos abstractos, se pueden utilizar

diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. (14)

Una especialización muy usada de la arquitectura en capas es la arquitectura de tres capas donde se observan muy bien delimitadas las responsabilidades de cada funcionalidad en la aplicación: presentación, negocio y datos.

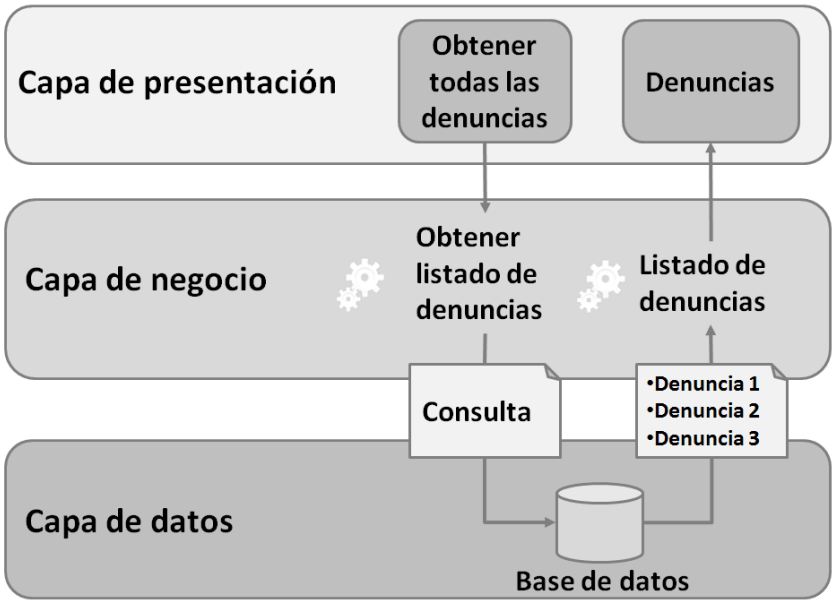


Figura 4 Arquitectura en tres capas

Capa de presentación: es la que ve el usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso. Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario.

Capa de negocio: es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso de negocio. Se denomina capa de 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 para almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Capa de 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.

Todas estas capas pueden residir en un único ordenador, si bien lo más usual es que haya una multitud de ordenadores en donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o mas ordenadores. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

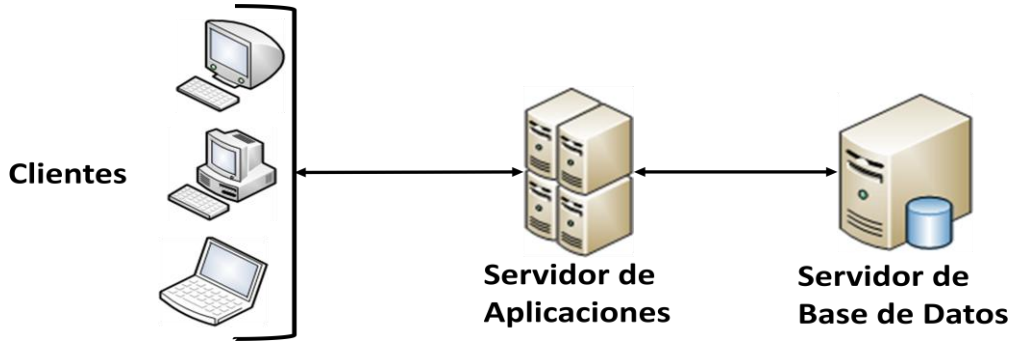


Figura 5 Distribución física de una arquitectura en capas

1.9 Modelo – Vista – Controlador

Modelo – Vista – Controlador (MVC) es un patrón 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 patrón fue descrito por primera vez en 1979 por Trygve Reenskaug, realizando la primera implementación para Smalltalk-80[9]. (18)

[9] Lenguaje de programación orientado a objeto con tipos dinámicos.

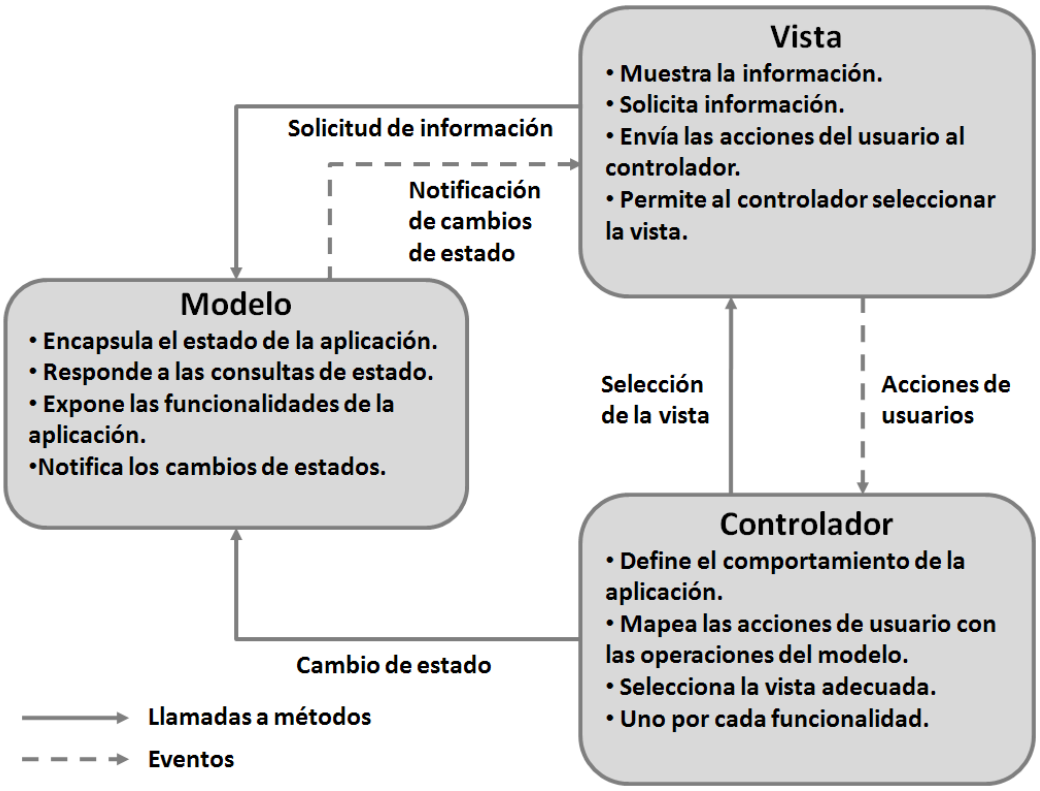


Figura 6 Modelo - Vista - Controlador

El modelo representa los datos y las reglas de negocio que rigen su acceso y actualización; puede verse como una modelación de los procesos del mundo real. (18)

Las vistas se encargan de presentar los datos obtenidos del modelo. Es responsabilidad de las vistas mantener la información actualizada, esto se pude logar a través de peticiones de actualización al modelo o a través de notificaciones de cambio que el modelo emite (eventos). (18)

El controlador actúa como un traductor de las acciones que se realizan en las vistas en las operaciones que ocurren en el modelo. Las acciones realizadas por el modelo desencadenan la activación de procesos de negocio o cambian el estado del modelo. Sobre la base de las acciones del usuario y los resultados del modelo, el controlador responde mediante la selección de la vista apropiada. (18)

1.10 Patrones de diseño

Christopher Alexander en 1979 dijo *"cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez."* (19)

No obstante, no fue hasta principios de los 90's cuando los patrones de diseño tuvieron un gran éxito en el mundo de la informática a partir de la publicación del libro Design Patterns escrito por el GoF[10], compuesto por Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides, en el que se recogían 23 patrones diseño comunes.

Un patrón de diseño aborda problemas recurrente del diseño orientado a objetos. En él se describe el problema, la solución, el momento de aplicar la solución, y sus consecuencias. También da consejos de aplicación y ejemplos. La solución es un conjunto de clases y objetos y las relaciones que se establecen entre ellos para resolver el problema. La solución es personalizada y aplicada para resolver el problema en un contexto particular. (19)

Aunque la orientación a objetos facilita la reutilización de código, la reutilización efectiva sólo se produce a partir de un buen diseño basado en el uso de soluciones que han probado su utilidad en situaciones similares; por lo que patrones constituyen un conjunto plantillas básicas que cada diseñador adapta a las peculiaridades de su aplicación, logrando soluciones elegantes y efectivas.

[10] Banda de los cuatro, GoF por sus siglas en inglés (Gang of Four)

Capítulo 2

Tecnologías y herramientas de desarrollo

En este capítulo se analizan las características fundamentales de las tecnologías y herramientas de desarrollo que se utilizarán en el desarrollo del sistema.

2.1 J2EE

Java 2 Platform, es una plataforma creada por la Sun Microsystems[11], en su edición empresarial (J2EE[12]) define un estándar para el desarrollo de aplicaciones empresariales multicapas. La plataforma J2EE simplifica el desarrollo de estas aplicaciones sobre una base estandarizada, con componentes modulares acompañado de una amplia gama de servicios y manejando muchos detalles del funcionamiento de las aplicaciones, sin necesidad de una programación compleja.

J2EE mantiene muchas de las características de la edición estándar de Java 2 Platform (J2SE[13]), como su portabilidad *"write once, run anywhere"*[14], el API[15] de JDBC[16] para el acceso a bases de datos, y un modelo de seguridad que protege los datos incluso en aplicaciones de Internet. Sobre esta base J2EE adiciona soporte completo para los componentes Enterprise JavaBeans (EJB[17]), el API para Java Servlets[18], Java Server

[11] Es una empresa informática de Silicon Valley, fabricante de semiconductores y software. Las siglas SUN se derivan de «Stanford University Network», proyecto que se había creado para interconectar en red las bibliotecas de la Universidad de Stanford.
[12] Siglas del inglés Java 2 Enterprise Edition
[13] Siglas del inglés Java 2 Standard Edition
[14] "escribir una vez, ejecutar en cualquier parte"
[15] Application Programming Interface, es el conjunto de objetos y funciones que ofrece cierta biblioteca para ser utilizado por otro software.
[16] Java Database Connectivity, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.
[17] Enterprise JavaBeans, proporcionan un modelo de componentes distribuido estándar del lado del servidor.
[18] Un servlet es un objeto que se ejecuta en un servidor o contenedor JEE (ej. Tomcat), especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente HTML

Pages (JSP[19]) y tecnología XML. Además J2EE asegura la interoperabilidad de servicios web proveyendo soporte para *WS-I Basic Profile*[20].

Una de las principales razones por la que se seleccionó esta plataforma es porque se cuenta con limitaciones legales por parte del cliente: el decreto Ley No. 3.390 publicado en La Gaceta Oficial de la República Bolivariana de Venezuela establece que la Administración Pública Nacional empleará prioritariamente Software Libre Desarrollado con Estándares Abiertos, en sus Sistemas, Proyectos y Servicios Informáticos, permitiendo el uso de otro tipo de software solo en casos de no ser posible la adquisición de Software Libre Bajo Estándares Abiertos, en tal caso, los órganos y entes de la Administración Pública Nacional deberán solicitar ante el Ministerio de Ciencia y Tecnología, la autorización para adoptar otro tipo de soluciones bajo normas y criterios establecidos por ese Ministerio. (20)

2.2 Spring Framework

Spring Framework (también conocido simplemente como Spring) es un framework de código abierto para el desarrollo de aplicaciones en la plataforma Java. La gran popularidad que ha adquirido Spring en la comunidad de desarrolladores es un indicador de su valía, existen también razones técnicas que lo avalan:

- ✓ **En él se abordan aspectos importantes que muchos otros frameworks populares no lo hacen**. Spring se centra en torno a una forma de gestionar su negocio de objetos. (21) (22)
- ✓ **Es amplio y modular**. Spring tiene una arquitectura en capas, lo que significa que se puede elegir para utilizar casi cualquiera de sus partes en forma aislada, su arquitectura es internamente coherente. Se puede optar por utilizar Spring sólo para simplificar el uso de JDBC, por ejemplo, u optar por la aplicación de Spring para gestionar todos sus objetos de negocio. Es fácil de introducir gradualmente en proyectos ya existentes. (21) (22)
- ✓ **Está diseñado desde el principio para ayudarte a escribir código fácil de testear.** (21) (22)
- ✓ **Hace un aporte significativo para la integración de tecnologías**. Su rol en este sentido es reconocido por varios proveedores. (21) (22)

[19] Java Server Pages (JSP) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.
[20] Especificación del consorcio Web Services Interoperability (WS-I) que proporciona orientación para la interoperabilidad entre las tecnologías fundamentales de servicios web, tales como SOAP, WSDL y UDDI.

Spring no es solo un framework más del cual depende un proyecto, es, quizás, la solución para el proyecto. En él se abordan las principales temáticas para el desarrollo de aplicaciones empresariales.

Beneficios arquitectónicos de Spring

- ✓ Es efectivo administrando el conjunto de objetos que maneja su aplicación, quedando libre la elección o no del uso de EJB. Este mecanismo puede ser utilizado en cualquier capa arquitectónica.
- ✓ Estandariza el uso de ficheros XML para la configuración de su aplicación. Esto es logrado mediante el uso de la **inversión del control.**
- ✓ Promueve el uso de buenas prácticas de programación. Reduce casi a cero el costo de utilizar interfaces en lugar de clases.
- ✓ Es diseñado para que las aplicaciones dependan en la menor medida posible de sus librerías. Los objetos de negocio pueden ser construidos sin crear dependencias con Spring.
- ✓ Se puede diseñar un modelo de objetos haciendo uso de EJB o no, así como de otras tecnologías como POJO[21], sin afectar su funcionamiento.
- ✓ Ayuda a resolver muchos problemas sin utilizar EJB. Spring proporciona una alternativa a los EJB que es apropiado para muchas aplicaciones.
- ✓ Proporciona mecanismos consistentes para el acceso a los datos, ya sea utilizando JDBC o un framework de mapeo objeto – relación (ORM[22]), tales como Hibernate, iBATIS o JDO.

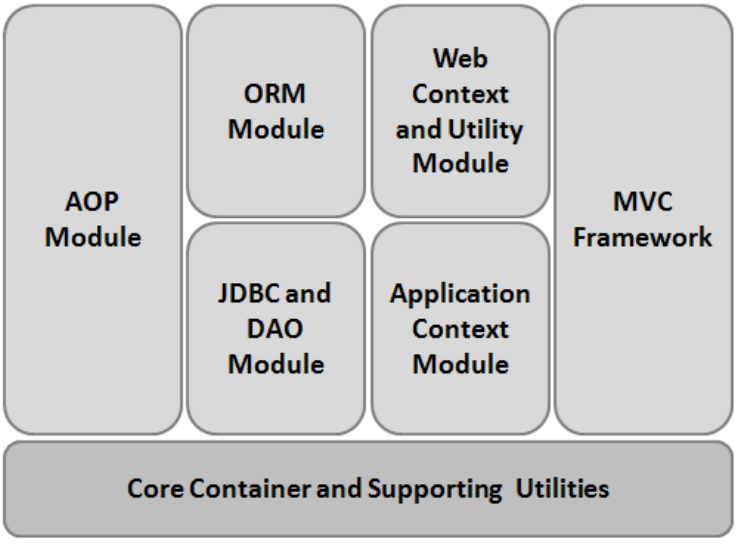


Figura 7 Spring está compuesto por módulos bien definidos.

[21] Acrónimo de Plain Old Java Object es una sigla utilizada por programadores de Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.
[22] Object - Relational mapping, es una técnica de programación para convertir una base de datos relacional en un modelo de datos en un lenguaje de programación orientado a objetos.

Spring está compuesto de siete bien definidos módulos. Cuando se toman en su conjunto, estos módulos proveen todo lo necesario para desarrollar una aplicación empresarial. Sin embargo, es posible elegir los módulos que se adapten a la aplicación a desarrollar e ignorar el resto. (22)

2.2.1 Inversión del control (IoC[23])

La inversión del control es el corazón de Spring. El concepto detrás de IoC es el principio de Hollywood: *"Don't call me, I'll call you"*[24]. Lo cual mueve la responsabilidad de hacer que las cosas sucedan al framework, y libera de ello al código de la aplicación. (22)

¿Qué es lo que se invierte? El aspecto que se invierte es el modo que los objetos obtienen las instancias de los objetos que colaboran con él o de los cuales depende. Bajo esta premisa Martin Fowler denominó como **inyección de dependencias** a esta forma de invertir el control. (23)

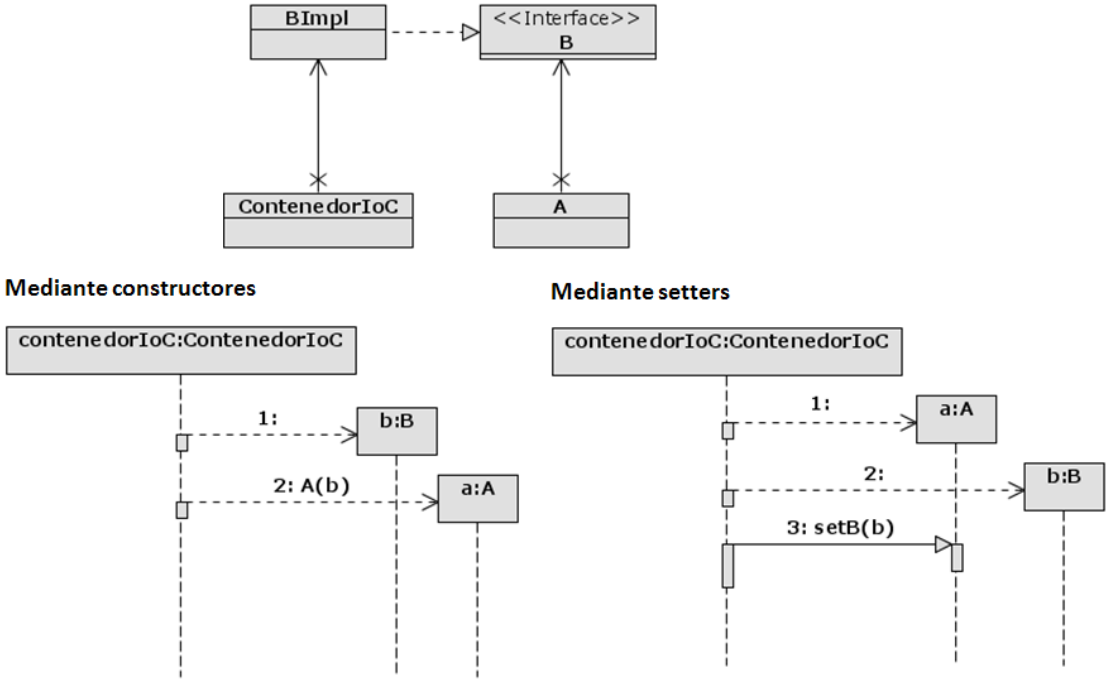


Figura 8 Inyección de dependencias.

Cualquier aplicación, no trivial, se compone de dos o más clases que colaboran entre sí para realizar alguna lógica de negocio. Tradicionalmente, cada objeto es responsable de la obtención de sus propias referencias a los objetos con los que colabora (y sus

[23] Siglas del inglés Inversion of Control.
[24] "No me llames, yo te llamaré"

dependencias). Esto dar lugar a un fuerte acoplamiento entre una clase y sus dependencias. (22)

Con el uso de IoC, los objetos seden la responsabilidad de la creación de sus dependencias a una tercera entidad que administra los objetos del sistema, la cual "inyecta" las dependencias en los objetos correspondientes.

2.2.2 Spring MVC

Es un módulo de Spring que implementa el patrón ***modelo – vista – controlador***[25] para aplicaciones web. Desde el momento en que una petición es recibida hasta el instante en que la respuesta es enviada al cliente muchos componentes de Spring MVC entran en acción. La Figura 9 muestra el ciclo de vida por el que atraviesa una petición.

El proceso comienza cuando un cliente (generalmente un navegador web) envía una petición①. El primer componente es el *Dispatcher Servlet* (despachador). Como la mayoría de los frameworks MVC en java, Spring MVC dispone de un único *controlador frontal*. Un controlador frontal es un patrón de diseño web donde se tiene un único servlet que delega la responsabilidad de procesar las peticiones en otros componentes. En el caso de Spring MVC, el Dispatcher Servlet es el controlador frontal.

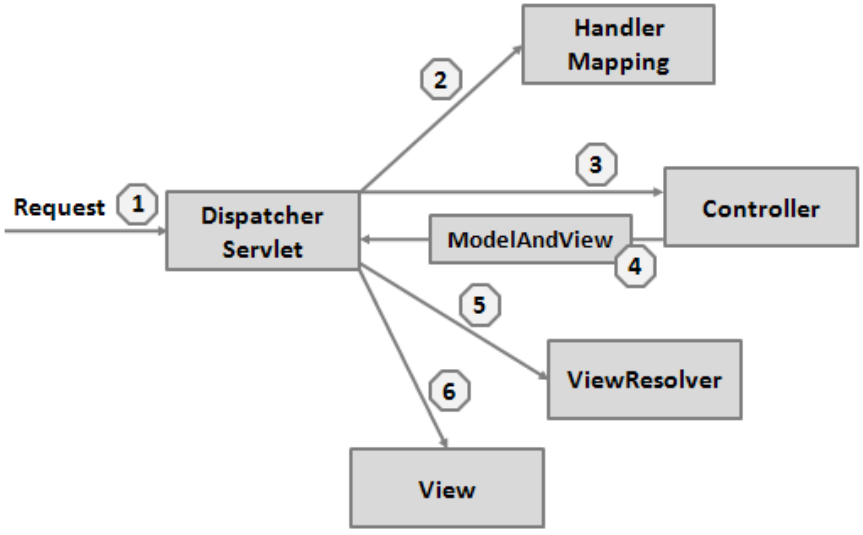


Figura 9 Ciclo de vida de una petición en Spring MVC

El componente encargado de procesar una petición es el controlador. Para conocer el controlador al que corresponde procesar una petición el despachador frontal se auxilia del

[25] Ver el epígrafe: 1.9 Modelo – Vista – Controlador.

Handler Mapping (uno o varios)②. Los Handler Mapping realizan su trabajo mediante un mapeado de peticiones con sus respectivos objetos controladores.

El despachador envía al controlador indicado los datos de la petición para que sean procesados, ejecutándose la lógica de negocio asociada a esa solicitud③. En un buen diseño, el controlador no realiza ninguna lógica de negocio asociada con la petición, o solo muy poca, si no que delega esa funcionalidad en objetos de servicio diseñados para ello.

Como resultado de este proceso el controlador devuelve un objeto ModelAndView④, que contiene una vista, o el nombre lógico de una vista, y los datos que se mostraran en dicha vista.

El despachador mediante el ViewResolver⑤ obtiene la vista (por ejemplo una página JSP) que mostrará los datos. Finalmente envía los datos a la vista⑥, la cual será la encargada de construir la respuesta a la solicitud enviada.

2.2.3 Spring AOP

El sueño de muchos desarrolladores de software se manifiesta en tener un mundo en el que desarrollar aplicaciones se sustente en ensamblar componentes de software que interactúen a través de simples llamadas a métodos. Sin embargo, por lo general esto pasa por alto que ciertos aspectos de las aplicaciones suelen afectar a todas las partes de un programa. (24)

A la larga los programas tienden a contaminarse con segmentos de código que intentan lidiar con aquellos aspectos de las aplicaciones que no son el foco central del problema que éstos intentan solucionar. Con frecuencia estos suelen entrometerse en el código original afectando con ello la aspiración de obtener soluciones reutilizables. Esta es el área que aborda lo que se ha dado en llamar *Programación Orientada a Aspectos* (AOP)[26] al intentar proveer estrategias de reutilización para solucionar los problemas mencionados. (24) (22)

Ejemplos tradicionales de estos “aspectos” que pueden estar presentes o no en muchos componentes de software, con independencia del dominio del problema al que están dedicados, son la seguridad, la serialización, la sincronización, etc.

[26] Siglas del inglés Aspect Oriented Programming.

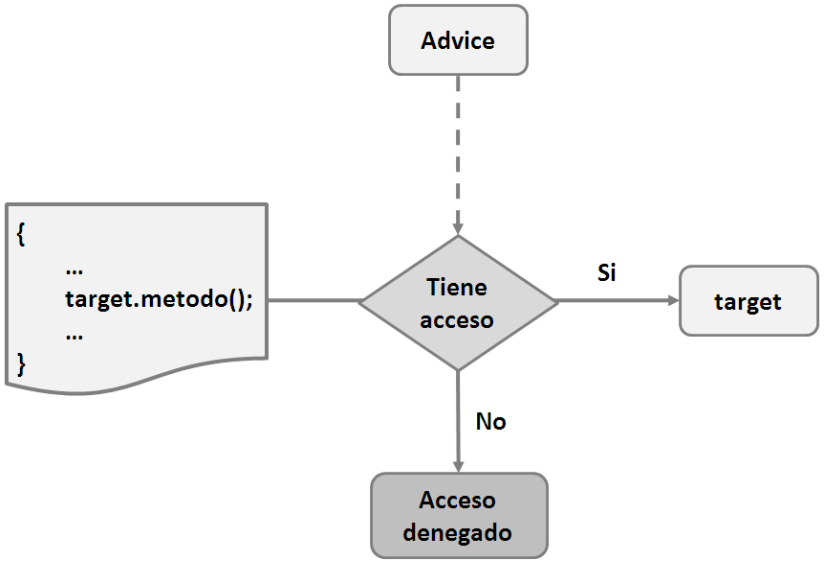


Figura 10 Intercepción de la llamada a un método mediante la utilización de aspectos.

Spring AOP es un módulo que provee mecanismos para la instrumentación de los conceptos referentes a la AOP. Estos conceptos son los siguientes: (25)

Aspect: es la funcionalidad que se quiere separar del resto del sistema y que va a actuar a lo largo del mismo, como por ejemplo la seguridad.

Joint point: es un punto durante la ejecución de un programa, tales como la ejecución de un método o el tratamiento de una excepción.

Advice: es la acción concreta que realiza el aspecto en un punto de unión. Esta acción puede ejecutarse en diferentes instantes relativos al punto de unión, como pueden ser antes de la ejecución o después de la ejecución de un método. Por ejemplo, interceptar la llamada a un método y verificar que el usuario autenticado en el sistema tenga los privilegios necesarios para ejecutarlo.

Target: es el objeto al que será aplicado el aspecto. Por ejemplo, el objeto propietario del método interceptado.

2.3 Acegi Security System

Acegi Security System es un framework que proporciona mecanismos de seguridad para aplicaciones basadas en Spring. Provee un conjunto de clases que se configuran en el contexto de la aplicación, explotando al máximo las características de programación orienta a aspectos y de inyección de dependencias que contiene Spring.

Acegi soporta la mayoría de los procesos de autenticación existentes además de permitir adicionar nuevos. Posee mecanismos de autorización que involucran el filtrado de

direcciones URLs, y por intercepción de llamadas a métodos mediante la utilización de programación orientada a aspectos. Toda su configuración es muy flexible permitiendo la reutilización de componentes.

2.4 Servicio Central de Autenticación (CAS)

Single Sign-On (SSO) es un aspecto importante en la seguridad de aplicaciones web. El nombre lo dice todo: entra solo una vez y accede a todo. El Grupo de Tecnología y Planificación de la Universidad de Yale ha desarrollado una excelente solución SSO llamada Central Authentication Service (CAS) que funciona muy bien con Acegi.

CAS es una aplicación web que funciona como servidor para autenticar a varias aplicaciones de un sistema una sola vez, sin tener que autenticarse nuevamente. Brinda soporte para el mecanismo de autenticación de Acegi, aunque permite utilizar cualquier mecanismo de autenticación. Maneja de forma transparente la interacción de las aplicaciones cuando el usuario se ha autenticado, y permite desconectar a un usuario de todas las sesiones en las que pudiera estar trabajando.

2.5 JCaptcha

Captcha es el acrónimo de Completely Automated Public Turing test to tell Computers and Humans Apart (*Prueba de Turing*[27] pública y automática para diferenciar a máquinas y humanos). JCaptcha es un framework, escrito en Java, que provee funcionalidades para generar imágenes, a partir de un conjunto de caracteres generados aleatoriamente, de forma que su contenido no pueda ser identificado por un robot o programa de ordenador, que realizan tareas predeterminadas de las aplicaciones. Es una prueba de Turing pública y automática que se basa en ofrecer un desafío para determinar si el usuario es humano o no.

Figura 11 Este es un típico test para la secuencia "smwm" que dificulta el reconocimiento de la máquina distorsionando las letras y añadiendo un gradiente de fondo.

Posee múltiples formas de generar imágenes con distintos tipos de deformaciones, colores, tamaños, tipos de letras, colores de fondo, imágenes de fondo, etc. La desventaja

[27] Se llama Prueba o Test de Turing al procedimiento desarrollado por Alan Turing para corroborar la existencia de inteligencia en una máquina.

que posee no es una desventaja propia del framework sino del concepto en si, debido a que personas con discapacidades visuales no serán capaces de identificar la imagen tratándoseles como si fuera un programa de ordenador a los que se le impide el acceso.

2.6 JUnit

El desarrollo de pruebas automatizadas al código es una parte fundamental en el proceso de desarrollo. Un componente de software no puede probar su efectividad hasta tanto no sea sometido a un conjunto de pruebas. En 1997, Erich Gamma y Kent Beck crearon una herramienta para el desarrollo de pruebas de unidad para aplicaciones escritas en Java, llamada JUnit. (26)

JUnit es un proyecto de código abierto que se convirtió rápidamente en el estándar para el desarrollo de pruebas de unidad para Java. De hecho, las pruebas de modelo, conocido como xUnit, están en camino de convertirse en un framework estándar para "*cualquier"* lenguaje. Hay xUnit frameworks disponible para ASP, C + +, C #, Eiffel, Delphi, Perl, PHP, Python, REBOL, Smalltalk y Visual Basic entre otros. (26)

El término de pruebas de software no fue inventado por JUnit, ni tampoco el de pruebas de unidad. Originalmente es termino era usado para describir pruebas al comportamiento de una unidad simple de trabajo. Con el tiempo su uso fue extendido, por ejemplo, la definición de la IEEE sobre pruebas de unidad es "Pruebas de unidades individuales de hardware o software o grupos de unidades relacionadas". (26)

JUnit 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.

2.7 EasyMock

EasyMock es un framework para la construcción de objetos falsos (Mock Object) que facilitan el desarrollo de las pruebas de unidad con JUnit. Esto se logra a través del mecanismo de proxy de Java. (27)

El uso de los objetos falsos permite la simulación de los objetos reales, mediante la utilización de datos falsos o generados estáticamente. Esto posibilita que se puedan

realizar pruebas de unidad a componentes aislados, falseando los objetos de los que depende.

2.8 iBATIS

iBATIS Data Mapper Framework hace más fácil el uso de una base de datos en una aplicación Java. iBATIS mapea objetos con procedimientos almacenados o sentencias SQL a través del usos de ficheros XML de configuración. La simplicidad es la mayor ventaja de iBATIS sobre otras herramientas de mapeo objeto-relación.

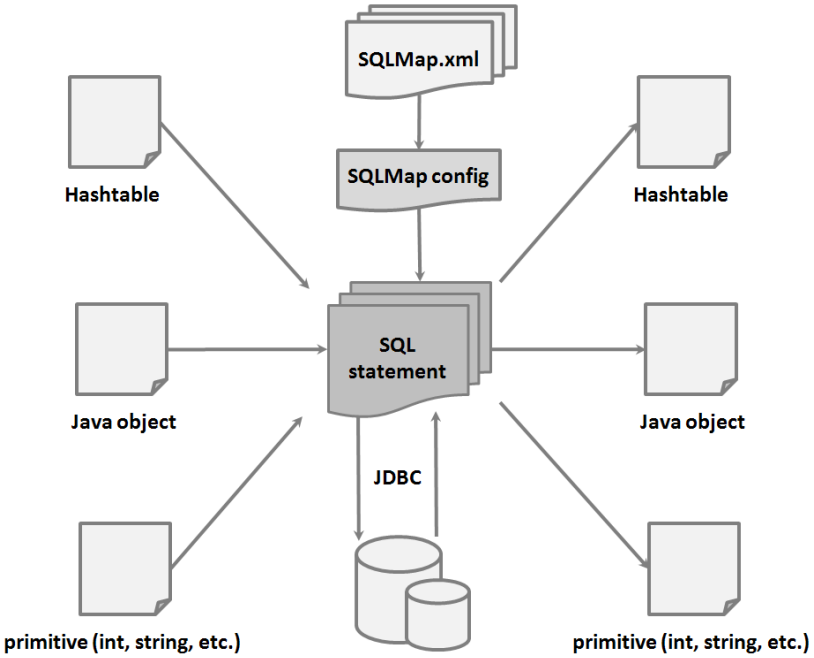


Figura 12 iBATIS Data Mapper Framework

Con el uso de iBATIS Data Mapper se pueden explotar todas las potencialidades inherentes a los procedimientos almacenados y sentencias SQL.

2.9 JasperReports

JasperReports es una librería, de código abierto escrita en Java, para la construcción de reportes. Es integrable a cualquier aplicación en Java, ofreciendo información operacional donde quiera que se necesite: en la pantalla, en la impresora o para otras aplicaciones. Los formatos de salida que incluye son: PDF, HTML, XLS, CSV, RTF (Word), TXT o archivos XML.

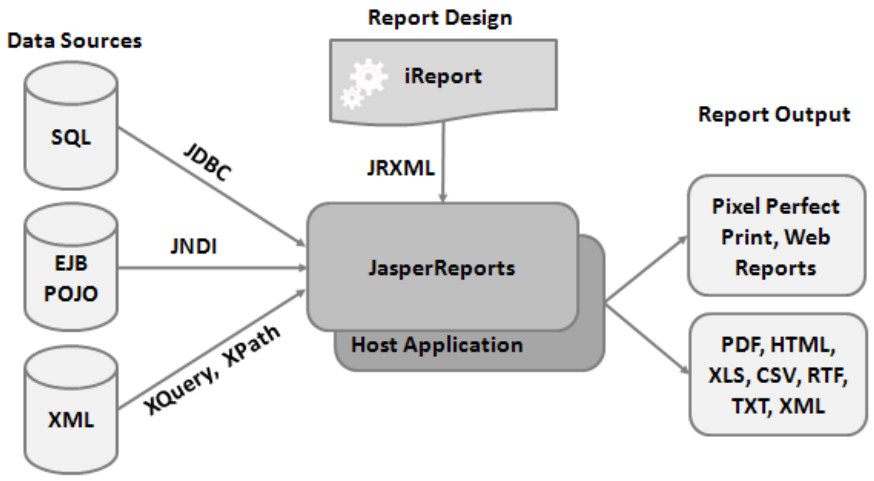


Figura 13 Arquitectura de JasperReports.

Los datos utilizados en la confección de los reportes pueden ser subministrados a partir de parámetros de entrada o a través de una fuente de datos incluida en JasperReports. Incluye soporte para fuentes de datos como JDBC, JavaBeans, POJOs y XML. El uso de JasperReports generalmente está aparejado al de iReport. La herramienta iReport es un constructor / diseñador de informes visual, poderoso, intuitivo y fácil de usar. Este instrumento permite que los usuarios corrijan visualmente informes complejos como cartas, imágenes, sub informes, etc.

2.10 Dojo

Dojo es un conjunto de herramientas para DHTML escrito en JavaScript; tiene como objetivo resolver algunos problemas históricos en el uso de DHTML en desarrollo de aplicaciones web dinámicas, además permite agregar funcionalidades dinámicas a las páginas web o cualquier otro entorno que soporte JavaScript, también provee un conjunto de componentes capaces de hacer una aplicación más funcional y posibilitan la creación de interfaces de usuario agradables, potenciando el uso de AJAX[28] de forma sencilla. Con el uso de esta API es posible escribir código en JavaScript compatible con los navegadores web más populares.

[28] Asynchronous JavaScript And XML (JavaScript asíncrono y XML) es una técnica de desarrollo web para crear aplicaciones interactivas.

2.11 Apache Tomcat

Un servidor web es una aplicación que se encarga de aceptar peticiones HTTP de clientes web (generalmente navegadores web) y servir las respuestas HTTP contenedoras de información en formato HTML, XML, imágenes, etc.

Sin embargo en el mundo de J2EE surge un término muy común, contenedor web, el cual además de interceptar solicitudes enviadas en los protocolos: HTTP, HTTPS, FTP, y otros, es esencialmente un entorno de ejecución Java que proporciona una implementación de las tecnologías Java Servlets y Java Server Pages. Además de ser responsable de inicializar, invocar, y gestionar el ciclo de vida de los servlets y las páginas JSP.

Apache Tomcat es un contenedor web desarrollado por la Apache Software Foundation, en un ambiente participativo y abierto, basado en las especificaciones de Sun Microsystems. La versión 5.x es implementada a partir de las especificaciones Servlet 2.4 y JSP 2.0 y cuenta con un mecanismo de recolección de basura perfeccionado.

2.12 Oracle 10g

Oracle 10g Enterprise Edition es un software de gestión de bases de datos desarrollado por la compañía Oracle. Posee excelentes características de desempeño y escalabilidad para el procesamiento de transacciones y depósitos de datos de gran escala en Windows, Linux y servidores UNIX. (28)

Además, Oracle ofrece protección ante las fallas del servidor, fallas del sitio, errores humanos, y reducción del tiempo de baja programado. Incluye protección de datos con seguridad en el nivel de filas, auditorías detalladas, y encriptación transparente de datos. Soporta *data warehousing* [29]de alto desempeño, procesamiento analítico online, y características de extracción de datos. (28)

Oracle 10g Enterprise Edition soporta todos los estándares de tipos de datos relacionales, así como de almacenamiento nativo en XML, texto, documentos, imágenes, audio, vídeo y datos de ubicación geográfica. El acceso a los datos almacenados es a través de interfaces estándar, tales como SQL, JDBC, SQLJ, ODBC, OLE DB y ODP.NET,

[29] Colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza.

SQL/XML, XQuery, y WebDAV. La lógica de negocio desplegada en la base de datos se puede escribir tanto en Java como en PL/SQL[30]. (28)

2.13 Eclipse SDK

La plataforma Eclipse esta diseñada para la integración de IDEs[31] que pueden ser utilizados para diversos propósitos, como desarrollar programas en Java, C++, PHP, etc. Esta diseñada para integrarse y ejecutar módulos llamados plug-ins[32].

Eclipse SDK[33] es el entorno de desarrollo utilizado para desarrollar plug-ins para la plataforma Eclipse. Incluye un conjunto de plug-ins, llamado Java Development Toolkit (JDT), para el desarrollo de aplicaciones en Java, que provee una buena cantidad de vistas, editores, asistentes, compiladores y herramientas de refactorización que facilitan su uso.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

2.13.1 WTP

La plataforma de herramientas web de Eclipse o Eclipse Web Tool Platform (WTP) provee varias APIs para desarrollo de aplicaciones sobre la Web y J2EE. Éstas incluyen editores gráficos, de código fuente para una variedad de lenguajes, asistentes y aplicaciones incorporadas para simplificar el desarrollo de servicios web, además de herramientas y APIs para soportar el despliegue, ejecución y prueba de aplicaciones.

Soporta integración con servidores Web dentro de Eclipse como ambiente de ejecución de primera clase para aplicaciones web. También incluye la configuración de servidores y su asociación con los proyectos web, permitiendo la depuración sobre el servidor de los recursos y las clases.

[30] Lenguaje de consultas embebido en Oracle y PostgreSQL.
[31] Entorno Integrado de Desarrollo, IDE por sus siglas en inglés (Integrated Development Environment).
[32] Es una aplicación que se relaciona con otra para aportarle una funcionalidad nueva y generalmente muy especifica.
[33] Siglas del inglés Software Development Kit.

2.13.2 Spring IDE

Spring IDE es un plug-in que sirve como interfaz de usuario gráfica para la configuración de los archivos usados por Spring Framework. Permite el completamiento de etiquetas, valores de atributos y elementos en estos archivos de configuración.

Muestra un árbol con todos los proyectos de Spring y sus archivos de configuración, permite hacer búsquedas de beans en dichos archivos, además de mostrar gráficamente los beans y sus dependencias.

Presenta un editor gráfico con completamiento y validaciones para los archivos de definiciones del flujo web usado por el Spring Web Flow.

2.13.3 Subclipse

Subclipse es un plug-in para Eclipse que adiciona integración para el control de versiones (Subversion específicamente), permitiendo operaciones de sincronización, actualización, entre otras. Permite bloqueos a recursos para que otros usuarios no puedan modificarlo. Dispone de una vista de comparación entra el recurso local y remoto en caso que exista conflicto entre la versión del recurso local con el remoto, muestra una vista del historial de versiones de los recursos con un conjunto de atributos de las acciones realizadas sobre el recurso.

Soporta conectarse a varios repositorios de control de versiones al mismo tiempo, permitiendo hacer operaciones sobre el repositorio directamente.

Es un plug-in muy útil para el desarrollo colaborativo, en el que intervienen un conjunto de desarrolladores trabajando sobre el mismo proyecto, poniendo a disposición del equipo de desarrollo facilidades para el trabajo en equipo.

Capítulo 3

Estructura lógica y física

En este capítulo se analiza la estructura de SIGEPOL, en términos de la lógica de los componentes, de la agrupación en distintas capas y niveles que se comunican entre sí y con otros clientes y aplicaciones. Las capas se refieren a la división lógica de componentes y funcionalidades, y no tienen en cuenta la ubicación física de los componentes en diferentes servidores o en diferentes lugares. Los niveles se refieren a la distribución física de los componentes y las funcionalidades en diferentes servidores, computadoras, redes y ubicaciones remotas. (29)

3.1 Organización general

La aplicación SIGEPOL está dividida en módulos, esta es una división lógica, agrupando las funcionalidades afines en un módulo. Todos los módulos están desarrollados sobre la plataforma J2EE usando las funcionalidades que brindan Spring Framework, JasperReports, Dojo, iBATIS, etc.

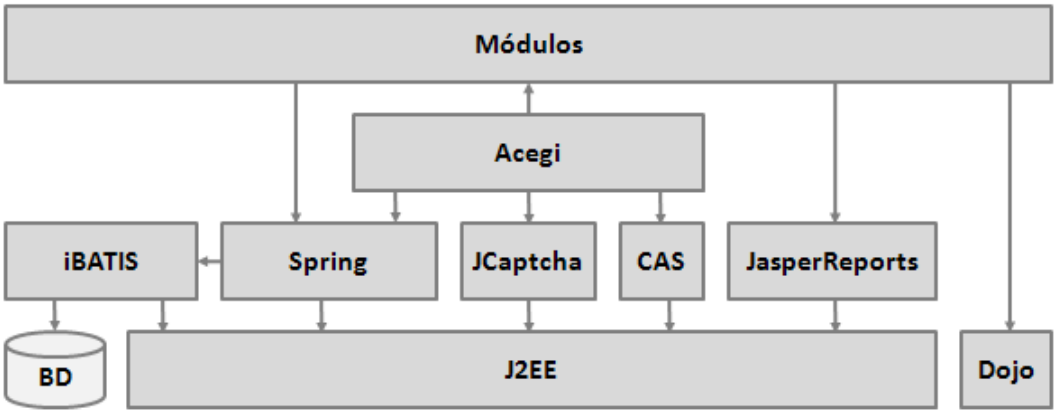


Figura 14 Estructura general de SIGEPOL

Los módulos existentes son:

- ✓ Denuncias
- ✓ Reseñas
- ✓ Dependencias
- ✓ Operativos Policiales
- ✓ Administración
- ✓ Común

El módulo **común** agrupa las funcionalidades comunes a todos los módulos del sistema.

3.2 Estructura de un módulo

Un módulo es un conjunto de funcionalidades acopladas convenientemente para realizar un conjunto de procesos de negocios comunes dentro del sistema. Cada uno de los módulos de SIGEPOL tiene una estructuración lógica en 3 capas: Presentación, Negocio y Acceso a Datos.

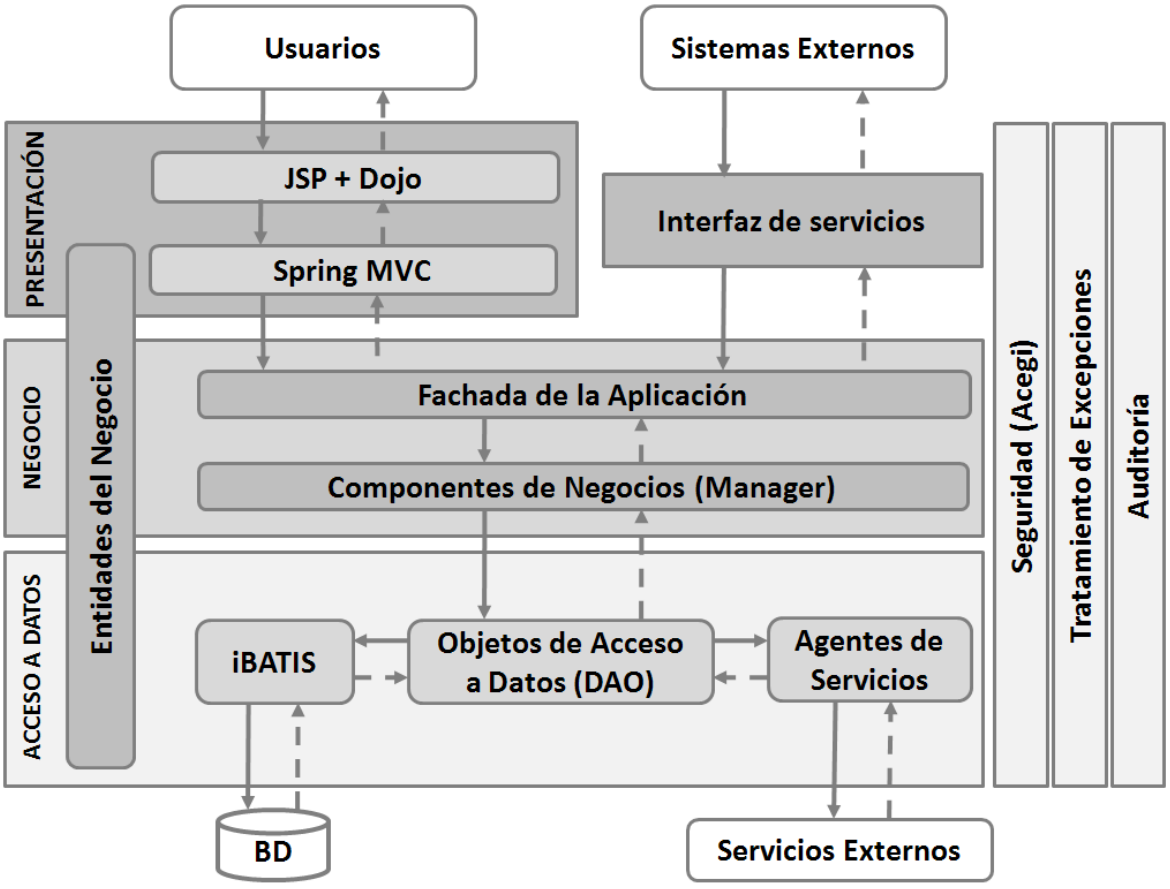


Figura 15 Estructura lógica de un módulo.

Una característica fundamental de las aplicaciones web que son sistemas distribuidos, siguiendo el patrón arquitectónico cliente – servidor. SIGEPOL está dividido en tres niveles físicos: Clientes, Servidor web, Servidor de base de datos.

El cliente será representado por un navegador web donde se ejecutará parte de la lógica de presentación. Las capas más importantes de la aplicación correrán en el servidor web Apache Tomcat, que incluye parte de la capa de presentación, y las capas de negocio y acceso a datos. La aplicación y los datos estarán físicamente separados, estando estos

últimos alojados en un servidor de datos administrados por Oracle 10g. La Figura 16 muestra como quedan distribuidas las capas lógicas en los niveles físicos de la aplicación.

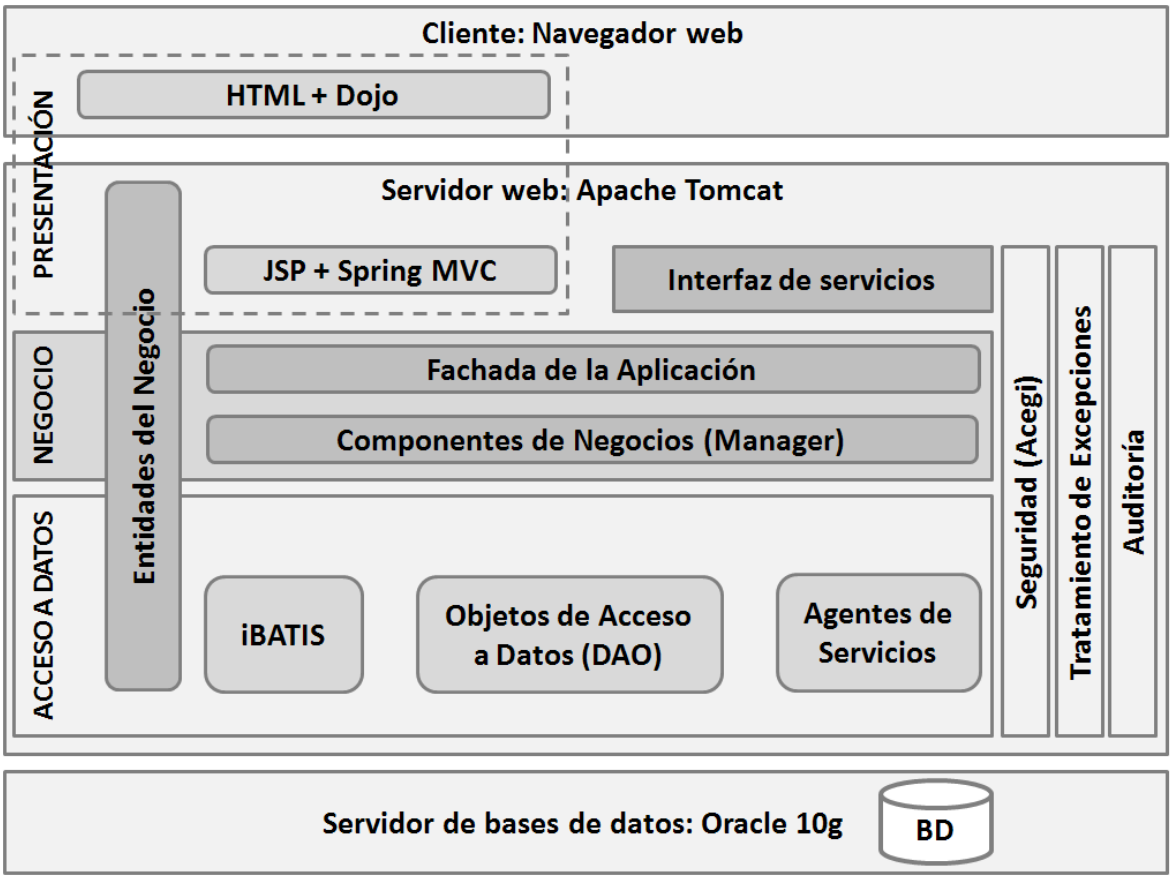


Figura 16 Niveles físicos de la aplicación

3.3 Capa de Presentación

La capa de presentación contiene los componentes que implementan y muestran la interfaz de usuario (UI[34]), además administra la interacción con el usuario y ejecuta la lógica asociada con la validación de los datos.

Los componentes de UI proporcionan una forma para que los usuarios interactúen con la aplicación, mostrando los datos en el formato adecuado. Asimismo, obtienen y validan los datos entrados por el usuario. Para mostrar la información se utilizaron los componentes de UI de HTML proporcionados por las páginas JSP y los widget[35] de Dojo.

Lógicamente la capa de presentación es una sola, pero atendiendo a su distribución física se puede ver dividida en dos: del lado del cliente y del lado del servidor. Del lado del

[34] Siglas del inglés User Interface
[35] Componente de UI que el programador reutiliza y tienen un gran valor para el usuario.

cliente la capa de presentación realizará la interacción con el usuario mediante componentes de entrada y salida de información y componentes de control de procesos de interfaz de usuario. Estos componentes de control de UI serán escritos es Java Script mediante de la utilización del API de Dojo. Además realizará la lógica asociada a la validación de los datos proporcionados por el usuario.

Del lado del servidor la capa de presentación será gestionada usando Spring MVC[36]. Como medida de seguridad siempre se realizará la validación de los datos en el servidor, independientemente a la validación hecha en el cliente.

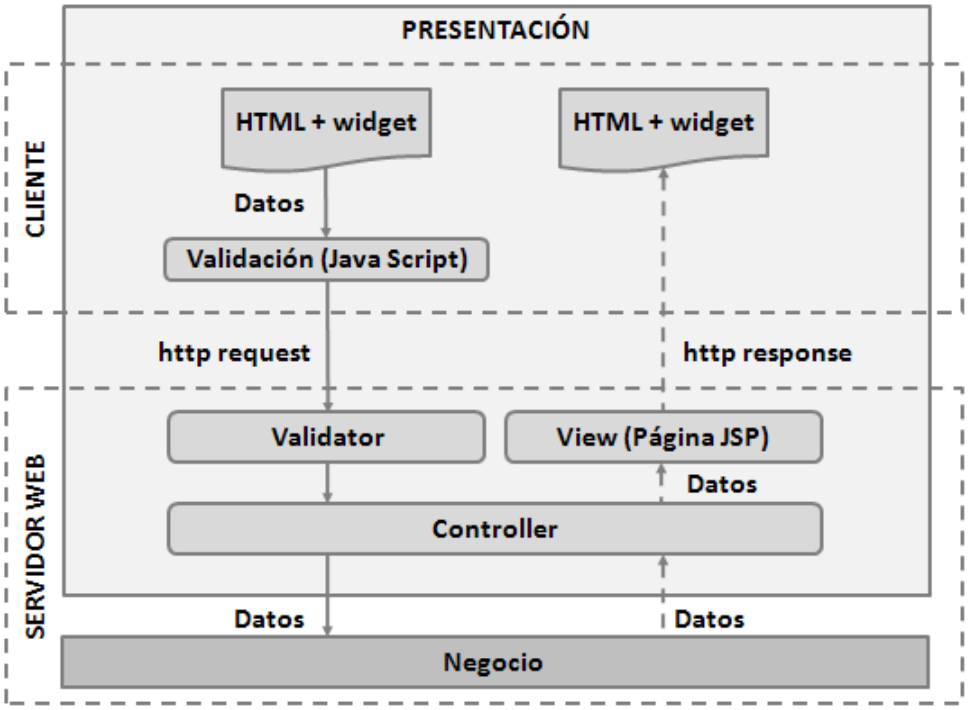


Figura 17 Componentes de la capa de Presentación

3.4 Capa de Negocio

En esta capa se implementa el núcleo de las funcionalidades de la aplicación y encapsula la lógica de negocio más relevante. Esta se compone por tres partes fundamentales: la fachada del negocio del caso de uso, entidades del negocio y los componentes de negocio (Managers).

La fachada es un conjunto de clases que agrupa a todas las funcionalidades del negocio del módulo, dividas por caso de uso, para optimizar el intercambio de datos entre el negocio y la capa superior. Para lograr un bajo acoplamiento entre la capa de negocio y la

[36] Ver el epígrafe: 2.2.2 Spring MVC

capa de presentación, cada fachada esta diseñada como una interfaz que define todas las funcionalidades correspondiente a un caso de uso y una clase concreta que implementa dichas funcionalidades, haciendo uso de la lógica de negocio contenida en los Managers.

Las entidades de negocio se utilizan para transmitir datos entre componentes. Estos datos suelen encapsularse en entidades informativas, que representan objetos o fenómenos del mundo real, y no son más que clases cuyo valor fundamental reside en la información que representa y no en su comportamiento en sí.

Por cada entidad del negocio que exista en el módulo, se crea un Manager que es el encargado de implementar la lógica de negocio asociada a dicha entidad. Los Managers incluyen además las operaciones más comunes como Insertar, Actualizar, Eliminar, etc. Los Managers se encargan de interactuar con la capa de acceso a datos.

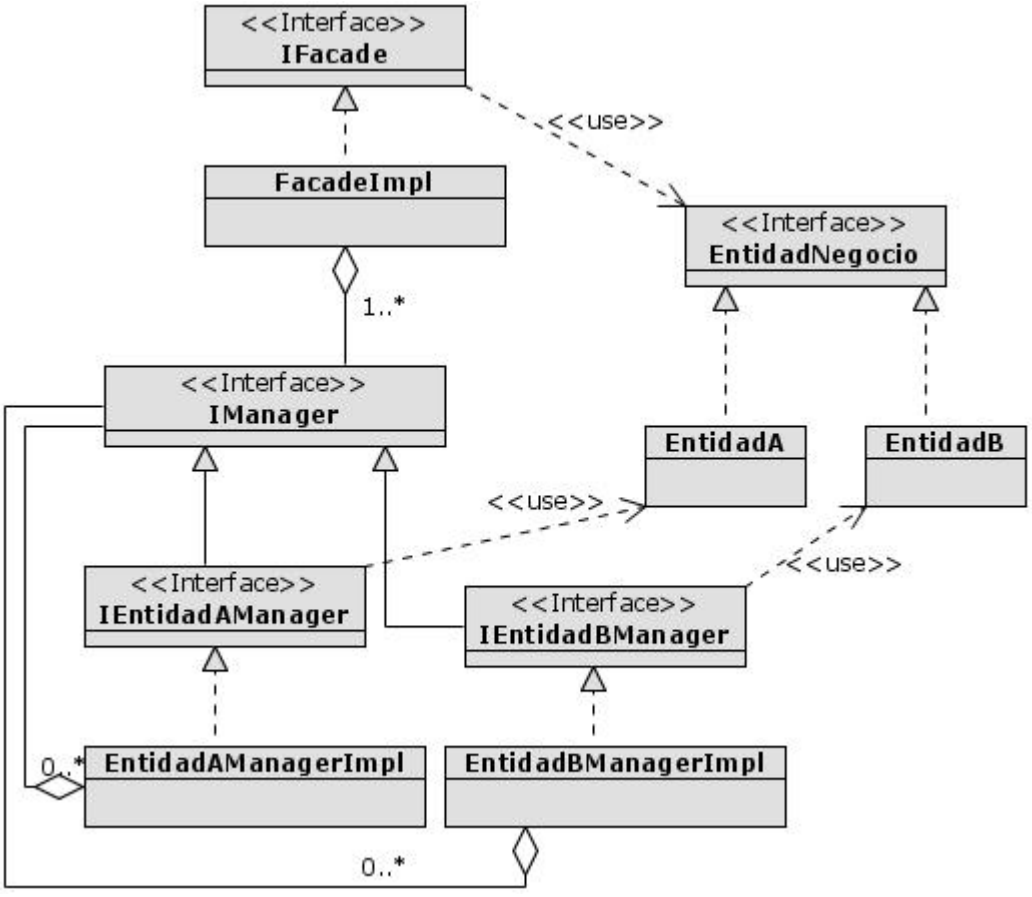


Figura 18 Componentes de la capa de negocio.

3.5 Capa de Acceso a Datos

La capa de acceso a datos es una abstracción que facilita y estandariza el acceso a los datos de la aplicación, encapsulando la lógica inherente a la tecnología usada para la

gestión de los datos. Esta capa esta compuesta por Objetos de Acceso a Datos (DAO[37]) y Agentes de Servicios.

Los DAOs son los encargados de encapsular la lógica de acceso a los datos, de esta manera se centraliza dicha funcionalidad, haciendo más fácil el mantenimiento y la configuración de la aplicación. La información pude ser suministrada por la base de datos de la aplicación o por servicios externos a la aplicación.

El Spring ORM Framework contiene un API para el desarrollo de objetos de acceso a datos. El paquete **org.springframework.orm.ibatis.support** da soporte para la implementación de DAOs que acceden a la base de datos a través de los mecanismos brindados por iBATIS.

Los Agentes de Servicios son entidades que implementan la semántica implícita al consumo de los datos subministrados por servicios externos. Además, realizan la traducción necesaria de los datos obtenidos al formato de los datos en la aplicación. Mediante el uso de Spring Web Service los agentes de servicio obtienen la información expuesta por los servicios web.

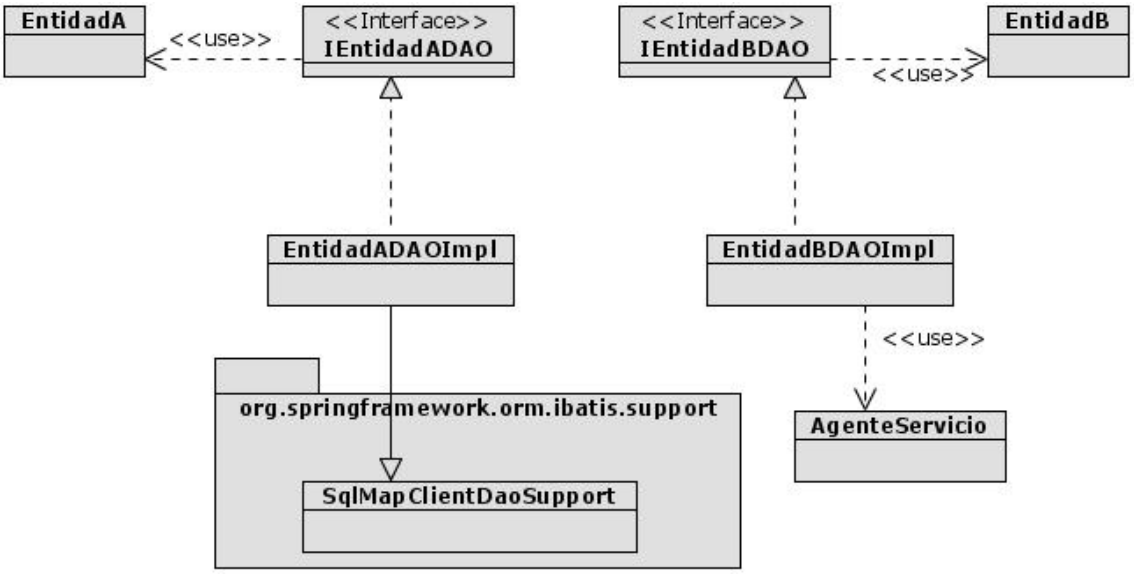


Figura 19 Componentes de la capa de acceso a datos

3.6 Interfaz de servicios

La interfaz de servicios provee a otras aplicaciones acceso a una parte de la lógica de negocio implementada en el sistema. Esta actúa en forma de fachada de la aplicación y utiliza servicios web como mecanismo para exponer estas funcionalidades.

[37] Siglas del inglés Data Access Object.

La interfaz de servicios está implementada con la utilización de Spring Web Services, haciendo uso del protocolo SOAP[38] para la comunicación entre la aplicación y otros sistemas externos (ver Anexo 1).

3.7 Integración entre las capas

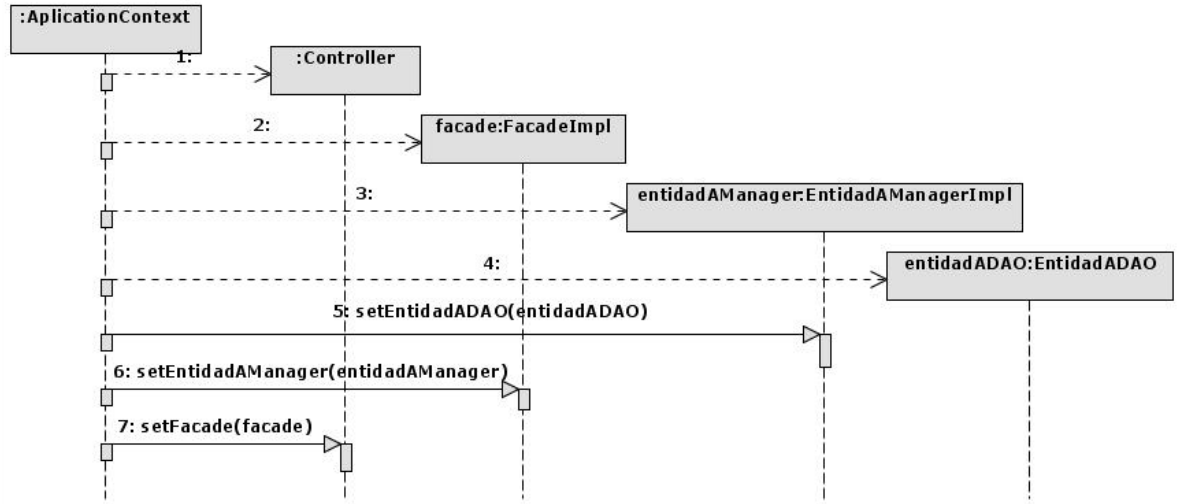


Figura 20 Acoplamiento entre las capas del sistema.

La capa de presentación, mediante los controladores, interactúa con la capa de negocio a través de la fachada que expone dicha capa. El módulo de IoC de Spring provee a la capa de presentación de la implementación concreta de la fachada. De la misma manera la capa de negocios, mediante los Managers, interactúa con la capa de acceso a datos a través de las interfaces de acceso a datos que expone dicha capa. Igualmente el módulo de IoC de Spring provee la implementación concreta de los DAOs. El flujo de información entre las capas se realiza a través de las entidades del negocio (ver Anexo 2).

3.8 Seguridad

La seguridad es un aspecto importante para la protección de la integridad y la privacidad de los datos y los recursos de una aplicación web. En el diseño de una estrategia de seguridad para una aplicación web se deben utilizar soluciones de probada eficacia, así como instrumentar mecanismos para la autenticación, autorización y validación de datos que protejan la aplicación de una serie de amenazas.

[38] Siglas del inglés Simple Object Access Protocol.

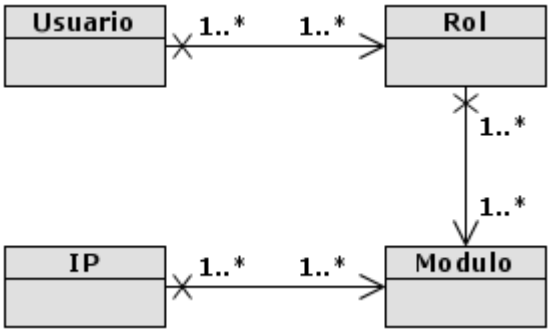


Figura 21 Diagrama de clases de la definición de roles y usuarios.

En SIGEPOL se definió un modelo basado en roles. En este modelo a los usuarios le son asignados uno o varios roles mientras que estos son asociados con uno o varios módulos del sistema en los que tendrán determinados permisos y privilegios. Además, los módulos solo pueden ser ejecutados desde determinadas estaciones de trabajo representadas por su dirección IP. De esta manera un usuario puede tener diferentes niveles de privilegios al acceder en la aplicación, en dependencia de la estación de trabajo en la que se autentique.

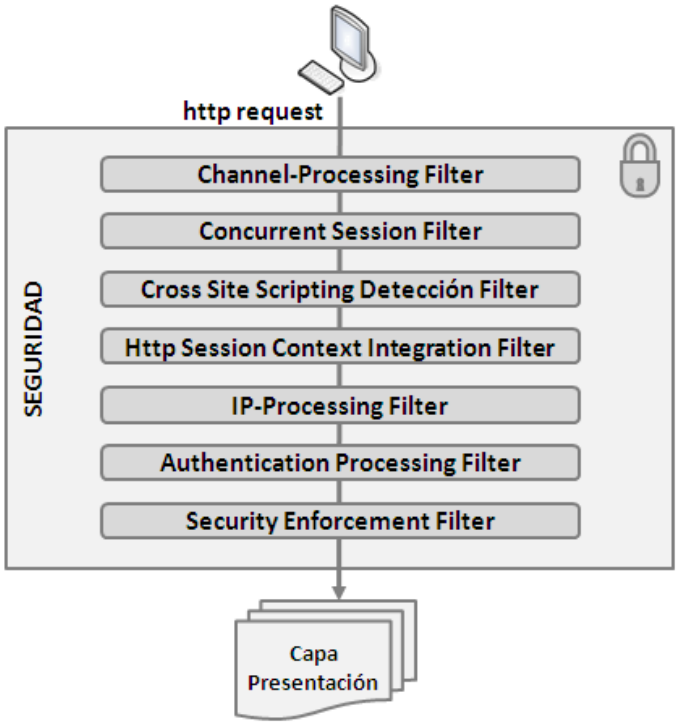


Figura 22 Flujo de una petición a través de los filtros.

Acegi brinda mecanismos para la seguridad de aplicaciones web basándose en la aplicación de filtros. Estos filtros interceptan una petición aplicando criterios de seguridad antes que la misma sea tratada por la aplicación. Acegi provee un conjunto de filtros que

pueden ser aplicables en dependencia de las necesidades de la aplicación. Además pueden ser desarrollados filtros adicionales que establezcan nuevos requisitos de seguridad. La Figura 22 muestra los filtros aplicados a todas las peticiones en SIGEPOL.

Channel-Processing Filter será el primero en ocuparse de una solicitud. Este filtro examinará el canal por el cual fue formulada la solicitud (normalmente HTTP o HTTPS) y determina si el canal cumple con los requisitos de seguridad establecidos. Si no es así, la solicitud se redirige a la misma URL, alterando el canal para satisfacer los requisitos de seguridad.

Concurrent Session Filter se encarga de determinar si la sesión ha expirado. En caso positivo invalida la sesión y redirecciona la petición hacia una página de expiración de sesión, en la que se le informa al usuario que su sesión ha expirado y que debe autenticarse nuevamente en la aplicación para continuar su labor. En caso contrario reinicia el contador de tiempo de inactividad de la sesión y deja que la petición siga su curso.

Cross Site Scripting Detection Filter se encarga de revisar todos los parámetros que viajan en la petición verificando que no contengan caracteres o fragmentos de cadenas que pudieran propiciar algún tipo de ataque Cross Site Scripting (XSS[39]). Los ataques XSS engloban cualquier intento de ejecutar código de *scripting*, como JavaScript, con el objetivo de afectar, cambiar o agregar funcionalidades a la aplicación. Las peticiones que se detecte contengan algún tipo de material no confiable serán redireccionadas hacia una página que informe al usuario los motivos por los cuales no fue aceptada su petición.

Http Session Context Integration Filter se encarga de almacenar información de seguridad, como por ejemplo la sesión HTTP.

IP-Processing Filter evita que un usuario autenticado cambie de dirección IP y continúe autenticado. Básicamente comprueba por cada petición que la dirección IP sea la misma que la que se utilizó al autenticarse. En caso que se detecte algún cambio en la dirección IP la sesión queda invalidada obligando al usuario a autenticarse nuevamente. De esta manera se obliga al usuario a acceder a la aplicación desde un IP de confianza, y además evita que un usuario pueda apoderarse de una sesión ajena.

Authentication Processing Filter este filtro detectará si se esta intentando acceder a algún recurso del sistema sin estar autenticado, en ese caso, desencadenará el proceso

[39] Renombrados así para evitar confusiones con las hojas de estilo en cascada: CSS (Cascading Style Sheets)

de autenticación. El proceso de autenticación consiste en obtener las credenciales del usuario (nombre de usuario y contraseña) y comprobar su validez. Además, se asegura que el usuario acceda desde una estación de trabajo de confianza para el sistema. Si en este proceso se introducen erróneamente las credenciales del usuario 3 veces, se incluirá un test de detección de ataques automáticos (Captcha).

Security Enforcement Filter instrumenta el proceso de autorización. Una vez que la petición llega a este filtro, ya ha pasado por una serie de restricciones previas que incluyen la autenticación del usuario, se determina si el usuario tiene los privilegios necesarios para acceder al recurso solicitado por la petición. De ser así, esta continuaría su curso, en caso contrario se redireccionaría hacia una pagina donde se informe al usuario que no pose los privilegios necesarios para acceder al recurso deseado.

3.9 Tratamiento de excepciones

Diseñar una buena estrategia de gestión de excepción es importante para la seguridad y la fiabilidad de la aplicación. No hacerlo puede implicar que la aplicación sea vulnerable a ataques de denegación de servicio, o revele información sensible o confidencial.

Ante la posibilidad de que ocurra un error inesperado, y es muy probable que ocurra si de software se trata, la aplicación debe ser capaz de mostrar un mensaje amigable al usuario, evitando revelar cualquier tipo de información asociada a la excepción producida.

Spring provee un mecanismo que permite tener un control centralizado de de las excepciones no tratadas en el sistema. **SimpleMappingExceptionResolver** es la clase que instrumenta este mecanismo. Mediante su uso es posible mapear las excepciones que se produzcan con una vista que informe al usuario del error ocurrido, mediante un mensaje adecuado. Además permite agregar cualquier lógica asociada a una excepción. En SIGEPOL se guarda la traza de cada excepción no tratada que ocurra en el sistema, con el fin de tomar acciones que contrarresten la ocurrencia de errores.

3.10 Auditoría

La auditoría no es más que un conjunto de procedimientos y técnicas para evaluar y controlar, total o parcialmente, un sistema, con el fin de proteger sus activos y recursos, verificar si sus actividades se desarrollan eficientemente y de acuerdo con la normativa informática y otras existentes en cada empresa.

Por la sensibilidad de los datos que se manejan en SIGEPOL se guarda un registro de las acciones del usuario más importantes, como el registro de datos, modificación y acceso a

los mismos. Estos registros tienen gran utilidad en la realización de auditorias que ayuden a encontrar brechas de seguridad en el sistema.

```
<bean id="sigepolAudit"
    class="sigepol.core.audit.advice.AuditMethodInterceptor">
    <property name="sqlMapClientTemplate" ref="sqlMapClientTemplate"/>
</bean>

<bean id="patronPointCut"
    class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="patterns">
        <value>
            insertar,registrar,modificar,eliminar,buscar,obtener
        </value>
    </property>
    <property name="advice">
        <ref local="sigepolAudit"/>
    </property>
</bean>

<bean name="auditProxy"
    class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="beanNames" value="*DAO*"/>
    <property name="interceptorNames" value="patronPointCut"/>
</bean>
```

Figura 23 Configuración del aspecto para la auditoría de los DAOs.

Para la instrumentación de este mecanismo se utilizó el módulo Spring AOP[40]. Este módulo provee una interfaz **Advice** para la implementación de la lógica del aspecto y una clase **ProxyFactory** para crear un proxy al objeto concreto (target). En SIGEPOL se diseñó un aspecto que intercepta las llamadas a los métodos *registrar*, *modificar*, *eliminar*, *obtener* y *buscar* de todos los objetos de acceso a datos (DAO) de la aplicación y registra las incidencias de estas llamadas.

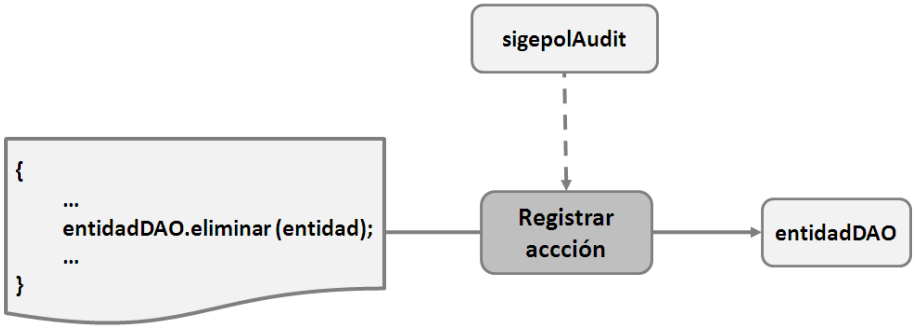


Figura 24 Auditoría sobre una operación de acceso a datos.

[40] Ver el epígrafe: 2.2.3 Spring AOP.

3.11 Modelo de despliegue

Un diagrama de despliegue modela la topología del hardware sobre el cual se ejecuta un sistema. Muestra la configuración de los nodos que participan en la ejecución y de los componentes que residen en ellos. Un nodo es un elemento físico que existe en tiempo de ejecución. Representa un recurso computacional que, por lo general, tiene memoria y capacidad de almacenamiento. Representa a un procesador o un dispositivo sobre el cual se despliegan los componentes.

El diagrama de despliegue del SIGEPOL está formado por servidores, PC clientes e impresoras como dispositivos necesario para lograr un correcto funcionamiento del sistema. Las impresoras estarán conectadas a las PCs clientes debido a que desde la aplicación se podrán imprimir documentos generados por ella.

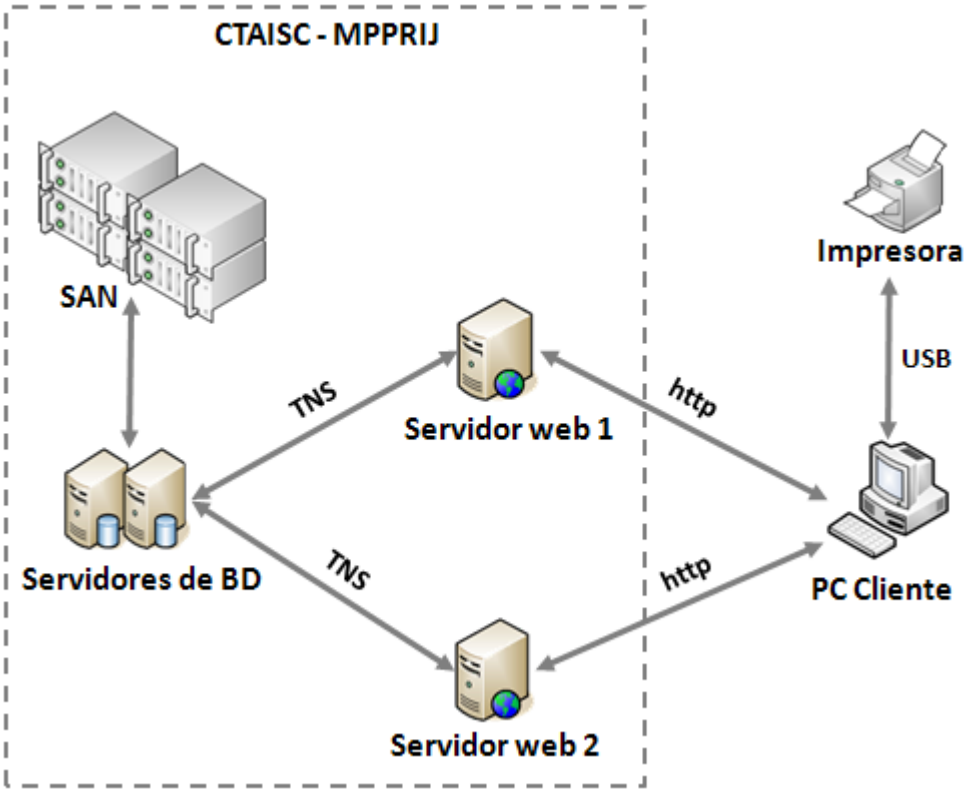


Figura 25 Diagrama de despliegue del sistema

3.11.1 SAN

Red de área de almacenamiento (de sus siglas en inglés Storage Area Network). Es una red concebida para conectar servidores, arreglos de discos y librerías de respaldo principalmente. Su función es la de conectar de manera rápida, segura y confiable los distintos elementos que la conforman.

Requerimientos de Hardware

- ✓ 500 GB de capacidad de almacenamiento.

3.11.2 Servidores de Bases de Datos

Los servidores de Base de Datos van a contener la base de datos de SIGEPOL que tendrá un alcance nacional, almacenado la información generada en cualquier dependencia policial del país conectada al sistema.

Requerimientos de Hardware para cada Servidor

- ✓ 4 Procesadores a 1.6 GHz.
- ✓ 24 GB de Memoria RAM.
- ✓ 2 discos duros de 72GB a 7200 rpm en RAID1[41].
- ✓ Tarjeta de Red de 10/100/1000 Gigabit.

Requerimientos de Software para cada Servidor

- ✓ Red Hat Linux Advanced Server 4.0 (RHELAS4).
- ✓ Oracle Database Enterprise Edition 10g R2.
- ✓ Oracle Real Application Cluster 10g.

La comunicación de los servidores de BD con la aplicación se realiza mediante TNS (Transparent Network Substrate), que es una capa de comunicación que utilizan las bases de datos Oracle.

3.11.3 Servidores web

Los servidores web contendrán la aplicación en sí, en un servidor web estarán ubicados los módulos de Denuncias, Administración y Operativos Policiales. En el otro servidor Web estarán ubicados los módulos de Reseña, Dependencias y el CAS.

Requerimientos de Hardware para cada Servidor

- ✓ 2 Procesadores a 3 GHz
- ✓ 8 GB de Memoria RAM.
- ✓ 2 discos duros de 72GB a 7200 rpm en RAID1.
- ✓ Tarjeta de Red de 10/100/1000 Gigabit.

[41] Siglas del inglés Redundant Array of Independent Disks (conjunto redundante de discos independientes). Hace referencia a un sistema de almacenamiento que usa múltiples discos duros entre los que distribuye o replica los datos. Un RAID 1 crea una copia exacta (o espejo) de un conjunto de datos en dos o más discos.

Requerimientos de Software para cada Servidor

- ✓ Red Hat Linux Advanced Server 4.0 (RHELAS4).
- ✓ JVM 1.6 o superior.
- ✓ Apache Tomcat 5.5.17.
- ✓ En el servidor 1 estarán ubicados los módulos de Dependencias, Administración y Operativos Policiales del SIGEPOL.
- ✓ En el servidor 2 se instalarán los módulos de Reseña y Denuncias del SIGEPOL.

3.11.4 PCs cliente

Las PCs clientes deberán tener acceso, vía web, a los servidores y deberán tener instalado un navegador web para poder hacer uso de la aplicación.

Requerimientos de Hardware

1. Para el navegador web Mozilla Firefox 2.0
   - ✓ Procesador mínimo: 233 MHz
   - ✓ Procesador recomendado: 500 MHz
   - ✓ RAM mínima: 64 MB.
   - ✓ RAM recomendada: 256 MB.
   - ✓ Espacio mínimo de disco duro: 50 MB disponibles.
   - ✓ Espacio de disco duro recomendado: 100 MB disponible.
2. Para el navegador web Internet Explorer 7
   - ✓ Procesador mínimo: 233 MHz
   - ✓ RAM mínima: 64 MB.
   - ✓ RAM recomendada: 128 MB.

Requerimientos de Software

- ✓ Navegador web (Internet Explorer 7, Mozilla Firefox 2.0).

Capítulo 4

Estilo de desarrollo

En este capítulo se describe la estructura de desarrollo establecida en SIGEPOL, así como las pautas de codificación, que incluyen convenciones de nombres, organización del sistema de archivos y buenas prácticas de codificación, que serán de gran utilidad para el equipo de desarrollo. Además se presenta la estructuración en roles del equipo desarrollo y las responsabilidades de cada miembro.

4.1 Pautas de codificación

Las pautas de codificación o convenciones de código son de vital importancia para el desarrollo de una aplicación, ya que dictaminan estándares a seguir por todo el equipo de desarrollo en la escritura del código fuente. Las convenciones de código son importantes para los desarrolladores por un buen número de razones: (30)

- ✓ El 80% del coste del código de un programa va a su mantenimiento.
- ✓ Casi ningún software lo mantiene toda su vida el autor original.
- ✓ Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo más rápidamente y a fondo.
- ✓ Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.

Para que funcionen las convenciones, cada miembro del equipo de desarrollo debe conocerlas y aplicarlas.

Para el desarrollo de SIGEPOL se siguieron la mayoría de las pautas propuestas por *Sun Microsystems* para la escritura de códigos fuentes en Java y páginas JSP y las propuestas por *The Dojo Foundation* para los ficheros JavaScript.

4.1.1 Convenciones de nombres

Las convenciones de nombres hacen los programas más entendibles haciéndolos más fácil de leer. También pueden dar información sobre la función de un identificador, por ejemplo, cuando es una constante, un paquete, o una clase, que puede ser útil para entender el código. A continuación se muestra una tabla con los principales tipos de identificadores, las reglas que rigen su nombramiento y ejemplos.

Tipo de identificador	Regla	Ejemplos
Paquetes	lowercase[42]	sigepol
Clases	Los nombres de las clases deben ser sustantivos, intentando mantener los nombres de las clases simples y descriptivos. Usar palabras completas. Usar el estilo CamelCase[43]	Cliente ArmaFuego
Interfaces	Los nombres de las interfaces siguen la misma regla que las clases. Adicionándolo un "I" delante.	ICliente IArmaFuego
Métodos	Los métodos deben ser verbos que indiquen la acción que van a ejecutar. Usar el estilo lowerCamelCase[44].	buscar() eliminarTodos()
Variables	Excepto las constantes, todas las instancias, variables de clase o métodos y parámetros usaran el estilo lowerCamelCase. Los nombres de las variables deben ser cortos pero con significado, que pueda indicar a un observador casual su función.	arma ancho
Constantes	Usar el estilo UPPER_CASE[45]	MAXIMA_ALTURA PI

Otras convenciones de nombres importantes son:

1. Los métodos de acceso deben utilizar los prefijos *get* y *set*.
2. Los nombres de los controles de interfaz de usuario (UI) deben ser utilizados como sufijos de las variables de este tipo. Por ejemplo: leftComboBox, topScrollPane.
3. Para nombrar colecciones debe hacerse en plural.
4. Las variables de control de ciclos deben ser "i", "j", "k", etc.
5. Evitar usar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML)
6. Las clases que representan excepciones deben ser nombradas con los sufijos "Exception" o "Error".

4.1.2 Organización de los ficheros

Para la organización la aplicación se definieron directorios de código fuente por cada módulo contenido en SIGEPOL. Además, se definieron dos directorios especiales: **comun** [46] y **core**.

Con el objetivo de lograr el mayor grado posible de autonomía en los módulos, las funcionalidades comunes a dos módulos o más son definidas en el directorio **comun**.

[42] Todas las letras en minúscula.
[43] Primera letra de cada palabra en mayúscula.
[44] Primera palabra en minúscula, usar el estilo CamelCase en las restantes.
[45] Todas las letras en mayúscula usando el símbolo "_" entre palabras.
[46] Al referirse al nombre de un directorio de código fuente no puede escribirse con tilde.

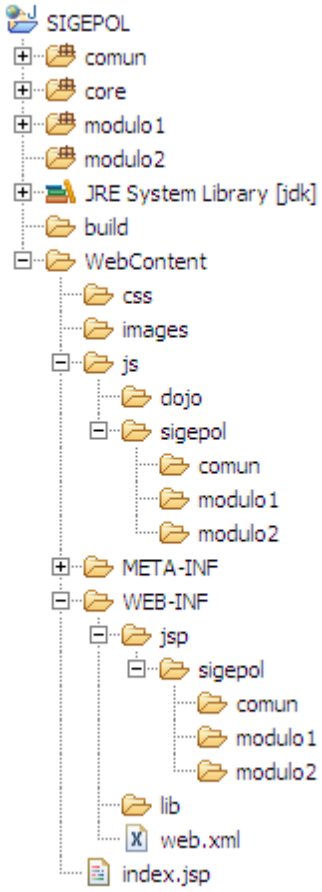


Figura 26 Organización de los ficheros en SIGEPOL.

El directorio **core** está compuesto por las clases que sustentan la arquitectura, además de las distintas configuraciones generales del sistema, como son las configuraciones de acceso a la base de datos y de seguridad.

La unidad básica de organización en Java son los paquetes, que son representados físicamente por un directorio en el sistema de archivos. Cada uno de los módulos del sistema presenta una estructura en paquetes similar. Esta estructuración en paquetes responde a la organización en capas lógicas de cada uno de los módulos (ver Figura 27).

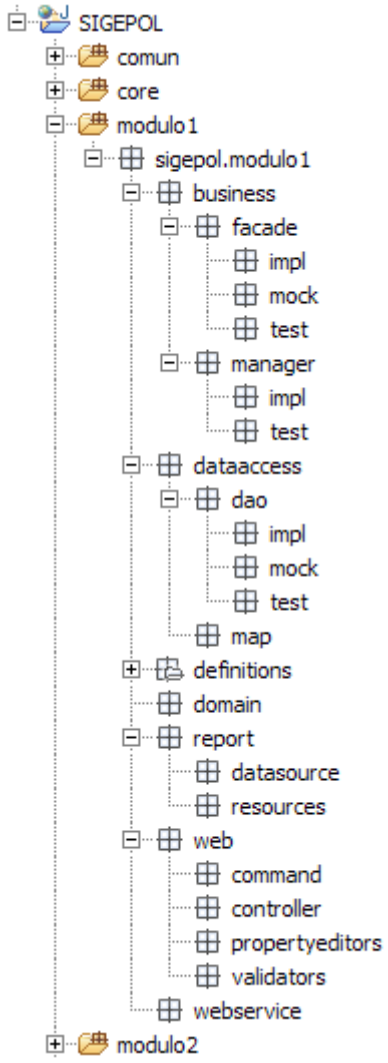


Figura 27 Estructura de paquetes de un módulo.

sigepol.modulo: paquete raíz de cada módulo, a partir del cual se crearán los otros paquetes del módulo.

sigepol.modulo.definitions: en este paquete estarán todos los ficheros XML de definiciones de objetos (beans) que serán incluidos en contexto de Spring.

- ✓ **sigepol-modulo-context.xml**: Definición de los objetos de negocio del módulo.
- ✓ **sigepol-modulo-mock-context.xml**: Definición de los objetos falsos de negocio del módulo.
- ✓ **sigepol-modulo-dataaccess-context.xml**: Definición de los objetos de acceso a datos del módulo.
- ✓ **sigepol-modulo-dataaccess-mock-context.xml**: Definición de los objetos falsos de acceso a datos del módulo.

- ✓ **sigepol-modulo-menu-context.xml**: Definición de las funcionalidades que se van a mostrar en el menú.
- ✓ **sigepol-modulo-report-context.xml**: Definición de los objetos de reportes que se van a utilizar en el módulo.
- ✓ **sigepol-modulo-security-context.xml**: Definición de la seguridad del módulo.
- ✓ **sigepol-modulo-service-context.xml**: Definición de los objetos de servicios web.
- ✓ **sigepol-modulo-servlet.xml**: Definición de los objetos asociados a la capa de presentación que estará en el servidor de aplicaciones (Controladores, Validadores, etc.).

Todos los ficheros XML que contengas definiciones de beans serán nombrados con el patrón ***-context.xml**.

sigepol.modulo.domain: en este paquete estarán definidas todas las entidades del negocio que pertenecen al módulo.

sigepol.modulo.web: en este paquete estará toda la lógica asociada a la parte de la capa de presentación incluida en el servidor de aplicaciones.

sigepol.modulo.web.**controller**: aquí estarán todas las clases controladoras que atenderán las peticiones hechas desde el cliente. Estos ficheros serán nombrados con el patrón ***Controller.java**.

sigepol.modulo.web.**command**: incluirá las clases asociadas a los datos que provienen del navegador web. Spring las utiliza para convertir los datos que vienen en el formulario en un objeto Java. Estos ficheros serán nombrados con el patrón ***Command.java**.

sigepol.modulo.web.**propertyeditor**: incluirá las clases utilizadas para convertir una cadena de caracteres que proviene de un formulario a un objeto Java específico. Estos ficheros serán nombrados con el patrón ***PropertyEditor.java**.

sigepol.modulo.web.validators: en este paquete se encontraran los validadores de datos que provienen de los formularios web. Estos ficheros serán nombrados con el patrón ***Validators.java**.

sigepol.modulo.webservices: en este paquete estará toda la lógica de servicios web que va a exponer el módulo a otros sistemas.

sigepol.modulo.reports: estará toda la lógica asociada a los reportes.

sigepol.modulo.reports.resource: incluirá todos los recursos necesarios para generar los reportes tales como: los archivos ***.jasper**, que definen el diseño del reporte, así como archivos de imágenes estáticas incluidas en el diseño del reporte.

sigepol.modulo.reports.datasource: contiene las implementaciones de los orígenes de datos de los reportes. Estos ficheros serán nombrados con el patrón ***DataSource.java**.

sigepol.modulo.business: este paquete incluirá toda la lógica de negocio del módulo.

sigepol.modulo.business.facade: este paquete incluirá la interfaces de las fachadas de negocio asociadas al módulo. Estos ficheros serán nombrados con el patrón ***Facade.java**.

sigepol.modulo.business.facade.impl: incluirá la implementación de las fachadas del módulo, la cuales serán nombradas con el patrón ***FacadeImpl.java**.

sigepol.modulo.business.facade.mock: incluirá las implementaciones falsas de las fachadas del módulo, la cuales serán nombradas con el patrón ***FacadeMock.java**.

sigepol.modulo.business.facade.test: incluirá las implementaciones de los objetos de pruebas de unidad de las fachadas del módulo, la cuales serán nombradas con el patrón ***FacadeTest.java**.

sigepol.modulo.business.manager: este paquete incluirá las definiciones (interfaces) de los managers asociados a las entidades del negocio que existan en el módulo. Estos ficheros serán nombrados con el patrón **IEntidadManager.java**, existirá uno por cada entidad.

sigepol.modulo.business.manager.impl: incluirán todas las implementaciones de los managers del módulo. Estos ficheros serán nombrados con el patrón **EntidadManagerImpl.java,** existirá uno por cada entidad.

sigepol.modulo.business.manager.test: incluirán todas las implementaciones de los objetos de prueba de los managers del módulo. Estos ficheros serán nombrados con el patrón **EntidadManagerTest.java,** existirá uno por cada manager.

sigepol.modulo.dataaccess: este paquete incluirá toda la lógica asociada al acceso a datos del módulo.

sigepol.modulo.dataaccess.dao: incluirá todas las definiciones de los objetos de acceso a datos. Estos ficheros serán nombrados con el patrón **IEntidadDAO.java**, existirá uno por cada entidad.

sigepol.modulo.dataaccess.dao.impl: en este paquete estarán las implementaciones de las interfaces de acceso a datos del módulo. Estos ficheros serán nombrados con el patrón **EntidadDAOImpl.java,** existirá uno por cada entidad.

sigepol.modulo.dataaccess.dao.mock: en este paquete estarán las implementaciones falsas de las interfaces de acceso a datos del módulo. Estos ficheros serán nombrados con el patrón **EntidadDAOMock.java,** existirá uno por cada entidad.

sigepol.modulo.dataaccess.dao.test: en este paquete estarán las implementaciones de los objetos de pruebas de los DAOs del módulo. Estos ficheros serán nombrados con el patrón **EntidadDAOTest.java,** existirá uno por cada entidad.

sigepol.modulo.dataaccess.map: incluirá los ficheros XML de mapeo para iBATIS.

Directorio	Ficheros que incluye
WebContent	Ficheros de recurso asociados a la capa de presentación, ficheros de script, hojas de estilos, imágenes, páginas JSP, etc.
WebContent\css	Hojas de estilos de la aplicación.
WebContent\images	Imágenes de la aplicación organizadas por modulo.
WebContent\js	Estarán todos los archivos JavaScript organizados por módulos, además de los ficheros de Dojo. Cada página de JSP tendrá asociado un fichero de JavaScript con igual nombre.
WebContent\WEB-INF	Todas las paginas JSP de la aplicación organizadas por módulos.

4.1.3 Consideraciones adicionales.

Sobre las variables:

1. Las variables deben ser inicializadas en el mismo lugar en que son declaradas, y con el menor alcance posible.
2. Una variable no debe tener más de un significado. En tal caso se deben usar dos o más variables.
3. El tiempo de vida de las variables debe ser el menor posible.
4. Declaraciones de ciclos:
    a) En la construcción de un ciclo “**for**” solo deben ser incluidas sentencias de control de ciclo.
    b) Las variables de control de ciclos deben ser inicializadas exactamente antes de comenzar la ejecución del mismo, en caso de sentencias “**for**” las variables de control deben ser inicializadas en la construcción de la sentencia.
5. Condicionales:

c) El uso de expresiones booleanas complejas debe ser evitado, en tal caso, use variables booleanas temporales.
d) El caso mas general debe ser colocado en la parte "**then**"(implícito) y las excepciones o casos menos generales deben ser situados en la parte "**else**" de una sentencia "**if**…**else**".
e) El uso de llamadas a métodos que realizan cálculos u operaciones no triviales deben ser evitados en las expresiones booleanas.

Sobre el uso de comentarios al código:

1. El código complejo no debe ser comentado, pero si reescrito.
2. Los comentarios deben ser identados con relación a su posición en el código, precediendo o a la derecha del código al que se refiere.
3. Se deben usar comentarios para explicar el significado de un segmento de código que por su importancia lo necesite.
4. No debe ser incluido un comentario a un código que su significado sea obvio.

4.2 Flujo de trabajo

El flujo de trabajo es un conjunto de actividades que guían a los desarrollares en la elaboración de los distintos componentes de la aplicación. Este flujo permite desarrollar las distintas capas a la vez, estableciendo una interacción adecuada entre los desarrolladores. Se definen un rol por cada capa lógica de la aplicación: Desarrollador de Presentación, Desarrollador de Negocio y Desarrollador de Acceso a Datos (ver Figura 28).

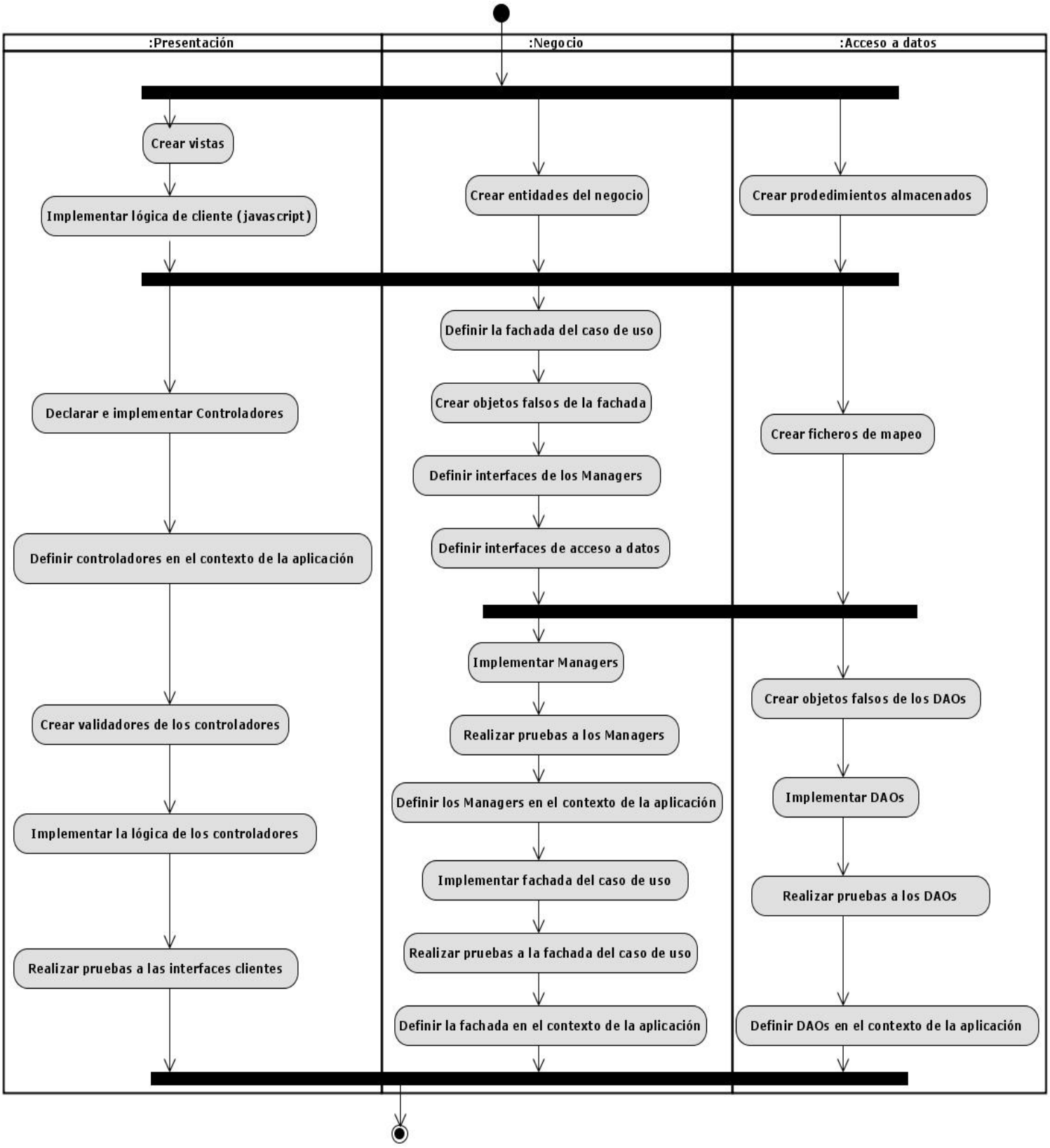


Figura 28 Flujo de trabajo

Desarrollador de acceso a datos

1. **Crear procedimientos almacenados**: Se implementaran los procedimientos almacenados necesarios para la interacción con la base de datos.
2. **Crear ficheros de mapeo:** Se mapean, en los archivos XML de iBATIS, los procedimientos almacenados con las clases de entidad.
3. **Crear objetos falsos de los DAOs**: Se implementan objetos falsos de los DAOs y los define en el dataaccess-mock-context.xml, de esta forma los desarrolladores de las capas superiores pueden realizar las pruebas a los componentes desarrollados.
4. **Implementar DAOs**: Implementar la lógica del acceso a datos.
5. **Realizar pruebas a los DAOs**: Realizar pruebas de unidad a los DAOs para verificar su correcto funcionamiento. En caso de que se detecten fallos se realizan los cambios pertinentes y se vuelven a ejecutar las pruebas de unidad.
6. **Definir DAOs en el contexto de la aplicación**: Definir en los ficheros de contexto de la aplicación correspondiente (dataaccess-context.xml) los DAOs que ya fueron testeados, reemplazando los DAOs falsos en su función.

Desarrollador de negocio

1. **Crear entidades del negocio**: Se crea el conjunto de entidades del negocio que se van a utilizar en el módulo donde se esté trabajando.
2. **Definir la fachada del caso de uso**: Se define una interfaz con las funcionalidades del negocio pertinentes para dar solución a un caso de uso en específico.
3. **Crear objetos falsos de la fachada**: Se implementan objetos falsos de las fachadas, con el fin de que se pueden realizar pruebas a los distintos componentes de la capa de presentación, y se definen en el mock-context.xml.
4. **Definir interfaces de los Managers**: Se definen las distintas funcionalidades que están asociadas a una entidad de negocio específica.
5. **Definir interfaces de acceso a datos**: Se define las interfaces de los DAOs.
6. **Implementar Managers**: Se implementa las funcionalidades definidas en las interfaces de los managers, conteniendo toda la lógica de negocio asociada a la entidad.
7. **Realizar pruebas a los Managers**: Comprobar el correcto funcionamiento de los Managers mediante las pruebas de unidad.

8. **Definir los Managers en el contexto de la aplicación**: Definir en el contexto de la aplicación los managers que fueron debidamente probados.
9. **Implementar fachada del caso de uso**: Se implementan todas las funcionalidades que se van a exponer a la capa de Presentación.
10. **Realizar pruebas a la fachada del caso de uso**: Se realizan pruebas de unidad a la fachada para comprobar su funcionamiento.
11. **Definir la fachada en el contexto de la aplicación**: Se define en contexto de la aplicación las fachadas que su funcionamiento se haya probado, reemplazando a las definiciones de las fachadas falsas en su función.

Desarrollador de presentación

Crear vistas: Se crearán las vistas que van a interactuar con el usuario, páginas JSP, ficheros de reportes, etc.

Implementar lógica de cliente (JavaScript): Se implementa la lógica de presentación que va a estar del lado del cliente, como la validación y formateado de los datos.

Declarar e implementar Controladores: Se declararán e implementan los controladores que van a interactuar con las vistas. En este paso son se incluirá lo indispensable para que una vista pueda ser mostrada.

Definir controladores en el contexto de la aplicación: Se definen los controladores en el contexto de la aplicación, mapeando las peticiones que atenderá y con cuales vistas va a interactuar.

Crear validadores de los controladores: Se implementan los validadores con la lógica de validación del lado del servidor.

Implementar la lógica de los controladores: Se implementa la lógica que va a ejecutar para atender la petición proveniente del cliente.

Realizar pruebas a las interfaces clientes: Se realizan pruebas a las interfaces de usuario desarrolladas.

4.2.1 Pruebas de unidad

Para la realización de las pruebas de unidad se implementa una clase que herede de la clase **sigepol.core.test.TestCase**, que hace uso del soporte brindado por Spring para el trabajo con JUnit. Todos los métodos de esta clase que comiencen por "*test*" serán

considerados casos de prueba. La Figura 29 muestra la realización de las pruebas de unidad a un Manager utilizando una implementación falsa de un objeto de acceso a datos.

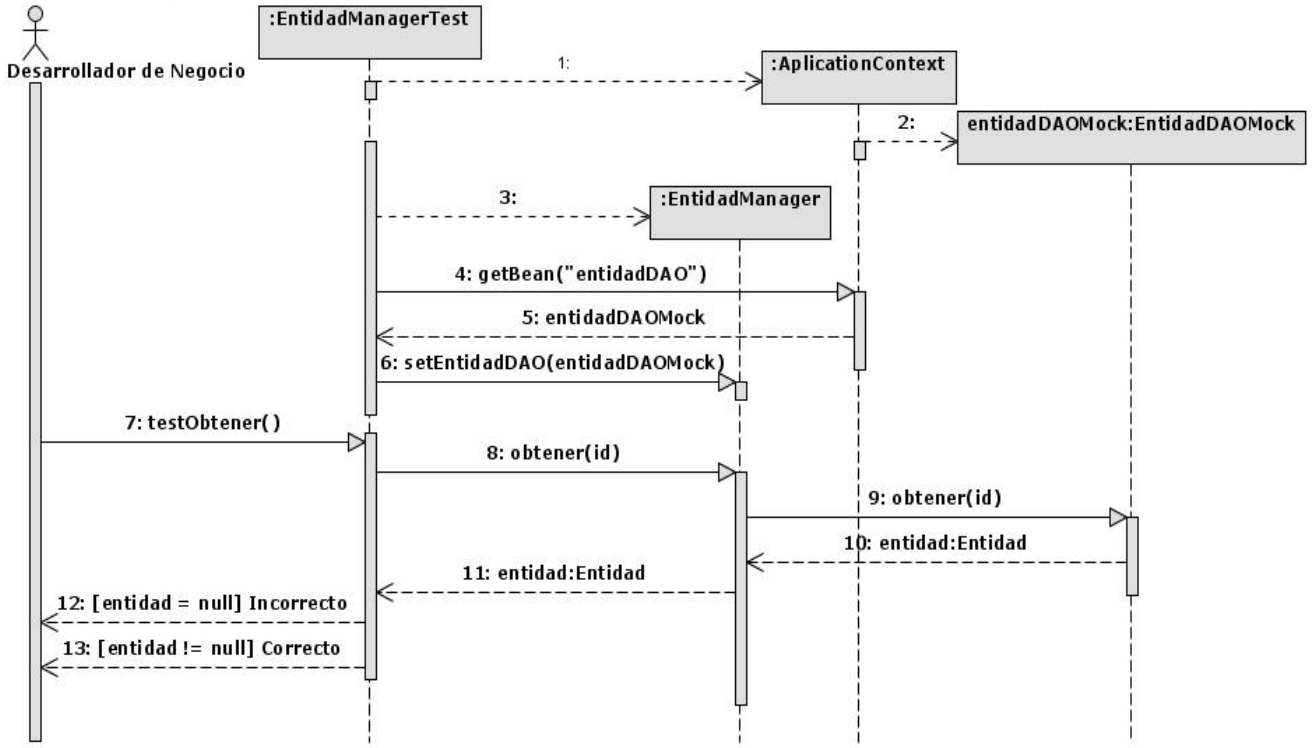


Figura 29 Pruebas de unidad a un objeto Manager con la utilización de un DAO falso.

Conclusiones

A lo largo del presente trabajo se abordaron temas de vital importancia par el desarrollado de la arquitectura base del Sistema de Gestión Policial, como son la seguridad, el tratamiento de excepciones, la auditoría y las formas de comunicación entre módulos y sistemas externos.

Además se seleccionaron las tecnologías y herramientas para el desarrollo del sistema, se aplicaron patrones arquitectónicos en la definición de la estructura física y lógica del sistema, se elaboró el diagrama de despliegue de la aplicación, se implementaron los mecanismo de seguridad y auditoría, se definieron los mecanismos de comunicación entre los componentes del sistema y con otros sistemas mediantes los servicios web, se definió una única forma de escribir el código para lograr un mayor entendimiento entre los desarrolladores del sistema. Se definió un flujo de trabajo para los distintos roles que tienen los desarrolladores del sistema para lograr una relación óptima entre estos y poder desarrollar el sistema de forma paralela en todas las capas.

La arquitectura propuesta fue aplicada en el desarrollo de SIGEPOL contribuyendo de manera significativa a que el mismo cumpliera con las normas y estándares de calidad establecidos por Calisoft[47], luego de realizadas todas las pruebas al sistema y quedando aprobado por los clientes.

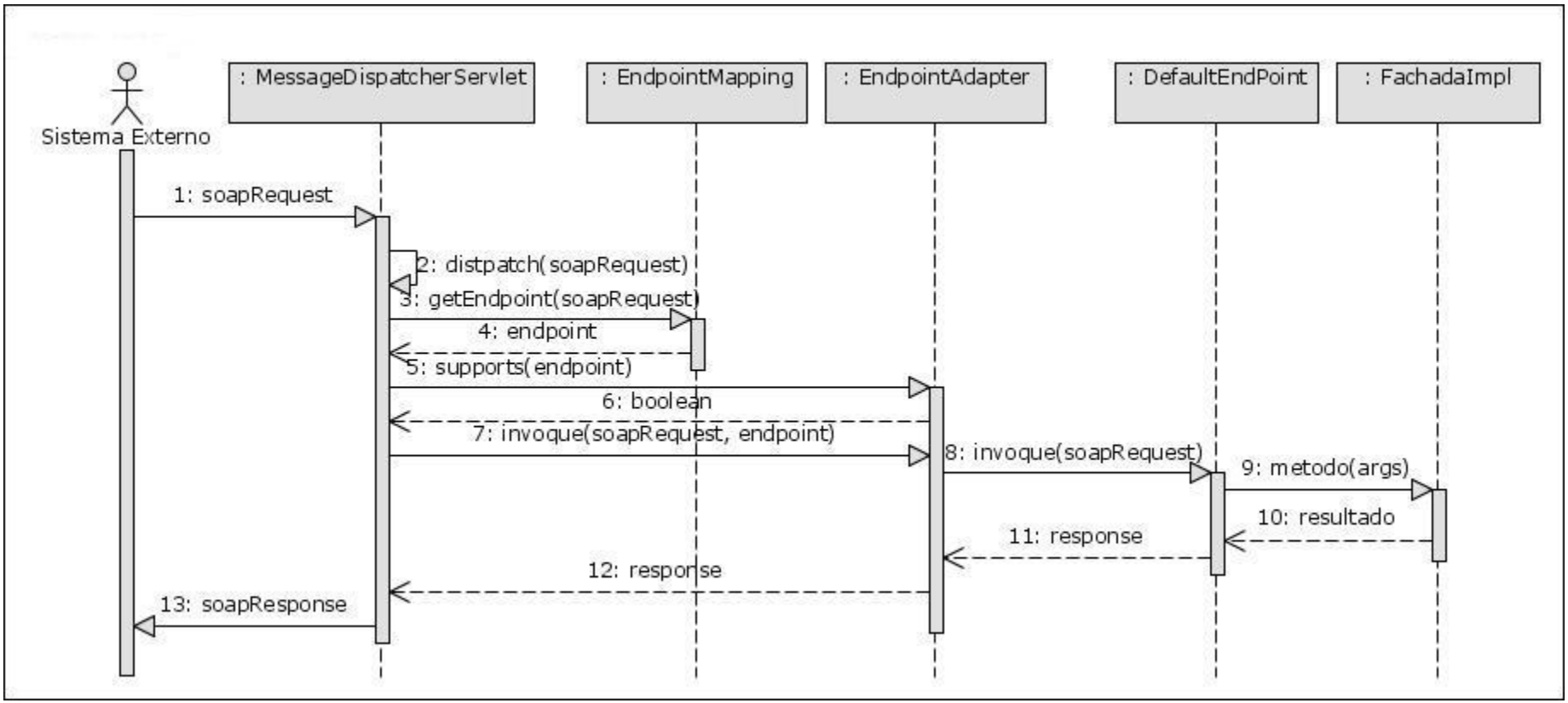
Los resultados obtenidos en la aplicación de la arquitectura propuesta en este documento en el desarrollo de SIGEPOL pueden ser aplicados a otros sistemas de características similares.

[47] Centro nacional de calidad de software.

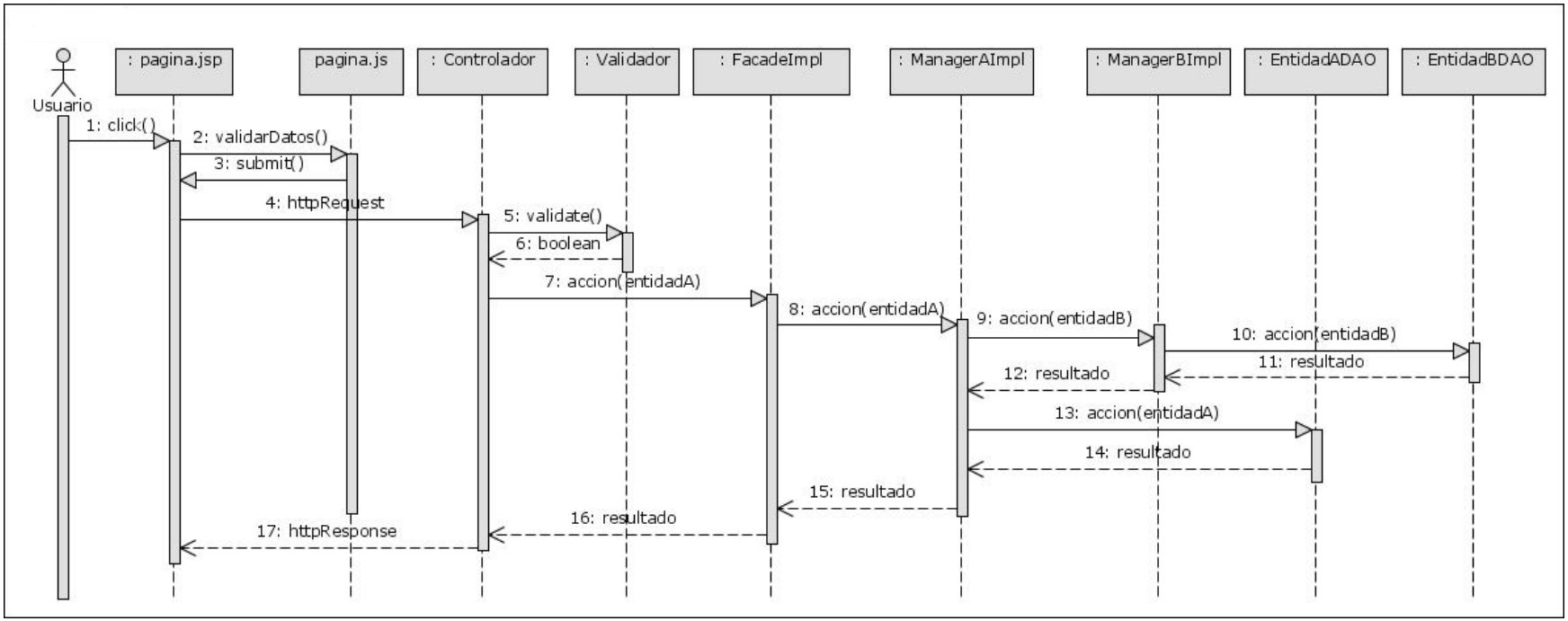
Recomendaciones

- ✓ Incluir en la solución el montaje de una Red Privada Virtual (VPN) que conecte las dependencias con el sistema.
- ✓ Investigar la posibilidad de utilizar algún software de balanceo de carga para los servidores de aplicaciones
- ✓ Investigar la posibilidad de incluir servicios web como mecanismos de comunicación entre los módulos del sistema.

Anexos



Anexo 1 Procesamiento de una petición en la interfaz de servicio.



Anexo 2 Procesamiento de una petición por las diversas capas de un módulo.

Bibliografía

1. **Chávez Frías, Hugo Rafael.** DECRETO CON FUERZA DE LEY DE COORDINACION DE SEGURIDAD CIUDADANA. [En línea] 2001. http://www.mpprij.gob.ve/IMG/pdf/Ley%20de%20Coordinacion%20Ciudadana.pdf.

2. **Wikipedia.** Software architecture. [En línea] http://en.wikipedia.org/wiki/Software_architecture.

3. **Clements, Paul y Northrop, Linda.** Software Architecture: An Executive Overview. [En línea] http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.003.html.

4. **Carlos Billy, Reynoso.** Introducción a la Arquitectura de Software. [En línea] 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx.

5. *Comentario en discusión sobre teoría y práctica de la ingeniería de software en conference de NATO Science Committee.* **Sharp, P. I.** 1969.

6. **Bass, Len, Clements, Paul y Kazman, Rick.** *Software Architecture In Practice, Second Edition.* Boston : Addison-Wesley, 2003. págs. 21-24. ISBN 0-321-15495-9.

7. **Kaiser, S. H.** *Software Paradigms.* 2005.

8. **SEI. Carnegie Mellon University.** How Do You Define Software Architecture? [En línea] http://www.sei.cmu.edu/architecture/definitions.html.

9. **Garlan, David y Shaw, Mary.** An Introduction to Software Architecture. [En línea] http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf.

10. **MSDN Latinoamérica.** Arquitectura, arquitecto y business. [En línea] 2005. http://www.mslatam.com/latam/msdn/comunidad/dce2005/secure/dashboard/dashboard_view.aspx.

11. **Wikipedia.** Stakeholder. [En línea] http://en.wikipedia.org/wiki/Stakeholder.

12. **Garlan, David.** Software Architecture: a Roadmap. [En línea] http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalgarlan.pdf.

13. **Canal, Carlos.** *Arquitectura, marcos de trabajo y patrones.* 2005.

14. **Pimentel González, Luis Alberto, Pérez Rivero, Iósev y Haro Pérez, Madelín.** *ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB.* 2007.

15. **Sánchez Téllez, Eddy y Maranje Agramonte, Adrian.** *Especificación de la Arquitectura Base del Sistema de Gestión de Emergencia y Seguridad Ciudadana 171.* 2008.

16. **Krueger, Charles W.** Introduction to Software Product Lines. [En línea] http://www.softwareproductlines.com/introduction/introduction.html.

17. —. Benefits of Software Product Lines. [En línea] http://www.softwareproductlines.com/benefits/benefits.html.

18. **Sun Microsystems, Inc.** Java BluePrints: Model-View-Controller. [En línea] http://java.sun.com/blueprints/patterns/MVC-detailed.html.

19. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley, 1995.

20. **Chávez Frías, Hugo Rafael.** Decreto N° 3.390. *La Gaceta oficial.* [En línea] 23 de Diciembre de 2004. http://www.gobiernoenlinea.ve/docMgr/sharedfiles/Decreto3390.pdf.

21. **Jonhson, Rod.** Introduction to the Spring Framework. *TheServerSide.Com.* [En línea] Mayo de 2005. http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework.

22. **Walls, Craig y Breidenbach, Ryan.** *Spring in Action.* s.l. : Manning Publications Co., 2005. 1-932394-35-4.

23. **Fowler, Martin.** Inversion of Control Containers and the Dependency Injection pattern. [En línea] Enero de 2004. http://martinfowler.com/articles/injection.html.

24. *Aspectos e intercepción de métodos en .net.* **Hernández, Yamil y Katrib, Miguel.** Número 10, 2004, dotNetManía. ISSN 1698-5451.

25. **Johnson, Rod, y otros.** Aspect Oriented Programming with Spring. *The Spring Framework - Reference Documentation.* [En línea] http://static.springframework.org/spring/docs/2.5.x/reference/aop.html.

26. **Massol, Vincent y Husted, Ted.** *JUnit in Action.* 2004. ISBN 1-930110-99-5.

27. EasyMock. [En línea] http://www.easymock.org/.

28. **Oracle.** Oracle Database 10g Enterprise Edition. [En línea] Abril de 2006. http://www.oracle.com/technology/products/database/oracle10g/pdf/DS_General_Oracle_Database10gR2_EE_0605.pdf.

29. **Meier, J.D., y otros.** *Application Architecture Guide 2.0.* 2008.

30. **Sun Microsystems, Inc.** Code Conventions for the JavaTM Programming Language. [En línea] 20 de Abril de 1999. http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html.

31. **Gallardo, David, Burnette, Ed y McGovern, Robert.** *Ecipse In Action: A Guide for Web Developers.* 2003.

32. **Sun Microsystems, Inc.** Simplified Guide to the Java™ 2 Platform, Enterprise Edition. *Sun Developer Network (SDN).* [En línea] Septiembre de 1999. http://java.sun.com/j2ee/reference/whitepapers/j2ee_guide.pdf.

33. —. Java 2 Platform, Enterprise Edition (J2EE) Overview. *Sun Developer Network (SDN).* [En línea] http://java.sun.com/j2ee/overview.html.

34. **Apache Software Foundation.** iBATIS. [En línea] http://ibatis.apache.org/.

35. **JCaptcha corp.** JCaptcha. [En línea] http://jcaptcha.sourceforge.net/.

36. **Jaspersoft Corporation.** JasperForge.org. [En línea] http://jasperforge.org/.

37. **The Dojo Foundation.** The Dojo Toolkit. [En línea] http://dojotoolkit.org/.

38. **Fowler, Martin, y otros.** *Patterns of Enterprise Application Architecture.* 2002. ISBN : 0-321-12742-0 .

39. **Clements, Paul, y otros.** *Documenting Software Architectures: Views and Beyond.* 2002. ISBN : 0-201-70372-6 .

40. **Hay Fung, Kam.** Code Conventions for the JavaServer Pages Technology Version 1.x Language. *Sun Developer Network (SDN).* [En línea] http://java.sun.com/developer/technicalArticles/javaserverpages/code_convention/.

41. **The Dojo Foundation.** Dojo Style Guide. [En línea] http://www.dojotoolkit.org/developer/StyleGuide.