Facultad 1

Título: Desarrollo de servicios comunes del negocio para el Sistema de Emisión de Pasaportes Diplomáticos de la República Bolivariana de Venezuela.

Autores: Hazzel Nuñez García.

Tutores: Ing. Sandy Moreno Baró.
Ing. Erick Vega de la Cruz.

Ciudad de La Habana, junio 2011.

Declaración de autoría

Declaro ser autor del presente trabajo de diploma y reconozco al Centro de Identificación y Seguridad Digital de la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de junio del año 2011.

Hazzel Nuñez García

_____

Firma del Autor

Sandy Moreno Baro

_____

Firma del Tutor

Erick Vega de la Cruz

_____

Firma del Tutor

Opinión del tutor

Agradecimientos

A mis amistades, aquellas personas que han estado a mi lado en estos 5 años, mis amigas Ana, Yoa, Alianis, Celia pero muy en especial a Yadira, Dayanis y Nidelso que han compartido conmigo momentos de fuerte estudio, grandes diversiones y de tristezas también.

A Annies que la quiero mucho, cuida a mi hermanito.

A Carlos A. mi compañero de tesis la mayor parte del tiempo de confección de esta tesis, quien me yudo enormemente.

A mis tutores Sandy y Erick que supieron guiarme para lograr estos resultados.

A las personas del proyecto que más que un equipo de trabajo fuimos una familia.

A todas aquellas personas que de una forma u otra han contribuido para que yo me graduara.

A Juana E. y J. Ramón por ayudarme tanto.

Dedicatoria

Esta tesis está dedicada a mi Mamá y mi Papá, por ellos soy la persona que ven y gracias a ellos he podido llegar hasta aquí, por sus esfuerzos, dedicación y cariño que si hoy yo me puedo graduar de Ingeniera, ustedes se gradúan de mejores padres.

A mi hermanito Dune por creerse un poco papi y gobernarme, pero ha sabido apoyarme y estar ahí en momentos muy dificiles, al fin lo logre como tú.

A toda mi familia por su cariño.

A mis abuelos que no han podido estar aquí por una razón u otra, Juana, Amelia, Argelio y Ramón.

A mi tata y tía Maida a Sergi y Eli que han estado pendientes de mí siempre y los quiero mucho.

Resumen

El análisis de los elementos teóricos referentes a tecnologías, técnicas, herramientas y metodologías para el desarrollo de los componentes permiten conocer las herramientas a utilizar y la existencia de sistemas informáticos que brindan soluciones similares. Para una mejor comprensión del problema tratado se efectúa un estudio de la arquitectura y los procesos de negocio del Sistema de Emisión de Pasaportes Diplomáticos de la República Bolivariana de Venezuela, donde se crea una propuesta teórica de solución a partir de la cual se elabora el modelo conceptual de dominio, se especifican los rasgos funcionales y no funcionales, además de los diagramas de diseño de componentes con su respectiva descripción. La implementación se realizó de acuerdo a la planeación de los rasgos por iteraciones, para validar los servicios comunes del negocio se utilizan métodos de prueba de caja negra y caja blanca, y la herramienta *JUnit* obteniendo como resultado el éxito de las pruebas y así validando la hipótesis de la investigación.

Índice

Índice de figuras

Índice de tablas

Introducción

Desde 2004, Venezuela lleva a cabo un proceso de remodelación de la infraestructura del Servicio Autónomo de Identificación, Migración y Extranjería (SAIME) y su plataforma tecnológica, lo cual le permitirá expedir documentos de identidad digitalizados. Esto con la intención de hacer más amenos los procesos de elaboración de los Pasaportes y Acreditaciones debido a que en la actualidad realizar trámites de esta índole son engorrosos los procesos en su conjunto se hacen muy extensos y frustrantes para los clientes, y no existe una forma de validar la identidad de la persona que solicita el servicio.

Todas estas dificultades se deben a que la República Bolivariana de Venezuela presenta en estos momentos varias problemáticas en el proceso de elaboración de pasaportes y acreditaciones, entre los cuales se encuentran:

- El proceso se realiza de forma manual.
- Los mecanismos de seguridad que poseen son insuficientes.
- No se realiza la validación de los datos y de la identidad del solicitante durante la emisión de los documentos.
- No se gestionan todos los datos necesarios sobre los titulares.
- No existe una base de datos que permita la gestión eficiente de la información referente a los trámites de Pasaportes y Acreditaciones.

Por estas razones surgió la necesidad de automatizar los procesos de Solicitud de Pasaportes Diplomáticos y de Servicio, así como las Acreditaciones. De esta forma surge el proyecto Transformación y Modernización del Sistema de Emisión de Pasaportes Diplomáticos, de Servicios y Acreditaciones de la República Bolivariana de Venezuela.

El proyecto se encuentra dividido en dos subsistemas (Pasaporte y Acreditación), estos subsistemas no cuentan con los servicios unificados, algunos se encuentran repetidos para varios procesos del negocio, esto trae consigo problemas de redundancia de código, duplicación de esfuerzos y recursos en la etapa de desarrollo, y un crecimiento innecesario del proyecto. Además, de existir una mayor cantidad de relaciones y dependencias entre las clases provocando un alto nivel de acoplamiento, haciendo el código menos flexible ante cambios y más difícil de mantener. Esta situación afecta el flujo de comunicación entre las capas de la arquitectura del sistema provocando una

degradación en el tiempo de respuesta, afectando negativamente la aceptación del usuario.

De acuerdo a la problemática planteada previamente, se precisó como **problema científico** ¿Cómo lograr el intercambio de información entre las capas de la arquitectura del Sistema de Emisión de Pasaporte Diplomáticos?

Se estableció como **objeto de estudio:** la arquitectura y los procesos del negocio del Sistema de Emisión de Pasaportes Diplomáticos de la República Bolivariana de Venezuela, y se limita al **campo de acción**: los servicios comunes del Sistema de Emisión de Pasaportes Diplomáticos de la República Bolivariana de Venezuela.

En este trabajo se parte de la **hipótesis:** con el desarrollo de los servicios comunes se logrará una mayor aceptación del usuario y mejorará el nivel de acoplamiento en el Sistema de Emisión de Pasaportes Diplomáticos.

Variable independiente: Servicios comunes.

Variables dependientes: Nivel de acoplamiento, Aceptación del usuario.

Operacionalización de las variables ver **anexo 1.**

Se tiene como **objetivo general**: desarrollar los componentes que expondrán los servicios comunes del negocio del Sistema de Emisión de Pasaporte Diplomático de la República Bolivariana de Venezuela. Del objetivo trazado se derivan los siguientes **objetivos específicos**:

- Analizar los elementos teóricos referentes a tecnologías, técnicas, herramientas y metodologías para el desarrollo de los componentes.
- Realizar un estudio de la arquitectura y los procesos de negocio del Sistema de Emisión de Pasaportes Diplomáticos de la República Bolivariana de Venezuela.
- Realizar el análisis y diseño para el desarrollo de los servicios comunes del negocio.
- Implementar los componentes que brindarán los servicios comunes del negocio.
- Validar los servicios comunes del negocio.

Las **tareas científicas** propuestas a desarrollar para dar cumplimiento a los objetivos son las siguientes:

- Elaboración del diseño teórico-metodológico de la investigación científica.
- Realización de un estudio de los aspectos teóricos conceptuales de las

tendencias, tecnologías y metodologías a utilizar.

- Identificación y caracterización de las herramientas a utilizar, así como plataforma que la soportan.
- Análisis de la arquitectura definida para el Sistema
- Definición de patrones y el estilo arquitectónico a utilizar en el desarrollo de los servicios comunes.
- Análisis del modelo de datos diseñado para la persistencia de los datos.
- Realización de un análisis de los requerimientos funcionales para determinar de los servicios comunes.
- Agrupación en los componentes de los servicios comunes que conllevan a un mismo fin.
- Definición de los componentes que serán imprescindibles para la implementación de los servicios comunes.
- Desarrollo de los componentes imprescindibles para implementación de los servicios comunes.
- Especificación y ejecución de las pruebas de aceptación a los servicios comunes.

Consecuentemente para desempeñar las tareas propuestas se utilizaron varios **métodos investigativos** como el **método teórico** que se utiliza para la construcción de las teorías científicas, para la elaboración de las premisas metodológicas de la investigación y también en la construcción de las hipótesis científicas, dentro de los que se encuentran:

El **histórico lógico** para determinar trayectoria, evolución y comportamiento de la República Bolivariana de Venezuela en la emisión y acreditación de pasaportes.

El **análisis y síntesis** para definir el negocio y la arquitectura del sistema.

El método de la **modelación** para representar gráficamente la arquitectura del sistema así como para modelar los contenidos necesarios para el análisis y diseño.

El método **sistémico** para la integración de las tecnologías utilizadas.

Otro método investigativo que se utilizó fue el **método empírico** el cual le permite al investigador, la recopilación de datos reales acerca del comportamiento de los hechos, fenómenos, objetos y procesos de la naturaleza y de la sociedad, dentro de este tipo de método se utiliza:

La **entrevista,** que se aplicó a los clientes, para determinar los requisitos que debía cumplir el sistema.

El método de la **experimentación** para el estudio de un objeto en el cual se crearán las condiciones o se adaptarán las existentes, para verificar la hipótesis. (16)

El presente documento se estructura en cuatro capítulos, que incluyen todo lo relacionado con el trabajo investigativo realizado, así como también el análisis y el diseño de la herramienta propuesta para la realización del mismo.

Capítulo I: ***Fundamentación Teórica***, en este se recopila un estudio del estado del arte del tema a tratar y de las tecnologías actuales. Se incluye además como aspectos de actualidad una descripción de las herramientas, lenguajes y metodologías a utilizar para la implementación del sistema.

Capítulo II: ***Propuesta de solución***, se describirán los procesos de negocio que sean comunes para el Sistema de Emisión de Pasaporte Diplomático de la República Bolivariana de Venezuela. Se describe el objeto de estudio y la propuesta de solución. Se identifica los requisitos necesarios, se exponen las principales clases y sus relaciones.

Capítulo III: S*ervicios comunes, elaboración y validación*, se identificaran las funcionalidades, se analizaran y se expondrán las clases y sus relaciones. Se describirá la implementación de los servicios comunes y las técnicas de programación usadas Definirán los patrones a utilizar. Se describe el diseño de los componentes. Se describen los casos de pruebas para asegurar la calidad y la validación de los componentes según los métodos establecidos.

Capítulo 1 Fundamentación Teórica

1.1 Introducción

Cada día aumentan las personas poseedoras de pasaportes, esta es una de las dificultades que presentan muchos países, por ello se viene manejando un nuevo término que es la gestión de pasaportes de forma digital, mejorando así la seguridad y dificultan la falsificación del tradicional pasaporte. Por el avance tecnológico y la creciente cantidad de información se ha hecho necesario crear sistemas informáticos para un mejor control y automatización de algunos procesos.

En el presente capítulo se describen las herramientas, metodologías, tecnologías, plataformas, lenguaje de programación y marcos de trabajo usados para el desarrollo de la solución. Además se brindaran conceptos teóricos referentes al desarrollo del proyecto para una mejor comprensión de la situación problemática. También se realizará un análisis acerca de sistemas informáticos que tenga una arquitectura similar a la utilizada en el Sistema de Emisión de Pasaportes Diplomáticos de la República Bolivariana de Venezuela a nivel internacional, nacional y en la universidad.

1.2 Objeto de Estudio

En las últimas décadas, el mundo entero ha sido conmovido por un masivo desarrollo científico tecnológico. Esto demuestra cuán importante ha sido y es hoy día el desarrollo tecnológico para el progreso de la humanidad. El desarrollo tecnológico ha demostrado las infinitas posibilidades que se abren al avanzar en este campo. Curiosamente el desarrollo tecnológico en la sociedad demuestra que compartir es la mejor forma de que todos se beneficien y amplíen sus conocimientos. Una de las causas protagonistas de esté desarrollo es la Arquitectura de Software, por la agilidad que proporciona en el desarrollo de los software.

1.2.1 Arquitectura del sistema

Las técnicas metodológicas desarrolladas con el fin de facilitar la programación se engloban dentro de la llamada Arquitectura de Software o Arquitectura lógica. Se refiere a un grupo de abstracciones y patrones que brindan un esquema de referencia útil para

guiarse en el desarrollo de software dentro de un sistema informático.

Así, los programadores, diseñadores, ingenieros y analistas pueden trabajar bajo una línea común que les posibilite la compatibilidad necesaria para lograr el objetivo deseado.

La arquitectura de software aporta una visión abstracta de alto nivel de la estructura del sistema, omite detalles de la implementación, brindando una perspectiva dinámica. Según el estándar IEEE Std 1471-2000 es “La organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

Estas Arquitecturas están definidas muchas veces por el tipo de tecnología a la cual se enfrenta un programador o grupo de programadores, por lo cual algunos tipos de arquitectura son más recomendables que otras para ciertas tecnologías.

En una arquitectura en capas como la especificada en el sistema, se define un conjunto de niveles o capas diferentes cada una realiza operaciones. En la capa externa, los componentes sirven a las operaciones de interfaz de usuario. Las capas intermedias proporcionan servicios de utilidad y funcionalidades del software. (25)

Unas de las ventajas propuestas por este tipo de arquitectura es la facilidad de estandarización, la dependencias se limitan entre capas, tiene una considerable contención de cambios a una o pocas capas, el desarrollo se realiza de forma paralela, las aplicaciones son más robustas debido al encapsulamiento.

Las capas están físicamente distribuidas, lo cual significa que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. (19)

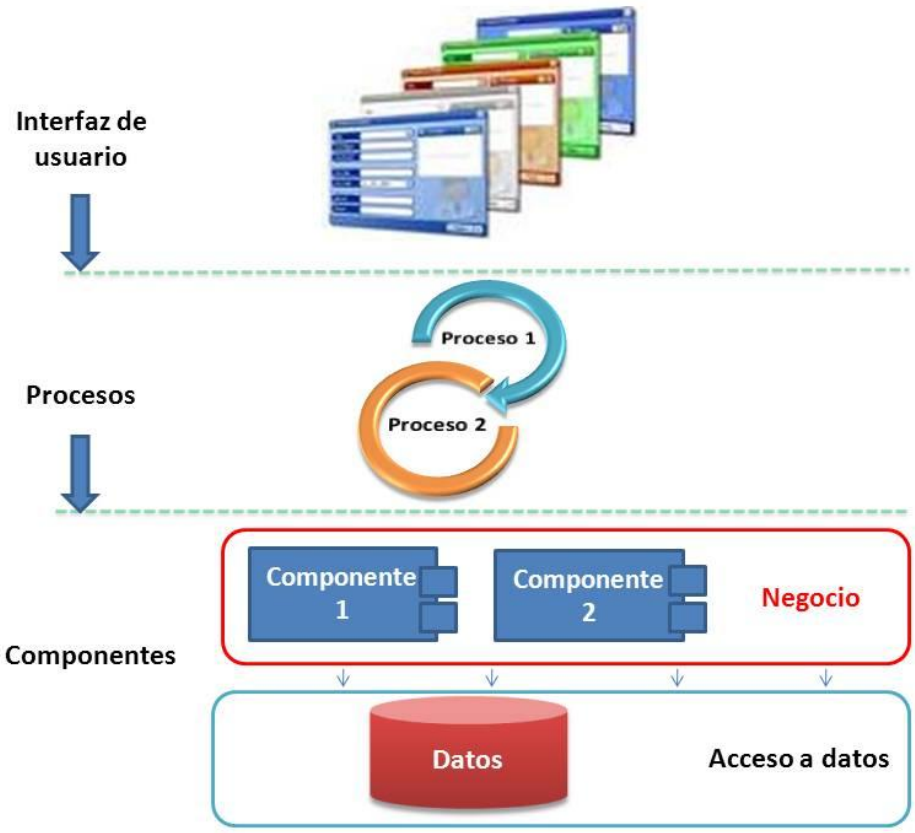

Figura 1 Arquitectura del sistema

Descripción de las capas:

La capa de **interfaz de usuario**: utiliza las funcionalidades que le brinda la capa de procesos para exponerla visualmente al usuario.

La capa de **procesos**: implementa la lógica del negocio de la organización. Esta se comunica con la capa de negocio.

La última capa de la arquitectura está fraccionada en dos capas, la subcapa de componentes donde se encuentra enmarcado el desarrollo del presente trabajo y la subcapa de accesos a datos:

La capa de **componentes**: la encargada de implementar las funcionalidades específicas del negocio a partir de los rasgos funcionales es la subcapa de **negocio** además de acceder a los datos de la capa inferior, y la subcapa de **acceso a datos** es donde residen los datos, está formada por un gestor de bases de datos que realiza todo el almacenamiento de datos.

1.2.2 Servicios

Los servicios son actividades identificables, intangibles y perecederas que son el resultado de esfuerzos humanos o mecánicos que producen un hecho, un desempeño o un esfuerzo que implican generalmente la participación del cliente y que no es posible poseer físicamente, ni transportarlos o almacenarlos, pero que pueden ser ofrecidos en renta o a la venta; por tanto, pueden ser el objeto principal de una transacción ideada para satisfacer las necesidades o deseos de los clientes. (27)

Los servicios representan grupos lógicos de operaciones relacionadas con algún concepto del negocio. Un servicio consiste en una implementación que provee lógica de negocio y datos, un contrato de servicio, las restricciones para el consumidor, y una interfaz que expone físicamente la funcionalidad. La implementación del servicio provee la lógica del negocio requerida y los datos correspondientes. Es la realización técnica que cumple con el contrato definido, en forma de componentes, programas, datos de configuración, base de datos. El contrato de servicios provee la especificación del propósito, funcionalidad, restricciones, y uso del servicio. La forma de esta especificación varía dependiendo del tipo de servicio. Las funcionalidades del servicio exponen en la interfaz del servicio del cliente conectados en la red. Existen distintos tipos de servicios con determinadas características que tienen distintos significados en SOA, un servicio funciona como una aplicación independiente, teniendo sus propias reglas de negocio, datos, procedimientos de administración y operación. Expone toda su funcionalidad utilizando un interfaz.

Un servicio de negocio es un componente reutilizable de software, con significado funcional completo, y que está compuesto por:

Contrato: especificación de la finalidad, funcionalidad, forma de uso y restricciones del servicio.

Interfaz: mecanismo de exposición del servicio a los usuarios.

Implementación: debe contener la lógica o el acceso a datos.

Tipos de servicios en cuanto a complejidad

Servicios básicos: pueden estar centrados en datos o en lógica y encapsulan

funcionalidades como cálculos complejos, acceso a datos y reglas complejas de negocio.

Servicios intermediarios: servicios adaptadores, fachadas, etcétera. Suelen ser servicios sin estado.

Servicios de proceso: servicios de negocio que encapsulan la lógica de proceso. Suelen conservar estado y pueden residir en herramientas BPM.

Servicios públicos: servicios accesibles por terceros (fuera de la organización).

1.2.3 Estilos arquitectónicos más usados

Caracteriza una familia de sistemas que están relacionados por compartir propiedades estructurales y funcionales. También puede definirse como la descripción de los tipos componente y de los patrones de interacción entre ellos. (14)

Shaw y Garlan (1996) definen estilo arquitectónico como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes. (26)

Buschmann et al. (1996) definen estilo arquitectónico como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Así mismo, se considera como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo. Éste incluye información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación. (11)

Modelo Vista Controlador (MVC)

Este patrón fue descrito por primera vez por Trygve Reenskaug en 1979, y la implementación original fue realizada en Smalltalk en los laboratorios Xerox.

MVC se basa en la separación de la aplicación en tres capas principales:

Modelo: es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio.

Vista: Se presenta el modelo en un formato adecuado para interactuar, usualmente un

elemento de interfaz de usuario.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos. MVC no menciona específicamente esta capa de acceso a datos porque supone que está encapsulada por el modelo.

El objetivo primordial del MVC es la reutilización del código ya implementado. Esta tarea se facilita mucho si a la hora de programar se tiene la precaución de separar el código en varias partes que sean susceptibles de ser reutilizadas sin modificaciones. (17)

Arquitecturas en Capas

El estilo arquitectural en capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan.

Descomposición de los servicios de forma que la mayoría de interacciones ocurre solo entre capas vecinas. Las capas de una aplicación pueden residir en la misma máquina o pueden estar distribuidos entre varios equipos. Los componentes de cada capa se comunican con los componentes de otras capas a través de interfaces bien conocidos. Cada nivel agrega las responsabilidades y abstracciones del nivel inferior. Muestra una vista completa del modelo y a la vez proporciona suficientes detalles para entender las relaciones entre capas. No realiza ninguna suposición sobre los tipos de datos, métodos, propiedades y sus implementaciones. Separa de forma clara la funcionalidad de cada capa. La comunicación entre capas está basada en una abstracción que proporciona un bajo acoplamiento entre capas. (15)

Arquitecturas Orientadas a Objetos

El estilo define el sistema como un conjunto de objetos que cooperan entre sí en lugar de como un conjunto de procedimientos. Los objetos son discretos, independientes y poco acoplados, se comunican mediante interfaces y permiten enviar y recibir mensajes.

Es un estilo para diseñar aplicaciones basadas en un número de unidades lógicas y código reusable. Describe el uso de objetos que contienen los datos y el comportamiento para trabajar con esos datos y además tienen un rol o responsabilidad distinta. Hace

hincapié en la reutilización a través de la encapsulación, la modularidad, el polimorfismo y la herencia.

Contrasta con el acercamiento procedimental donde hay una secuencia predefinida de tareas y acciones. El enfoque orientado a objetos emplea el concepto de objetos interactuando unos con otros para realizar las tareas. (15)

Arquitecturas Basadas en Componentes

El estilo de arquitectura basada en componentes describe un acercamiento al diseño de sistemas como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema. Se utiliza para diseñar aplicaciones a partir de componentes individuales.

Enfatiza la descomposición del sistema componentes con interfaces muy bien definidas. Define una aproximación al diseño a través de componentes que se comunican mediante interfaces que exponen métodos, eventos y propiedades.

Los componentes son diseñados de forma que puedan ser reutilizados en distintos escenarios en distintas aplicaciones aunque algunos componentes son diseñados para una tarea específica. Los componentes son diseñados para operar en diferentes entornos y contextos. Toda la información debe ser pasada al componente en lugar de incluirla en él o que este acceda a ella. (15)

Arquitecturas Orientadas a Servicios (SOA)

Las Arquitecturas Orientadas a Servicios están formadas por servicios de aplicación débilmente acoplados y altamente interoperables. Para comunicarse entre sí, estos servicios se basan en una definición formal independiente de la plataforma y del lenguaje de programación. La definición de la interfaz encapsula las particularidades de una implementación, lo que la hace independiente del fabricante, del lenguaje de programación o de la tecnología de desarrollo. Con esta arquitectura, se pretende que los componentes software desarrollados sean muy reusables, ya que la interfaz se define siguiendo un estándar; así, un servicio desarrollado en CSharp podría ser usado por una aplicación Java.

SOA es un estilo de arquitectura en el cual se exponen los procesos de negocio del sistema a construir como servicios independientes de alta cohesión y bajo acoplamiento que encapsulan dichos procesos y pueden ser invocados a través de interfaces bien definidas.

La Arquitectura Orientada a Servicios es un concepto de arquitectura de software que define la utilización de servicios como construcciones básicas para el desarrollo de aplicaciones. Es una arquitectura de una aplicación donde las funcionalidades se definen como servicios independientes, con interfaces accesibles, bien definidas, que pueden ser llamadas en secuencias dadas para formar procesos de negocios.

La Arquitectura Orientada a Servicios (SOA), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requerimientos de software del usuario. SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.

SOA está resultando un paradigma para la integración de sistemas empresariales, que no sólo integra diferentes tipos de software sino que además es independiente de la plataforma, el sistema operativo y el dispositivo. (20)

1.2.4 Tendencias tecnológicas

La competitividad es el imaginario que dirige las acciones empresariales en la actualidad. Lograr condiciones que permitan competir con mayores oportunidades, exige de las empresas desarrollar ventajas competitivas en su forma de operar. La fuente de estas ventajas esta en las actividades que desarrolla, por lo que la eficiencia en los procesos de negocio representa un foco de acción para sus directivos. Para apoyar este objetivo, las herramientas y metodologías para gestión de procesos han venido evolucionando con el paso del tiempo. En los últimos años, la necesidad de alinear la estrategia a la operación de negocio y el desarrollo de la tecnología de información, han generado nuevas formas de gestionar los procesos en las organizaciones. Workflow, Gestión de Procesos del Negocio (BPM), Servicios Web, Monitoreo de las Actividades del Negocio son algunas de las tecnologías que se perfilan como una nueva tendencia para aumentar la eficiencia del negocio y generar las ventajas competitivas que exige el mercado.

Workflow

En el mundo de los negocios, flujo de trabajo es como se mueve un elemento de una persona a otra a través de un proceso. Ese proceso es el proceso de negocio, y define los pasos necesarios para completar una pieza de trabajo. Pasos en el proceso puede ser

obligatorio u opcional.

Antes de definir lo que es *workflow* se debe de tener una definición clara de qué es un Proceso de Negocio:

"Un **proceso** es un orden específico de actividades de trabajo, que se realizan en el tiempo, en lugares específicos y por personas o sistemas, con un principio, un fin, y entradas y salidas claramente definidas. Es decir, una estructura cohesionada y coordinada adecuadamente para la acción."

Ahora bien, se puede definir el *workflow* como:

"La automatización de los procesos de negocio durante el cual "documentos", "información" y "tareas" son pasados de un participante a otro, incluso el cliente, acorde a un conjunto de reglas procedimentales."

Un sistema para la gestión del trabajo provee beneficios tanto a trabajadores como a la organización. Las tareas de los trabajadores se realizan más fácilmente y la organización conoce y controla las tareas que se llevan a cabo. Uno de los beneficios más importantes es que el *workflow* permite a las empresas optimizar sus inversiones existentes y del pasado en TI, implementando una arquitectura abierta basada en los estándares de la industria, simplificando su integración con cualquier sistema de "back-office", Middleware o ERP, y en cualquier plataforma o sistema operativo.

El objetivo de un sistema de *workflow* es, a través de un motor, gestionar de forma automatizada los procesos y flujo de actividades, documentos, imágenes y datos, orquestando e integrando los recursos informáticos y los roles. (10)

Los beneficios, tanto tangibles como intangibles, son numerosos. A continuación describo los más importantes:

- Mejora la atención y servicio al cliente.
- Incrementa el número de actividades ejecutadas en paralelo.
- Minimiza el tiempo requerido por los participantes para acceder a la documentación, aplicaciones y bases de datos.
- Disminuye "drásticamente" el tiempo de transferencia de trabajo, información y documentos entre actividades.
- Asegura la continua participación y colaboración de todo el personal en el proceso.
- Disminuye "drásticamente" el tiempo que los participantes, supervisores y administradores necesitan para conocer la situación de un ítem de trabajo (P.ej.: Orden de compra, participación de siniestro, pedido de cliente).

- Simplificación de salidas - “outputs” – automáticas.
- Disponibilidad de mecanismos para una mejor gestión y optimización de procesos.

Gestión de Procesos del Negocio (BPM)

BPM constituye una de las tendencias en gestión, que permite de manera deliberada y colaborativa manejar sistemáticamente todos los procesos de negocio de una empresa. Los beneficios de BPM para las organizaciones son extensos. Aporta visibilidad a los directivos sobre la dinámica de los procesos llevados de manera inconsciente por parte del equipo humano de las organizaciones y posibilita su modificación rápida a través de herramientas tecnológicas para acelerar la adopción del cambio en la forma de operar de las compañías. BPM se soporta sobre tecnología de información para automatizar tareas y dar agilidad a los cambios requeridos por la empresa. A diferencia de los sistemas de información tradicionales basados en la gestión de datos, estos sistemas se especializan en la gestión de procesos de negocio.

Se puede entender BPM como el mejoramiento de la gestión de los procesos de negocio de una firma de principio a fin, a partir de la definición deliberada, colaborativa e incremental de la tecnología; para alcanzar claridad en la dirección estratégica, alineación de los recursos de la empresa y disciplina de mejoramiento continuo, necesarias para cumplir las expectativas de los clientes. Los siguientes son otros beneficios identificados:

- Visibilidad de los procesos de las empresas.
- Mayor flexibilidad y agilidad para adaptación al cambio.
- Posibilidad de integrar la información del negocio dispersa en diferentes sistemas.
- Dirigir los esfuerzos de la empresa de una manera planeada y alineada con los objetivos estratégicos.
- Adquirir la habilidad para diseñar, simular y monitorear procesos de manera automática y sin la participación de usuarios técnicos.
- Adquirir una ruta de mejoramiento y eficiencia continua al convertir actividades ineficientes en menores costos a través de uso de tecnología enfocada en procesos.
- Reducir costos futuros de integración y mantenimiento al adquirir tecnología ya preparada para abordar el cambio.

Servicios Web

Un servicio web es cualquier servicio que está disponible a través de Internet, utiliza un estándar XML sistema de mensajería, y no está ligado a ningún sistema operativo o un lenguaje de programación.

Un servicio web debe ser auto-descripción. Si publica un nuevo servicio web, que también debe publicar una interfaz pública para el servicio. Como mínimo, el servicio de debe incluir la documentación legible para que otros desarrolladores pueden obtener más integrar fácilmente su servicio. Si ha creado un servicio SOAP, también debe idealmente incluir una interfaz pública por escrito en una común gramática XML. La gramática XML puede ser utilizada para identificar todos los métodos públicos, los argumentos de método, y el retorno valores.

Un consumidor de un servicio web no está vinculada a que el servicio web directamente, el servicio web interfaz puede cambiar con el tiempo sin comprometer la capacidad del cliente para interactuar con el servicio. Un sistema fuertemente acoplado implica que el cliente y el servidor de la lógica son estrechamente vinculados entre sí, lo que implica que si uno cambia la interfaz, el otro también debe se actualizará. La adopción de una arquitectura de acoplamiento flexible tiende a hacer que los sistemas de software más manejable y sencillo permite la integración entre diferentes sistemas. (12)

En los últimos dos años, tres tecnologías primarias se han convertido en estándares en todo el mundo que constituyen el núcleo de la tecnología de web de hoy en día los servicios. Estas tecnologías son:

1. Simple Object Access Protocol (Simple objeto de protocolo de acceso) SOAP: SOAP proporciona una estructura de empaquetado estándar para el transporte de documentos XML a través de una variedad de tecnologías estándar de Internet, incluyendo SMTP, HTTP y FTP. También define la codificación y normas vinculantes para la codificación no XML invocaciones RPC en XML para el transporte. SOAP proporciona una estructura sencilla para hacer RPC: documento de cambio. Al tener un mecanismo de transporte estándar, los clientes heterogéneos y servidores de repente puede llegar a ser interoperables.
2. Lenguaje de descripción de servicio web (WSDL): WSDL es una tecnología XML que describe la interfaz de un servicio web en una manera estandarizada. WSDL estandariza la forma de un servicio web representa la entrada y la salida de

parámetros de una invocación, la estructura de la función, la naturaleza de la invocación (en tan sólo, de entrada / salida, etc), y el protocolo del servicio obligatorio. WSDL permite que los diferentes clientes entiendan la forma interactuar con un servicio web.

3. Universal descripción, descubrimiento, e integración (UDDI): UDDI proporciona un registro mundial de servicios web para la publicidad, el descubrimiento y propósitos de integración. Los analistas de negocio y técnicos utilizan UDDI para descubrir servicios web, mediante la búsqueda de nombres, identificadores, categorías, especificaciones aplicadas por el servicio web. UDDI proporciona una estructura representativa para las empresas, las relaciones de negocio, servicios web y puntos de acceso a servicios web.

Individualmente, cada una de estas tecnologías es sólo de la evolución. Cada uno proporciona una norma para el siguiente paso en el avance de los servicios web, su descripción o su descubrimiento. Sin embargo, una de las grandes promesas de los servicios web es la integración sin fisuras: una pieza de software que descubrir, acceder, integrar, e invocar los nuevos servicios de desconocidos por empresas de forma dinámica. (13)

1.3 Tecnología de Persistencia (O/RM)

Las herramientas O/RM pueden permitir diseñar un modelo entidad relación y generar a partir de ello un esquema de base de datos real al mismo tiempo que se establece el mapeo entre objetos/entidades del dominio y la base de datos.

Si la base de datos ya existe, normalmente también las herramientas O/RM permiten generar el modelo de datos entidad-relación a partir de dicha base de datos existente y entonces mapear los objetos/entidades del dominio y la base de datos.

ADO.NET Entity Framework

Entity Framework permite a los programadores crear aplicaciones de acceso a datos programando con un modelo de la aplicación conceptual en lugar de programar directamente con un esquema de almacenamiento relacional. El objetivo es reducir la cantidad de código y mantenimiento que se necesita para las aplicaciones orientadas a datos. (3)

Las aplicaciones de *Entity Framework* ofrecen las siguientes ventajas:

- Las aplicaciones pueden funcionar en términos de un modelo conceptual más centrado en la aplicación, que incluye tipos con herencia, miembros complejos y relaciones.
- Las aplicaciones están libres de dependencias de codificación rígida de un motor de datos o de un esquema de almacenamiento.
- Las asignaciones entre el modelo conceptual y el esquema específico de almacenamiento pueden cambiar sin tener que cambiar el código de la aplicación.
- Los programadores pueden trabajar con un modelo de objeto de aplicación coherente que se puede asignar a diversos esquemas de almacenamiento, posiblemente implementados en sistemas de administración de base de datos diferentes.
- Se pueden asignar varios modelos conceptuales a un único esquema de almacenamiento.

TierDeveloper

TierDeveloper es una herramienta de mapeo objeto relacional que le ayuda a desarrollar complejos problemas de la vida real. TierDeveloper para ello, lo que le permite asignar objetos a bases de datos relacionales, SQL poner el poder y las reglas de negocio en ellos, y generar completamente el código de trabajo al instante. TierDeveloper también genera una aplicación ASP.NET para que sea probar los objetos generados o para utilizar como una aplicación de administración de gran alcance.

Algunos de los principales beneficios de utilizar TierDeveloper son:

Acelerar: .NET Desarrollo del 50% de su código se genera al instante.

Mejorar la calidad del código: Todo el código generado es consistente y se basa en un patrón de diseño.

Reducir la prueba de esfuerzo: Todo el código generado se basa en la pre-prueba plantillas.

TierDeveloper no sólo le permite generar su negocio y los objetos de datos, sino que también genera una aplicación ASP.NET de gran alcance. Aunque, el propósito principal de esta aplicación es para hacerle la prueba todo el código generado, también es una gran utilidad como una aplicación de administración en muchas situaciones. Esto le ahorra una gran cantidad de esfuerzo por el desarrollo de aplicaciones de administración

con la mano.

Marco de trabajo Hibernate 3

Hibernate es una herramienta de mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (Lenguaje de Marcas Extensible (XML)), que permiten establecer estas relaciones además de diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones y un gran número de tipos de datos. De una manera muy rápida y optimizada se podrán generar bases de datos en cualquiera de los entornos soportados: *Oracle*, DB2, *MySQL*, entre otros.

Hibernate presenta grandes ventajas en cuanto a sus prestaciones y flexibilidad al realizar la correspondencia entre tablas de la base de datos y las clases que representan estas tablas. Soporta diversos tipos de asociaciones, se pueden establecer asociaciones de uno a muchos, de uno a uno, de muchos a muchos y hasta asociaciones de herencia. Uno de los atractivos al utilizar *hibernate* es que cuenta con herramientas que permiten generar automáticamente los archivos de mapeo y las clases persistentes a partir de un esquema de base de datos existente, eliminando el proceso de crearlos manualmente por parte del programador. Este marco de trabajo presenta su propio Lenguaje de Consulta de *Hibernate* (HQL). Las consultas construidas con este lenguaje se realizan en base a las clases y sus atributos, con el uso de este lenguaje se establecen gran portabilidad hacia los distintos Sistema de Gestión de Base de Datos Relacional (SGBDR), ya que a partir de este lenguaje el propio *Hibernate* genera el código Lenguaje de Consulta Estructurado (SQL) nativo del SGBDR que se utilice.

1.4 Sistemas informáticos semejantes

El estudio y análisis de las tendencias y tecnologías más modernas de los productos de software existentes en el mercado que dan solución a problemas similares al tema, permitieron conocer, que tanto a nivel internacional como nacional, existen varias aplicaciones que proponen soluciones con la utilización de módulos comunes.

De esta manera se encuentran varios sistemas de los que se dan algunas características.

Sistema informático para la gestión de la información del capital humano en las empresas[1].

Con este se desea lograr diseñar un sistema para la gestión básica de información del personal y el pago a los trabajadores; el cual contiene un módulo de gestión de Capital Humano partiendo de que la única ventaja competitiva de las organizaciones. Y como objetivo final se espera realizar la gestión integral del capital humano basada en las competencias laborales. (23)

El sistema está formado por varios componentes los cuales pudieran ser utilizados en esta y otras soluciones similares. El mismo está estructurado en 5 componentes, los cuales se relacionan entre sí además de con otros subsistemas pertenecientes al Sistema Integral de Entidades CEDRUX[2]. Como se muestra en la figura

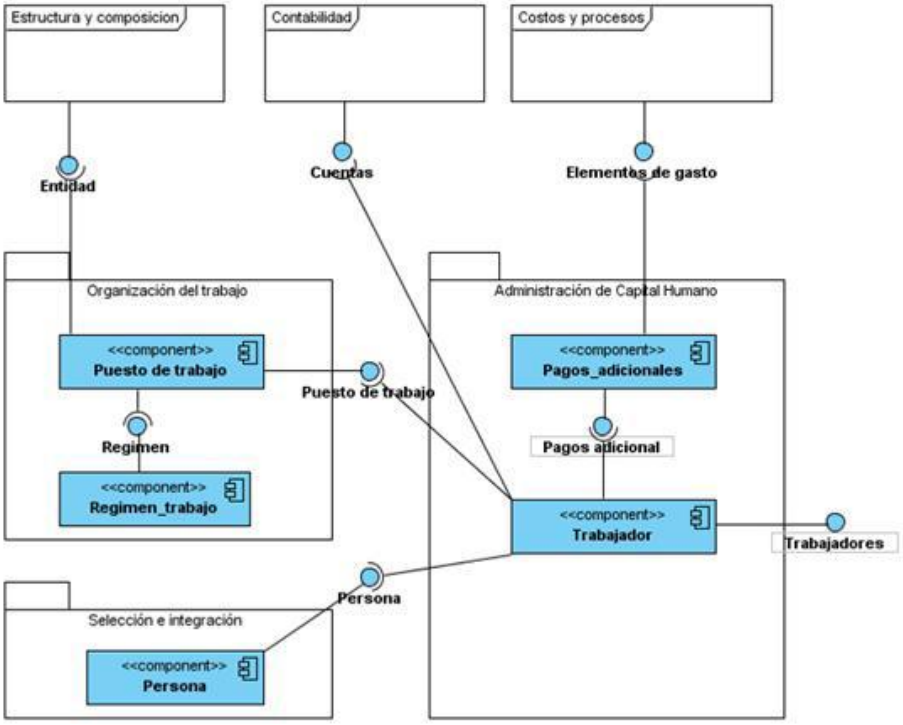

Figura 2 Estructura de los componentes

[1]Autores Arnolis Salgueiro Arzuaga, Fidel Jiménez Sanzano, José Antonio Linares, Universidad de las Ciencias Informáticas, Cuba

[2] Primer sistema integral de gestión cubano que sustituya la amplia gama de aplicaciones informáticas usadas para actividad empresarial en el país(ERP Cubano).

Solución base para el desarrollo de sistemas de control logístico de CEDRUX.

La Universidad de las Ciencias Informáticas (UCI) junto a otras organizaciones del país, ha venido trabajando en el desarrollo del Sistema Integral de Gestión Empresarial CEDRUX. Como parte de esta solución aparecen varios sistemas integrados: Configuración, Contabilidad, Planificación, Recursos Humanos y Logística. Esta última encargada del control de los medios materiales (Inventario), de los activos fijos tangibles y la facturación.

Estos procesos conforman subsistemas. Con el objetivo de lograr que estos subsistemas pudiesen funcionar aislados pero manteniendo todas las funcionalidades se crearon componentes comunes para ellos. La propuesta de solución está basada en la realización de varios componentes verticales, agrupados en los paquetes Común, Útiles y Estructura

Paquete Común:

Se encuentran componentes que son utilizados por los subsistemas de Logística con la particularidad de que no presentan interfaz gráfica. Común: Realiza varias verificaciones que son necesarias para los subsistemas de Inventario, Facturación, Activos fijos tangibles y Servicio.

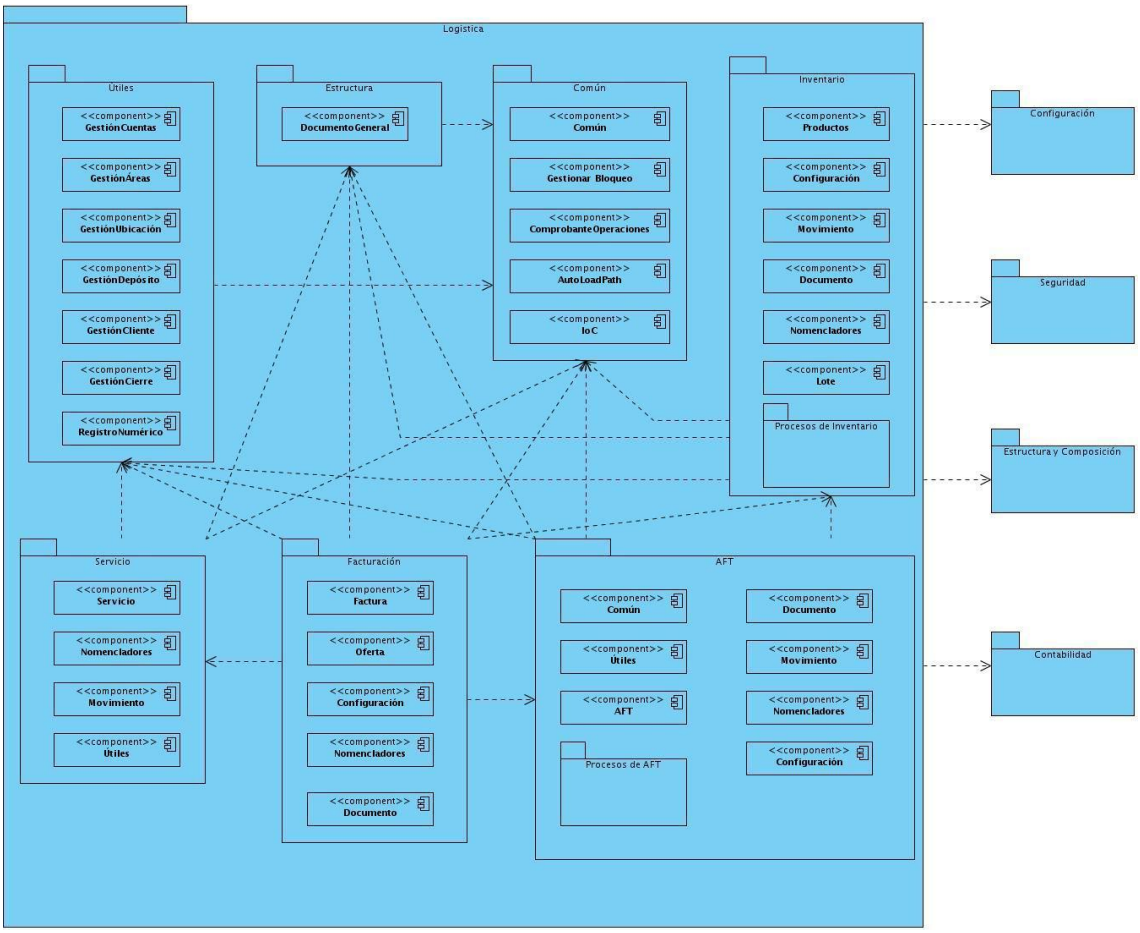

Figura 3 Relaciones de Uso de los componentes

Documento de arquitectura de software (SAD)[3]

SAD del Subsistema de Reservas del Sistema de Gestión Hotelera. Este sistema está conformado por varios subsistemas; entre ellos se encuentra el Subsistema de Reservas del cual se presenta una breve descripción de la arquitectura.

La vista de implementación presenta los componentes de implementación construidos para el Subsistema de Reservas. Bajo la tecnología .Net, la unidad de implementación son los ensamblados.

Estructura de la aplicación

Dentro del ensamblado de cada capa se encuentran los módulos que residen en ella desde el punto de vista lógico. Se cuenta con ensamblado adicional que contiene la

[3] Caso de estudio por el grupo COAL, autor Daniel Perovich, Universidad de la República de Montevideo

biblioteca de clases utilizadas para compartir información entre las demás capas, *Data Type.dll.* Se representa a continuación un diagrama de componentes que indica las dependencias entre los ensamblados.

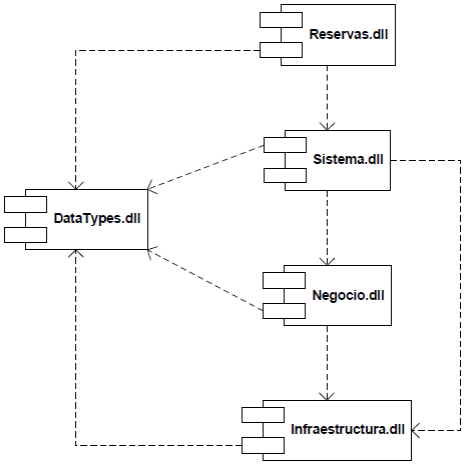

Figura 4 Arquitectura de implementación

1.5 Metodologías de desarrollo de software

Hace años que se trata de establecer una metodología estándar para el desarrollo del software. En la actualidad no se ha podido alcanzar la mencionada metodología debido a la existencia de diversos criterios y concepciones que tienen los productores de software.

Al inicio el desarrollo de software era artesanal en su totalidad, la fuerte necesidad de mejorar el proceso y llevar los proyectos a la meta deseada, tuvieron que importarse la concepción y fundamentos de metodologías existentes en otras áreas y adaptarlas al desarrollo de software. Esta nueva etapa de adaptación contenía el desarrollo dividido en etapas de manera secuencial que de algo mejoraba la necesidad latente en el campo del software.

Hay una serie de metodologías tradicionales propuestas casi todas ellas con anterioridad a los años 90 que pretendían ayudar a los profesionales indicando pautas para realizar y documentar cada una de las tareas del desarrollo del software. Sólo algunas metodologías tradicionales han sido revisadas y adaptadas, su funcionalidad suele estar limitada a proyectos no muy innovadores.

Entre las principales metodologías tradicionales se tienen los ya tan conocidos RUP(Proceso Racional Unificado) y MSF (Marco de Solución de Microsoft), que centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con

un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto. (21)

1.5.1 Metodologías de desarrollo de software ágiles

Las metodologías ágiles surgen como una extensión a las metodologías tradicionales para mejorar el desarrollo de sistemas, según el tipo de proyecto y empresa, añadiendo y mejorando (optimizando) las prácticas de desarrollo de software.

Se entiende como desarrollo ágil de software a un paradigma de desarrollo de software basado en procesos ágiles. Los procesos ágiles de desarrollo de software, conocidos anteriormente como metodologías livianas, intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados. Existen muchos métodos de desarrollo ágil; la mayoría minimiza riesgos desarrollando software en cortos lapsos de tiempo. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. (7)

Algunas metodologías ágiles de desarrollo de software:

- Desarrollo de software adaptable (ASD).
- Proceso unificado ágil (AUP).
- *Crystal Clear.*
- Proceso unificado esenciales (EssUP).
- Proceso unificado abierto (OpenUP).
- Programación Extrema (XP).
- *Scrum*.
- Desarrollo dirigido por funcionalidades (FDD)

A continuación se hará una descripción de la metodología utilizada para documentar:

FDD (Desarrollo dirigido por funcionalidades)

FDD es un enfoque ágil para el desarrollo de sistemas. Fue diseñado por Peter Coad, Eric Lefebvre y Jeff DeLuca. Dicho enfoque no hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción. Sin embargo, fue diseñado para trabajar con otras actividades de desarrollo de software y no requiere la utilización de ningún modelo de proceso específico. Además, hace énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del

proyecto. Al contrario de otras metodologías, FDD afirma ser conveniente para el desarrollo de sistemas críticos.

Se podría considerar a medio camino entre RUP y XP. Se basa en un proceso iterativo con iteraciones cortas (dos semanas) que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar. Las iteraciones se deciden en base a funcionalidades (pequeñas partes del software con importancia para el cliente). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software.

Un proyecto que sigue FDD se divide en 5 fases:

1. Desarrollo de un modelo general: los expertos del dominio presentan un documento inicial de alto nivel sobre el alcance del sistema y su contexto. A continuación los expertos del dominio y los desarrolladores construyen el esqueleto de un primer modelo del sistema bajo la tutela del jefe de arquitectos. Luego, el dominio es dividido en distintas áreas y a cada subgrupo se le asigna un área de dominio a desarrollar. Una vez finalizada cada área, el grupo se reúne para realizar un modelo global en base a todas las alternativas.
2. Construcción de la lista de funcionalidades: el equipo identifica las funcionalidades que el cliente quiere, las agrupa, las prioriza y las pondera. En iteraciones subsecuentes del proceso, el equipo se divide en subgrupos que se especializan en áreas relacionadas a las funcionalidades identificadas.
3. Plan de funciones: en base a la lista de funciones de la etapa anterior, el director del proyecto, director de desarrollo y el jefe de los programadores establecen hitos y diseñan un cronograma de diseño y construcción.
4. Diseñar en base a las funcionalidades: el jefe de los programadores toma la próxima funcionalidad a ser diseñada, identifica las clases involucradas y contacta al Class Owner correspondiente. Luego el equipo, trabaja en la realización del diagrama de secuencia correspondiente. El Class Owner hace una descripción de la clase y sus métodos
5. Implementar en base a las funcionalidades: en esta etapa cada Class Owner construye los métodos de clase para cada funcionalidad correspondiente y luego realiza la prueba unitaria para cada una de las clases. A su vez, se realiza una inspección del código y luego que el código fue implementado e inspeccionado,

el Class Owner realiza un chequeo al repositorio. Luego se realiza una compilación en el cual se hace la integración con la funcionalidad antes realizada. También se realizan las pruebas de integración correspondiente.

Se usan las sesiones de trabajo conjuntas en fase de diseño para conseguir una arquitectura sencilla y sin errores y las revisiones de código guiadas por algún programador con más experiencia.

El FDD clasifica sus roles en las siguientes tres categorías:

1. Key roles
   - Director del proyecto
   - Jefe de los arquitectos
   - Director de desarrollo
   - Jefe de los programadores
   - Class owner (propetario de clase)
   - Expertos del dominio
2. Supporting roles(roles de soporte)
   - Release manager(Jefe de lanzamiento)
   - Language lawyer(Idioma abogado)
   - Build engineer(ingeniero constructor)
   - Toolsmith(Herranienta de herrero)
   - Sistem administrator(Administrador de sistema)
3. Additional roles(Roles adicionales)
   - Probadores
   - Desarrolladores
   - Techical writers(Escritores técnicos)

Vale la pena aclarar que un miembro de un equipo puede ejercer varios roles y un rol pude ser compartido por varias personas.

Ventajas

- Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado.
- Propone tener etapas de cierre cada dos semanas.
- Se obtienen resultados periódicos y tangibles.

- Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorear.
- Define claramente entregas tangibles y formas de evaluación del progreso del proyecto.

Desventajas

- No hace énfasis en la obtención de los requerimientos sino en cómo se realizan las fases de diseño y construcción.
- Necesita tener en el equipo miembros con experiencia que marquen el camino a seguir desde el principio, con la elaboración del modelo global.

1.6 Lenguajes de programación y modelado

Java

Java es un lenguaje de programación y la primera plataforma informática creada por Sun Microsystems en 1995. Lenguaje orientado a objetos, basado en la sintaxis de los lenguajes de programación C y C++, pero presenta un modelo de objetos mucho más simple que estos. Debido a su característica de ser orientado a objeto soporta los tres pilares propios del paradigma de la programación orientación a objetos (POO), que son: el encapsulamiento, herencia y polimorfismo.

Java fue diseñado para crear software rápido, seguro y fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

Una de las principales características por las que *Java* se ha hecho muy famoso es que es un lenguaje independiente de la plataforma, presentando las siguientes ventajas:

- Reduce en un 50% los errores más comunes de programación.
- Reducción del tiempo de desarrollo de aplicaciones.
- Las aplicaciones desarrolladas sobre *Java* resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o del sistema, además es interpretado y dinámico.

Lenguaje Unificado de Modelado (UML)

Para dar soporte a dicha metodología de software se usó UML. Se trata de un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML es una consolidación de muchas de las notaciones y conceptos más usados orientados a objetos. Se ha convertido en un estándar con las siguientes características: (8)

- UML permite la creación de los diferentes modelos que ofrecen las vistas necesarias para la construcción de un software de calidad.
- Permite la compresión del sistema que se quiere realizar tanto por parte de los usuarios finales, como de los desarrolladores que implementarán la solución.
- Se puede usar para modelar distintos tipos de sistemas, ya sean sistemas de software o sistemas de hardware, y organizaciones del mundo real.
- Utilizando UML se pueden modelar sistemas bajo el paradigma orientado a objetos.
- Permite especificar las decisiones de análisis y diseño, construyéndose modelos precisos y completos.
- Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas, además cuenta con reglas para combinar dichos elementos.
- Es independiente del lenguaje de programación y de las características de los proyectos, ya que fue diseñado para modelar cualquier tipo de proyecto.
- Integra las mejores prácticas de los lenguajes de modelación existentes.
- A pesar de ser un lenguaje potente, es fácil de aprender y de usar.
- Permite documentar los artefactos de un proceso de desarrollo.
- Posee la capacidad de modelar toda la gama de sistemas que se necesite construir.

1.7 Herramienta para el desarrollo del software.

Marco de trabajo Spring 2.5.5

Spring es un marco de trabajo de código abierto, popular y ampliamente utilizado, que ayuda a los desarrolladores a crear aplicaciones de alta calidad de forma mucho más rápida. *Spring* provee una programación consistente y un modelo de configuración que es bien entendido y utilizado por millones de desarrolladores de todo el mundo. A diferencia de la tradicional plataforma de desarrollo *Java* EE, *Spring* ofrece una amplia gama de

capacidades para la creación e integración de aplicaciones empresariales desarrolladas en *Java*.

Spring proporciona un modelo de programación coherente y simple en muchas áreas, por lo que es un ideal arquitectónico. *Spring* le ayuda a resolver muchos problemas, puede proporcionar una alternativa apropiada para muchas aplicaciones que es utilizar Programación Orientada a Aspectos (AOP) para ofrecer gestión de transacciones declarativa. *Spring* consta de un Modelo de Vista Controladora (MVC) que se basa totalmente en las interfaces. Además, casi todas las partes del marco de *Spring,* MVC es configurable a través de conectar su propia interfaz y estas se configuran a través de los contenedor de inversión de control (IoC).

Spring provee una manera consistente de manejar objetos de negocio a través de un IoC como bloque básico, el cual se encargará de inicializar dichos objetos en el código. Teniendo esto como principal potencialidad, *Spring* fomenta las buenas prácticas de programar orientado hacia interfaces más que a clases, si a esto se le agrega que en la mayoría de los casos es no-intrusivo, o sea, que no fuerza a modificar el código de la aplicación para beneficio de su uso, entonces hace que se le considere *marco de trabajo* mundial para construir aplicaciones. En otras palabras se está ante "un *marco de trabajo* que permitirá desarrollar una aplicación robusta, segura y fiable, haciendo el código más reusable, manejable y portable".

Visual Paradigm 3.4

La herramienta utilizada para el modelado del sistema es *Visual Paradigm*. La cual es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis, diseño orientados a objetos, construcción, pruebas y despliegue. Con la utilización de la misma se obtienen una serie de ventajas como:

- Una rápida construcción de aplicaciones con calidad y a un menor costo.
- Ofrece capacidades de ingeniería directa e inversa.
- Es una poderosa herramienta de generación de Formato de Documento Portátil (PDF)/ Lenguaje de Marcado de Hipertexto (HTML) a partir de diagramas UML.
- Permite la sincronización entre el código fuente y el modelo en tiempo real.
- Cuenta con una abundante documentación, como son: tutoriales, demostraciones interactivas y proyectos UML.

- Es una herramienta colaborativa, es decir, por lo que permite múltiples usuarios trabajando sobre el mismo proyecto.
- Permite control de versiones.
- Se puede diseñar la documentación del sistema con un diseñador de plantilla.
- Se pueden estimar las consecuencias de los cambios con los diagramas de análisis de impacto.
- Genera código *Java*.

Postgres 8.3

PostgreSQL es un poderoso gestor de base de datos relacional de código abierto. Cuenta con más de quince años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación en la comunidad de desarrollo por su confiabilidad, integridad y corrección de datos. Funciona sobre los principales sistemas operativos *Linux*, *UNIX* y *Windows*.

Entre sus ventajas se encuentran:

- Ejecuta procedimientos almacenados en más de una docena de lenguajes de programación, como *Java*, *Perl*, *Python*, C, C + +, entre otros.
- Su biblioteca de funciones tiene cientos de funciones integradas que van desde matemáticas básicas hasta las operaciones de cadena a la criptografía.
- Alta concurrencia: mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.
- Provee soporte para números de precisión arbitraria, texto de extensión ilimitada, figuras geométricas, direcciones IP, direcciones MAC y arreglos. Además los usuarios pueden crear sus propios tipos de datos.

Eclipse Galileo 3.5

Eclipse es un entorno de desarrollo integrado de código abierto, multiplataforma. Típicamente ha sido usado para desarrollar Entornos Integrados de Desarrollo (IDE), como el IDE de *Java* llamado Paquete de Herramientas de *Java (JDT)* y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones

cliente.

Eclipse emplea *plug*-in (conjunto de componentes de software que agrega capacidades específicas para una aplicación de software más grande) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. De esta forma, Eclipse puede extenderse usando otros lenguajes de programación como C, C++, *Python* y *PHP*, puede además trabajar con sistemas de gestión de base de datos.

La arquitectura *plug-in* permite escribir cualquier extensión deseada en el ambiente. En cuanto a las aplicaciones clientes, Eclipse provee al programador marcos de trabajos muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software y aplicaciones *web*.

Administrador de negocios de procesos java JBoss JBPM 4.3

JBoss jBPM es un sistema flexible y extensible de administración de flujo de trabajo.

JBoss jBPM cuenta con un lenguaje de proceso intuitivo para expresar gráficamente procesos de negocio en términos de tareas, estados de espera para comunicación asíncrona, temporizadores, acciones automatizadas,... Para unir estas operaciones JBoss jBPM cuenta con un mecanismo poderoso y extensible de control de flujo. JBoss jBPM tiene mínimas dependencias y se puede utilizar con la misma simpleza que una biblioteca *java*. Pero también puede utilizarse en ambientes donde es esencial contar con un alto nivel de producción mediante la implementación en un servidor de aplicaciones J2EE en clúster.

JBoss jBPM se puede configurar con cualquier base de datos y se puede implementar en cualquier servidor de aplicación.

JBoss jBPM también incluye una herramienta gráfica de diseño. El diseñador es una herramienta gráfica para crear los procesos de negocio. El diseñador gráfico de proceso JBoss jBPM es un plug-in para eclipse. Existe una instalación independiente de la herramienta de diseño en la hoja de ruta.

La característica más importante de la herramienta gráfica de diseño es que incluye soporte tanto para las tareas del analista de negocios como para el desarrollador técnico. Esto permite una transición armónica desde la modelación de procesos de negocio a la implementación práctica. El plug-in está disponible como sitio de actualización local

(archivo zip) para la instalación a través del mecanismo estándar eclipse de actualización de software. Y también existe un paquete que se puede descomprimir en su directorio local eclipse.

JBoss está construido entorno al concepto de la espera. Eso puede sonar extraño dado que el software esta generalmente cerca de hacer las cosas, pero en este caso hay una muy buena razón para esperar. La guía de esta secuencia de "esperar, trabajar, esperar, el trabajo" es manejado por el motor de *JBoss.* El motor de *JBoss* mira nuestra definición del proceso y se resuelve en qué dirección debe dirigirse a través del proceso. Se sabe que la "definición del proceso" mejor como nuestro mapa de procesos gráficos.

En lugar de un *software* de programación en el código, se está programando el *software* utilizando un mapa de procesos visuales: se refiere como un "grafo dirigido". En este gráfico se dirige también mediante la representación XML. El gráfico más el XML es un conjunto de notación, que es propiamente llamado *JBoss*.

Una definición del proceso se especifica en *JBoss* está compuesto de nodos, transiciones y acciones. Durante la ejecución del proceso, a medida que el ejemplo a través de la gráfica se indica, lleva a través de un "componente léxico", que es un puntero al nodo de la gráfica en la que la instancia está a la espera. Una señal, dice el testigo que la transición se debe tomar desde el nodo: las señales de especificar qué camino tomar en el proceso.

1.8 Conclusiones

En este capítulo se caracterizan las herramientas y tecnologías utilizadas tanto en la documentación del software como las utilizadas para la implementación de la aplicación. Teniendo en cuenta el análisis de los sistemas identificados a través de la literatura se llega a la conclusión de que no se pueden aplicar ninguno de los ejemplos encontrados debido a que existen barreras de tipo económico, tecnológico así como imposiciones del bloqueo. Además los sistemas existentes no proponen una solución que dé respuesta a la problemática que se presenta. Actualmente existe una tendencia en el desarrollo de sistemas relativamente complejos en las estructuras de software, donde se agrupan las funcionalidades comunes en un módulo o paquete llamado en la mayoría de los casos módulo común, o útiles.

Capítulo 2 Propuesta de solución.

2.1 Introducción

En el capítulo anterior, se recolectaron todos los datos necesarios que servirán para dar paso a la propuesta de solución. Se desea contar con un módulo que está diseñado de acuerdo a los procesos del negocio involucrados, también debe poseer las últimas tendencias de las tecnologías informáticas.

Para un mejor entendimiento de esta propuesta de solución se describen la arquitectura de software, se realiza un modelo conceptual de dominio, y los modelos de los procesos de negocio.

2.2 Situación actual del sistema

El sistema emisión de pasaportes diplomáticos cuenta con un grave problema que se da ocasionalmente en muchos proyectos de *software* con un tamaño considerable y que la industria de *software*, las tendencias y nuevas tecnologías no aceptan. La repetición de código dificulta la tarea de desarrollar y mantener la aplicación, un simple cambio puede desencadenar una serie de cambios en cascada con consecuencias imprevisibles, además que trae consigo un crecimiento insatisfactorio del proyecto, a raíz de esto se necesitarían un mayor nivel tecnológico de hardware. Influyendo estos factores en la aceptación del cliente y los costos de producción.

2.3 Descripción de los procesos involucrados

Un proceso de negocio es una colección de actividades diseñadas para producir una salida específica para un cliente o mercado particular. El Sistema Emisión de Pasaportes Diplomáticos, de Servicio, y Acreditaciones cuenta con dos procesos principales, los cuales se dividen en dos subprocesos cada uno, estos son:

Proceso emisión de pasaportes diplomáticos y de servicio.

1. Subproceso Solicitud de pasaportes diplomáticos y de servicio: el subproceso comienza cuando un solicitante, llena la solicitud de pasaporte (diplomático o de servicio) para funcionarios o familiares de esos funcionarios. Los datos de la solicitud

son almacenados de forma temporal, hasta que se compruebe la validez de los datos del ciudadano. Luego se compara los datos contenidos en SAIME con los datos de los ciudadanos que realizan solicitudes de pasaporte. Si los datos coinciden con los datos de la solicitud, entonces se realiza la búsqueda de prohibiciones que tenga el ciudadano, para el cual se solicita la confección de un pasaporte. Posteriormente se deniega o aprueba aquellas solicitudes que no tengan prohibiciones ni sean excepciones enviando una notificación al ciudadano. El solicitante una vez recibida la notificación de aprobación y el serial del trámite puede reservar la cita.

2. Subproceso Enrolamiento de pasaportes diplomáticos y de servicio: en este subproceso inicia cuando el ciudadano se presenta en la cita previamente reservada, se procede a la captura de los datos biométricos que no fueron registrados como parte de la solicitud. Se imprime la planilla de control para validar con el ciudadano que todos los datos están correctos. Luego se revisan todos los datos contenidos en el sistema de acuerdo a la planilla de control que se le entregó al ciudadano. Una vez que el AFIS (Sistema Automático de Identificación de Huellas Dactilares) emitió su respuesta sobre la identidad del solicitante y la misma es correcta, el director aprueba el trámite para que se pueda imprimir el documento.

Proceso emisión de acreditaciones

1. Subproceso Solicitud de acreditaciones: este subproceso empieza cuando se realiza, por parte de los usuarios (solicitantes) una solicitud de una acreditación diplomática. Los datos de la solicitud son almacenados temporalmente hasta tanto no se compruebe la validez de los mismos. Luego se compara los datos contenidos en SAIME con los datos de los ciudadanos que realizan solicitudes de acreditaciones. Se revisa las solicitudes se deniegan si presentan algún error o se aprueban en caso de estén correctas, se emite una notificación al ciudadano cuando sea aprobada la solicitud. El ciudadano al recibir la notificación reserva una cita para el inicio del trámite.
2. Subproceso Enrolamiento de acreditaciones: en este se procede a capturar el resto de los datos necesarios. Una vez que el AFIS emitió su respuesta sobre la identidad del solicitante y la misma es correcta, el director aprueba el trámite para que se pueda imprimir el documento.

2.4 Información que se maneja

La información es un conjunto organizado de datos, que constituye un mensaje sobre un cierto fenómeno o ente.

- ✓ Solicitud online de pasaporte (diplomático o de servicio) para funcionarios o familiares de esos funcionarios.
- ✓ Datos del ciudadano (célula).
- ✓ Datos contenidos en SAIME.
- ✓ Notificación de aprobación de solicitud, la cual contiene el serial del trámite, y un link para que consulte en el portal los recaudos que debe llevar a la captura de datos.
- ✓ Datos que no fueron registrados como parte de la solicitud, como son características físicas, recaudos, acompañantes de viaje y vínculos con funcionarios (los dos últimos datos solo se registran en el caso de la solicitud de un funcionario).
- ✓ Datos biométricos del ciudadano (Huella, firma, foto)
- ✓ Notificación enviada por el AFIS
- ✓ Nota de elaboración: Documento que emite el organismo solicitante como oficio de solicitud de pasaporte.
- ✓ Planilla de pasaporte diplomático: Planilla de pasaporte que se identifica por su color amarillo.
- ✓ Cédula de identidad: Dato para almacenar en el cuaderno de entrega de pasaportes, identificador de usuario.
- ✓ Oficio de notificación: Es la constancia de que la revisión de los recaudos fue exitosa y por tanto se puede continuar con el trámite.
- ✓ Sello húmedo y personalizado: Indicador que refleja la disponibilidad final del pasaporte en cuestión.
- ✓ Cuaderno de entrega de pasaportes: Donde se registran los datos del usuario que recogió su pasaporte, en caso de no estar presente el mismo se registra el autorizo o la copia del oficio de solicitud.
- ✓ Copia del oficio de solicitud (Copia de la nota de elaboración): Documento que se presenta en caso de que sea el ente gubernamental quien se presenta a recoger el pasaporte.

- ✓ Autorización: Se refiere al documento que debe presentarse en caso de que no sea el titular de pasaporte quien se presenta a recoger el mismo.

2.5 Propuesta de solución

En las aplicaciones grandes y complejas, el modelo de Dominio tiende a crecer extraordinariamente. El modelo llega a un punto donde es complicado hablar sobre ello como "un todo", y puede costar bastante entender bien todas sus relaciones e interacciones entre todas sus áreas. Por esa razón, se hace necesario organizar y fraccionar el modelo en módulos heterogéneos[4]. Los módulos se utilizan como un método de organización de conceptos y tareas relacionadas (normalmente bloques de negocio diferenciados) y poder así reducir la complejidad desde un punto de vista externo. Una vez que se entienden las interrelaciones entre los módulos, es más sencillo focalizarse en más detalle de cada uno de ellos. Esta es una forma simple y eficiente de gestionar la complejidad. El lema "Divide y Vencerás" es la frase que mejor lo define.

Otra razón por la que hacer uso de módulos está relacionada con la calidad del código. Es un principio aceptado por la industria el hecho de que el código debe tener un alto nivel de cohesión y un bajo nivel de acoplamiento.

Los módulos comunes proponen una solución muy óptima, cumpliendo con muchas de las exigencias mencionadas anteriormente.

Como se viene indicando en el desarrollo de esta investigación. El módulo que se propone desarrollar se encuentra en la capa de componentes específicamente en la subcapa de negocio, donde se expondrán las funcionalidades o servicios comunes necesarios que utiliza la capa de procesos para su completo funcionamiento, el módulo se comunica con la capa de acceso a datos para obtener la información necesaria mediante las clases de objeto de acceso a datos (DAOs).

2.6 Modelo conceptual de dominio

Un modelo es una representación abstracta de algo para su posterior construcción. Teniendo en cuenta el contexto del sistema se construye el modelo conceptual de dominio.

[4] Compuesto de componentes o partes de distinta naturaleza:

El siguiente modelo conceptual de dominio define los conceptos del negocio tratados: el usuario(es un ciudadano funcionario, personal de servicio o un menor) los usuarios realizan una solicitud(que puede ser de pasaporte o acreditación), los datos de esta solicitud se comprueban con SAIME, el usuario posee un documento de identificación(pasaporte diplomático, pasaporte de servicio o acreditación) y datos biométricos(color del cabello, color de los ojos, forma del rostro que puede ser diamante, rectángulo, ovalado, circular, cuadrado o triangular; fotografía, firma y huellas), estos datos biométricos se comparan con el AFIS.

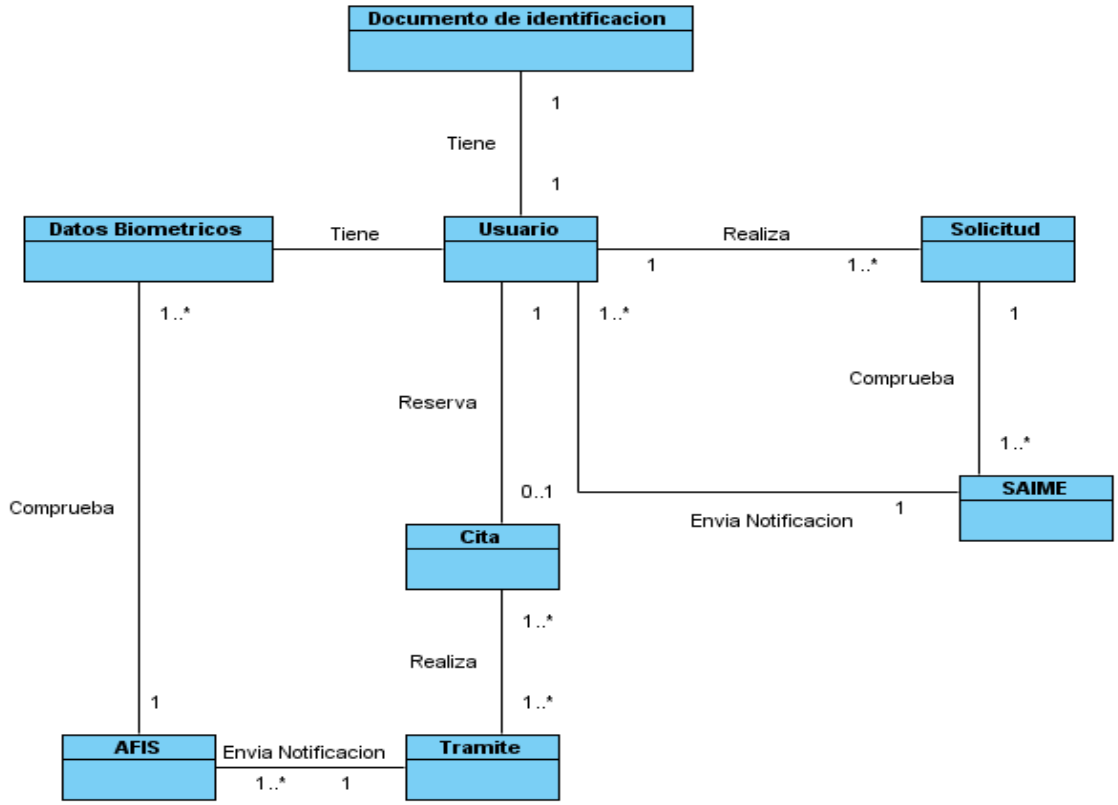

Figura 5 Modelo conceptual de dominio general.

2.7 Conceptos fundamentales tratados

Usuario: Ciudadano que solicita o se le realiza la tramitación del documento de identificación.

Solicitud: Petición realizada por el ente gubernamental para la realización de un

documento de identificación.

Datos biométricos: Conjunto de información que identifica al ciudadano

Documento de identificación: Documento que se expide al finalizar el proceso de tramitación que pueden ser acreditación, pasaporte diplomático o de servicio.

Cita: Encuentro con hora, fecha y lugar definidos para completar con un proceso de negocio.

Trámite: Proceso que se realiza para el enrolamiento de los datos de un ciudadano, con el fin de expedir un documento de identificación.

AFIS: Sistema que valida que la persona es quien dice ser.

SAIME: Sistema que valida los datos de los ciudadanos y buscar prohibiciones que puedan tener estos.

2.8 Modelo de procesos del negocio

El modelado de procesos de negocio es la base para comprender mejor la operación de una organización, documentar y publicar los procesos buscando una estandarización en la organización, buscar eficiencias en la operación e integrar soluciones en arquitecturas orientadas a servicios.

Se muestran los modelos de los procesos principales del sistema emisión de pasaportes.

Consultar anexo 2 para ver los modelos de los procesos y subprocesos.

2.9 Especificación de los rasgos del software.

Los requerimientos para un sistema de software determinan lo que hará el sistema y definen las restricciones de su operación e implementación, tanto en lo que se refiere al comportamiento interno (gestión de los datos) como al externo (interacción con el usuario y con otras aplicaciones). De acuerdo al modelo conceptual del dominio se identificaron p rasgos funcionales y los respectivos rasgos no funcionales.

2.9.1 Lista de rasgos funcionales

A partir del modelo conceptual de dominio se definieron una serie de características o rasgos organizados jerárquicamente:

RF1 Gestionar una solicitud de pasaporte

1.1. Registrar una solicitud de pasaporte

- Solicitud (pasaporte)
- Tipo del trámite (Solicitud de pasaporte para funcionarios, Solicitud de pasaporte para familiares)
- Tipo de pasaporte (diplomático, de servicio)
- Tipo de solicitud (emisión, renovación)

1.2. Modificar datos de una solicitud de pasaporte

- Actualizar los datos de la solicitud de pasaporte

1.3. Eliminar solicitud de pasaporte

RF2 Gestionar una solicitud de acreditaciones.

- Registrar una solicitud de acreditaciones
- Solicitud (Acreditación)
- Tipo del trámite (Solicitud de acreditación para funcionarios diplomáticos, Solicitud de acreditación para familiares de diplomáticos, Solicitud de acreditación para personal de servicio doméstico)
- Tipo de solicitud (emisión, renovación)
- Extranjero
- Venezolano

2.1. Modificar datos de una solicitud de acreditación

- Mostrar un listado de las solicitudes de acreditaciones
- Actualizar los datos de la solicitud de acreditación

2.2. Eliminar solicitud de acreditación

RF3 Gestionar trámites.

- Registrar un nuevo trámite.
- Registrar recaudos.
- Modificar datos de un trámite.
- Cancelar tramites
    - Registrar causas de la cancelación.

RF4 Gestionar datos del usuario

4.1. Registrar datos de un usuario

- Primer Nombre

- Segundo Nombre
- Primer Apellido
- Segundo Apellido
- Venezolano
- Extranjero
- Correo electrónico
- Nombre de usuario
- Organización

4.2. Modificar datos del usuario

- Mostrar un listado de los usuarios
- Actualizar los datos del usuario

4.3. Eliminar usuario

RF5 Permitir el registro de características físicas (datos biométricos).

- Color del cabello
- Color de los ojos
- Forma del rostro (Diamante, Rectángulo, Ovalado, Circular, Cuadrado, Triangular)
- Capturar fotografía, firma y huellas.

RF6 Gestionar cita

Registrar datos de una nueva cita

- Fecha de la cita
- Lugar de la cita
- Hora de la cita
- Especificación de la cita

6.1. Modificar datos de una cita

6.2. Eliminar una cita

RF7 Enviar notificación

7.1. Enviar notificación de aprobación de trámites

- Serial
- Dirección

7.2. Enviar notificación de denegación de una solicitud

- Tipo de error (Datos, Nota verbal, Pasaporte sin Visa, Por reglamento)

- Observaciones

RF8 Autenticar usuario

8.1. Gestionar usuario

- Adicionar un nuevo usuario
- Modificar usuario
- Eliminar usuario

8.2. Gestionar contraseña

- Adicionar una contraseña
- Permitir el cambio de la contraseña.

RF9 Gestionar rol

- Adicionar un nuevo rol
- Modificar un rol.
- Eliminar rol.

2.9.2 Lista de rasgos no funcionales

Los rasgos no funcionales son propiedades o cualidades que el software debe cumplir.

RnF 1 Eficiencia

1.1. Rendimiento

Capaz de aceptar varias peticiones a la vez.

1.2. Recursos de *hardware*

Un giga byte de memoria RAM (memoria de acceso aleatorio).

De espacio en disco duro 120 GB.

RnF 2 Restricciones en el diseño

2.1. Plataforma de desarrollo

J2EE (Plataforma Java, Edición Empresarial)

Framework Spring para la integración de toda la aplicación.

Framework Hibernate para el manejo y control de los datos persistentes.

2.2. Ambiente de desarrollo

Netbeans para la creación de los clientes de escritorio y eclipse para la automatización e implementación de los componentes de software.

2.3. Requisitos de licencia

Framework Spring bajo la licencia Apache v2.0 para la integración del Sistema.

El gestor de base de datos se encuentra bajo licencia Artistic License.

2.10 Planeación de las iteraciones por rasgos

La metodología FDD (Desarrollo Basado en Funcionalidades) utilizada en esta investigación se basa en un proceso iterativo con iteraciones cortas, a continuación se muestra un cronograma donde se especifica la duración del diseño y la construcción de cada uno de los rasgos. Comenzando por los rasgos más significativos y teniendo en cuenta la dependencia, para no incluir dentro de la misma iteración rasgos con dependencia entre ellos mismos, a no ser rasgos que no se estén implementando. Se definieron 4 iteraciones planeadas de la siguiente forma:

Iteración 1: Gestionar datos de una solicitud de pasaporte y de acreditaciones.

Iteración 2: Gestionar trámites, gestionar citas.

Iteración 3: Gestionar los datos de un usuario, enviar notificación.

Iteración 4: Autenticar usuario, gestionar rol.

A continuación se describe la iteración 1, para ver las iteraciones 2, 3 y 4 consultar **anexo 3.**

Iteración 1

GESTIONAR SOLICITUD DE PASAPORTE

Rasgo	Modelo General	Diseño	Construcción	Prueba
Crear una solicitud de pasaporte.	28/01/2011	1/02/2011	2/02/2011	17/02/2011
Modificar datos de una solicitud de pasaporte.	28/01/2011	1/02/2011	7/02/2011	17/02/2011
Eliminar una solicitud de pasaporte.	28/01/2011	1/02/2011	10/02/2011	18/02/2011

Tabla 1 Planeación del rasgo registrar solicitud de pasaporte.

GESTIONAR SOLICITUD DE ACREDITACIÓN

Rasgo	Modelo General	Diseño	Construcción	Prueba
Crear una solicitud de acreditación.	28/01/2011	14/02/2011	14/02/2011	18/02/2011

Modificar datos de una solicitud de acreditación.	28/01/2011	14/02/2011	15/02/2011	19/02/2011
Eliminar una solicitud de acreditación.	28/01/2011	14/02/2011	16/02/2011	19/02/2011

Tabla 2 Planeación del rasgo registrar solicitud de acreditación.

2.11 Patrones de diseño

Patrones de diseño o más comúnmente conocidos como "Design Patterns". ¿Qué son los patrones de diseño? Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan.

Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Han revolucionado el diseño orientado a objeto.

Los patrones del diseño utilizados en el desarrollo de los componentes son:

- Patrones generales del software para asignación de responsabilidades (Grasp).

Bajo acoplamiento: uno de los principios para proteger al software frente al cambio es mantener bajo el acoplamiento entre clases. El acoplamiento de una clase es el conjunto de dependencias que tiene con otras clases, cuanto menor sea el acoplamiento entre clases, menor influencia tendrán los cambios. En la solución que se ofrece, cada clase se relaciona sólo con quien lo necesita para realizar sus procedimientos (o métodos). Como ejemplo se tiene que: para realizar las acciones del rasgo Enviar notificación, la clase NotificacionFacade recurre a Notificacion solamente, pues es la clase de dominio representante, en el sistema, de la tabla perteneciente a la base de datos del negocio o enlace entre los atributos de la clase y las columnas de la tabla de la base de datos. (11)

Alta cohesión: al asignar responsabilidades en el diseño, se buscan soluciones que asignen los métodos a las clases de forma coherente, completa y relacionada. De esta forma, se obtienen clases cohesionadas. Las ventajas son evidentes. Una clase cohesionada facilita el cambio. Al realizar un cambio en una clase muy cohesionada,

todos los métodos que pueden verse afectados, toda la información que se necesita controlar, estará a la vista, en el mismo fichero. Este patrón se relaciona con el de bajo acoplamiento, porque un diseño cohesionado tendrá un bajo acoplamiento entre clases. Ejemplos de este patrón se pueden encontrar en todas las clases de la capa de componentes de la capa de negocio, donde cada clase realiza los métodos que le competen, según su concepción y finalidad.

Experto: se tiene en consideración a qué clase debe pertenecer un método, este principio sugiere que se asigne a la clase que más sepa del método (es decir al experto). Esto es una consecuencia del principio de alta cohesión, ya que si se asignan los métodos a las clases que tienen la información necesaria para ejecutarlos, se están creando clases altamente cohesionadas. Un ejemplo sería la clase SolicitudFacadeImpl, la cual es la responsable de realizar todas las operaciones acerca de una solicitud, por lo que los métodos relacionados con las operaciones básicas sobre los locales estarán en la misma. Asignados

- Patrones de diseño a objetos (GoF).

Patrones de creación: Muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resultas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades.

- **Builder**: Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.
- **Factory Method**: Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

Patrones estructurales: Describen la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros

- **Facade**: Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema es más fácil de usar.

2.12 Estructura de los componentes

Los diagramas de paquetes se usan para reflejar la organización de los paquetes y sus elementos. Este diagrama muestra cómo los componentes se dividen en agrupaciones lógicas, de igual manera muestra las dependencias que existen entre estas agrupaciones donde presentan los componentes que exponen los servicios comunes:

Administración: Contiene los servicios gestionar rol y gestionar usuario.

Biometría: Contiene los componentes que exponen los servicios relacionados con datos biométricos de un ciudadano.

Identidad: Contiene los servicios concernientes a la gestión de un ciudadano.

Solicitud: Posee los servicios de la gestión de una solicitud y organismo.

Trámite: Tiene todo los tipos de recaudos y su gestión, además de las entidades legales.

Útiles: Contiene los componentes que exponen los servicios referentes a las citas, notificación, nomenclador y el control de mensajes.

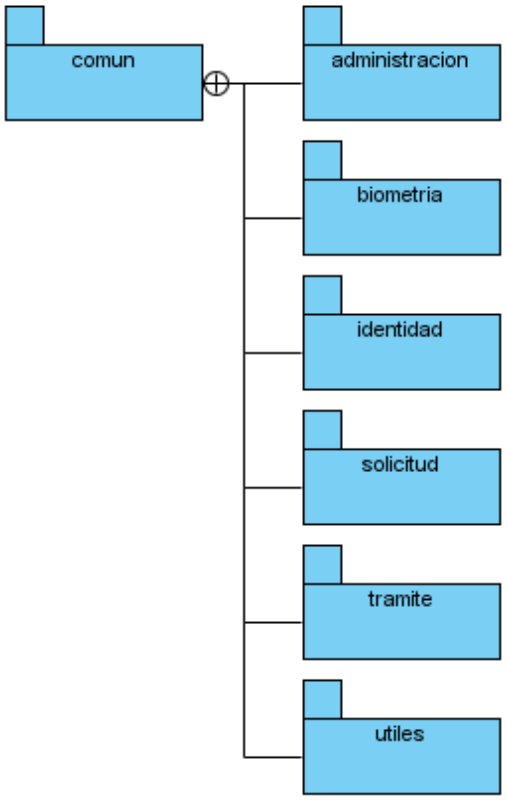

Figura 6 Diagrama de paquetes del módulo común

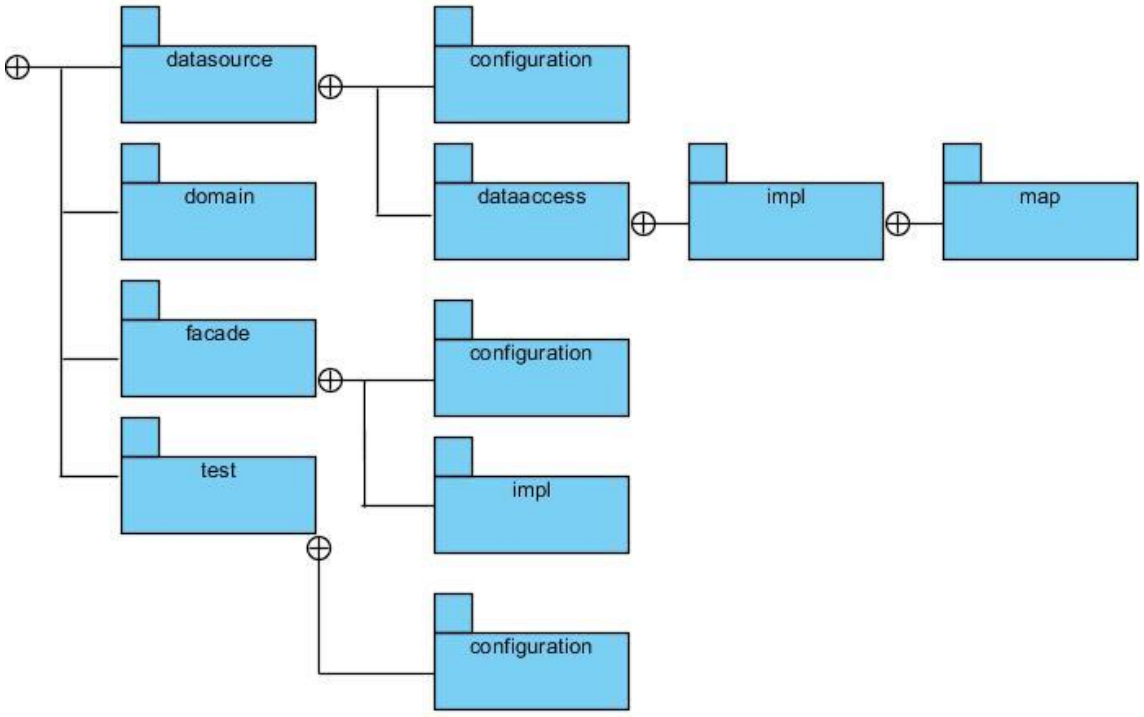

Figura 7 Diagrama de subpaquetes del módulo común

Paquete **datasource**: contiene todos los archivos de configuración, clases de consumo de servicios y acceso a datos.

Paquete **configuration**: contiene los archivos de configuración de los servicios, conexión a la base de datos.

Paquete **dataaccess**: contiene los archivos necesarios para el acceso a datos.

Paquete **impl**: contiene la implementación de los archivos.

Paquete **map**: contiene los archivos de mapeo.

Paquete **domain**: contiene todas las clases de dominio o persistentes con que se trabajan.

Paquete **facade**: contiene todas las clases de negocio con que se trabajan.

Paquete **test**: contiene todas las clases de pruebas para las diferentes capas de abstracción que se tienen.

2.13 Conclusiones

En el finalizado capítulo se realizaron las primeras fases de acuerdo a la metodología de desarrollo basada en rasgos, donde se identificaron las principales clases del sistema, con su referente lista de rasgos. Se determinó la realización de 4 iteraciones para la entrega completa de los servicios comunes del negocio.

Con el desarrollo del módulo común, como fue nombrado por el equipo de trabajo del proyecto transformación se logrará la definitiva interoperabilidad e integración de los servicios necesarios que se exponen. La propuesta brinda una visión general de cómo será el modulo después de concluido su desarrollo.

Capítulo 3 Servicios comunes, elaboración y validación

3.1 Introducción

El diseño y descripción de las clases de los componentes, a través de los diagramas de clases del diseño de cada uno de ellos sirven de gran ayuda pues ayudan a elegir que clases desarrollar así como que propiedades y métodos se deben de implementar mostrando las relaciones entre ellos. Hacer las pruebas de un programa implicaría ponerlo en todas las situaciones posibles, así se requiere un conjunto de pruebas para asegurarse del correcto funcionamiento libre de errores de los servicios comunes, alcanzando además validar la hipótesis planteada.

3.2 Modelo de clases del diseño de los componentes

El modelo de clases es el diagrama principal para el análisis y diseño. Un modelo de clases presenta las clases del sistema con sus relaciones estructurales y de herencia. La definición de clase incluye definiciones para atributos y operaciones.

3.2.1. Clasificación de componentes

A continuación se muestra la clasificación de los componentes del módulo común, partiendo de tres aspectos importantes:

Dependencia: es la relación o la comunicación que necesita de otro componente para su desarrollo. En la columna perteneciente a la dependencia se especificaran el número de los componentes que él depende.

Complejidad: se refiere a la complejidad de desarrollo o implementación que se necesita realizar para logra obtener un componente funcional. En la columna perteneciente a la Complejidad se evaluara en Baja o Media o Alta.

Prioridad: se refiere a la importancia que tiene ese componente para la arquitectura del proyecto para la realización del mismo. En la columna perteneciente a la Prioridad se evaluara de 1 a 5, para señalar lo que requiere de mayor atención se le dará la calificación de 5 y descendentemente la importancia - calificación.

No	Componentes	Dependencia	Complejidad	Prioridad
1	SolicitudFacade	6, 7	Alta	5
3	CitaFacade		Media	5
4	ControlMensajesFacade		Baja	3
5	DireccionFacade		Media	5
6	NomencladorFacade		Baja	5
7	NotificacionFacade		Baja	4
8	BiometriaFacade		Alta	5
9	ControlBiometricoInternoAFIS_SAIMEFacade	10	Media	5
10	ControlBiometricoExternoAFIS_SAIMEFacade		Media	1
11	CiudadanoFacade	6	Alta	5
12	EstadoCiudadanoFacade		Media	5
13	AccesoFacade		Alta	5
14	EntidadLegalFacade		Baja	4

Tabla 3 Clasificación de los componentes

3.2.2. Diseño de los componentes

Describe gráficamente las especificaciones de las clases, interfaces y sus relaciones que serán usadas en el *software*. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa. Los diagramas se utilizan generalmente para facilitar el entendimiento de largas cantidades de datos y la relación entre diferentes partes de los datos. A continuación se encuentran los diagramas de los componentes solicitud, biometría y ciudadano.

Para ver los diagramas de los restantes componentes consultar **anexo 4.**

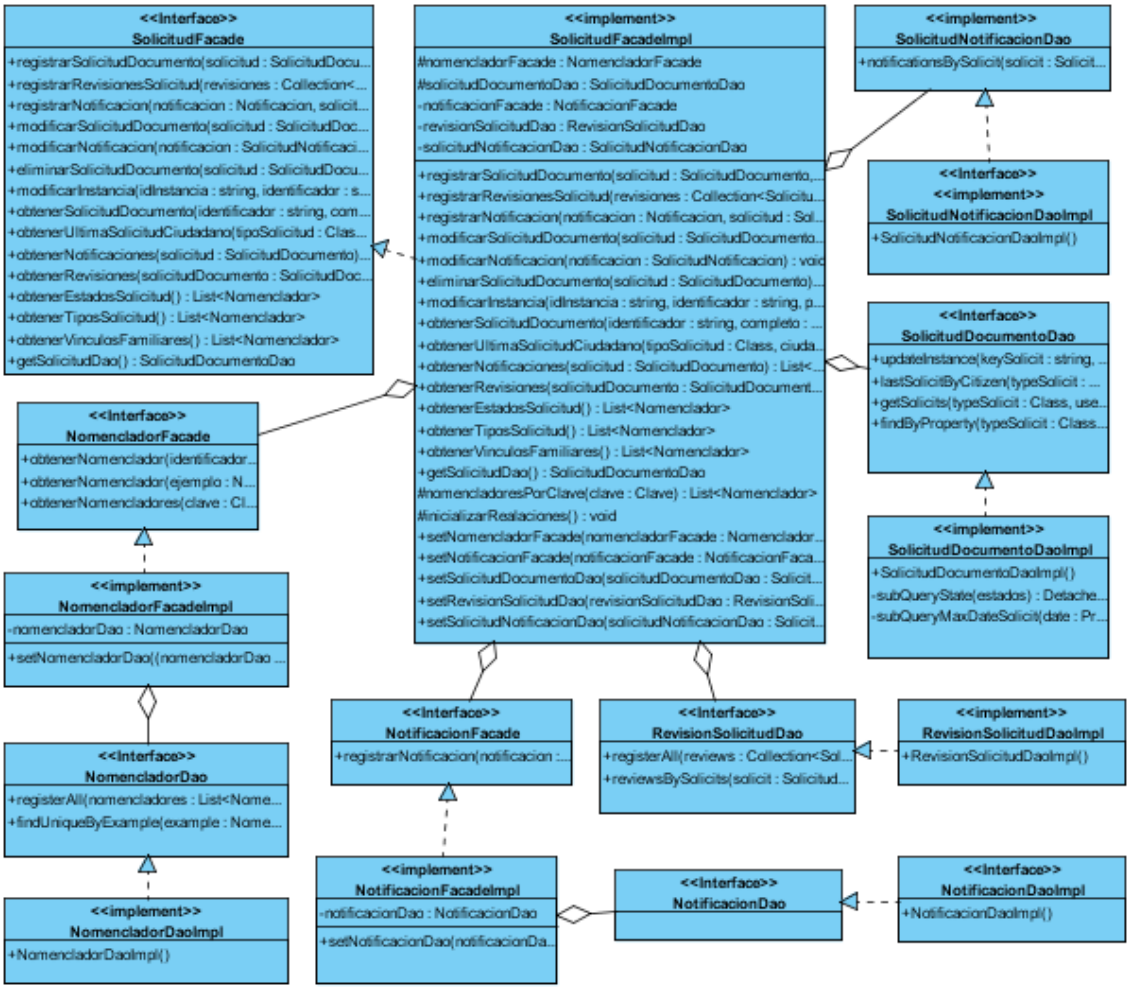

Figura 8 Diagrama de clases del diseño del componente Solicitud

3.2.3. Descripción de los componentes

Componente		
Nombre	**Tipo**	**Descripción**
SolicitudFacade	Interfaz	Está concebida para la gestión y manejo de las acciones básicas sobre el objeto **Solicitud**, esta interfaz es la que interactúa directamente (o sea a través de DAOs) con la base de datos.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**

registrarSolicitudDocumento	String	Registra un documento **Solicitud** realizada por un usuario que tenga los permisos para realizar esta acción y devuelve como resultado el id de la solicitud registrad.
registrarRevisionesSolicitud	void	Registra una colección de documentos de **Revisión de Solicitud.**
registrarNotificacion	String	Registra la notificación que se le realiza a una **Solicitud** y devuelve como resultado el id de la **Solicitud de Notificación.**
modificarSolicitudDocumento	void	Modifica o cambia el documento **Solicitud**.
modificarNotificacion	void	Modifica o cambia el documento la **Solicitud de Notificación**.
modificarInstancia	boolean	Modifica o cambia la instancia del documento **Solitud.**
eliminarSolicitudDocumento	void	Elimina dado una instancia u objeto.
obtenerSolicitudDocumento	SolicitudDocumento	Obtiene como resultado una objeto (SolicitudDocumento) dado su identificador y una variable boolean que si está en true devuelve el objeto con todos sus datos e instancias y en caso contrario devuelve el objeto pero solo con los datos atómicos.
obtenerUltimaSolicitudCiudadano	SolicitudDocumento	Obtiene la última **Solicitud** realizada por un Ciudadano.
obtenerNotificaciones	List<SolicitudNotificacion>	Devuelve una colección de las notificaciones realizada a una **Solicitud.**
obtenerRevisiones	List<SolicitudRevision>	Devuelve una colección de las revisiones realizada a una **Solicitud.**
obtenerEstadosSolicitud	List<Nomenclador>	Devuelve todos los estados que puede tener una **Solicitud**.
obtenerTiposSolicitud	List<Nomenclador>	Devuelve todos los tipos que puede ser una **Solicitud**.
obtenerVinculosFamiliares	List<Nomenclador>	Devuelve todos los tipos de vínculo familiar.

Tabla 4 Descripción de la clase SolicitudFacade

Para ver las descripciones de los restantes componentes consultar **anexo 5.**

3.3 Descripción de clases y métodos

3.3.1. Estándar de codificación

Las convenciones o estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para

facilitar la lectura, comprensión y mantenimiento del código.

3.3.1.1. Lenguaje java

3.3.1.1.1. Criterios de calidad

Es de vital importancia que durante el desarrollo se tengan en cuenta permanentemente los siguientes criterios de calidad:

Criterio	Objetivo
Facilidad de comunicación	Proporcionar a los usuarios y desarrolladores entradas y salidas fácilmente asimilables.
Descripción	Proporcionar y/o plasmar en el código detalles y explicaciones sobre la implementación realizada.
Simplicidad	La implementación realizada debe hacerse de la forma más comprensible posible.

Tabla 5 Criterios de calidad

3.3.1.1.2. Declaraciones de las clases o interfaces (estructura básica general)

Cada clase o interfaz deberá tener la siguiente estructura general, cuyos detalles se explicaran más adelante:

```
package java.ourprojectname.somemodule;

import java.somepackage.Class;

/**
 * La descripción de la clase será escrita aquí.
 *
 * @author  Nombre del equipo de desarrollo que creó la clase (Procesos, Negocio o Presentación).
 */

 public class Example extends OtherClass {
    public static final String VAR = "SomeName";
    public Object instanceVar1;
    protected int instanceVar2;
    private Object[] instanceVar3;

    /**
     * ...documentación o javadoc del constructor Example aquí...
     */
     public Example() {
        // ...la implementación va aquí...
     }
```

```
    /**
     * ...documentación o javadoc del método doSomething aquí...
     * @param someParam ...descripción del parámetro utilizado...
     * @return ... detalle de lo que devuelve ...
     * @throws DataAccessException ... motivos y escenarios en los que se lanza la excepción ...
     */
    public double doSomething(Object someParam) throws DataAccessException{
        // ...la implementación va aquí...
    }

    //Support Methods ...(en caso de que se necesiten metodos de soporte)...
    private void otherMethod() {
        // ...la implementación va aquí...
    }

    //Encapsulation
    public int getInstanceVar2() {
        return instanceVar2;
    }

    public void setInstanceVar2(int instanceVar2) {
        this.instanceVar2 = instanceVar2;
    }
}
```

Figura 9 Estructura de clase

3.3.1.2. Marco de trabajo de Spring

3.3.1.2.1. Normas de configuración de Spring

En aras de lograr una mayor claridad a la hora de concebir el archivo XML de configuración de Spring, se utilizarán (siempre que se pueda) las formas abreviadas de configuración. Esto ayudará a la hora de leer los archivos, y por ende entenderlos. La configuración sería más o menos así:

```
<bean id="exampleFacade" class="org.module.process.specificClass">
   <property name="firstVar" value="Primera"/>
   <property name="objectDao" ref="objectDao"/>
</bean>

<!-- En detrimento de esto segundo -->

<bean id="exampleFacade" class="org.module.process.specificClass">
   <property name="firstVar">
       <value>Primera</value>
   </property>
   <property name="objectDao">
       <ref bean="objectDao">
   </property>
</bean>
```

Figura 10 Estructura de configuración de Spring.

3.3.1.2.2. Encabezado de los archivos de configuración

Resulta muy útil agregar un encabezado en los archivos de configuración, en el cual se detallará brevemente la finalidad de los *beans* definidos en el archivo, todo esto para mejorar en comprensión y organización.

```
<beans>
   <description>
       En este archivo se definen los beans de negocio pertenecientes
       al módulo de solicitud de pasaportes, los mismos dependen de
       los beans de acceso a datos.
   </description>
</beans>
```

Figura 11 Estructura de los encabezados de los archivos de Spring.

3.3.1.2.3. Convenciones de nomenclatura

Siguiendo la misma filosofía que se utilizará en el uso del lenguaje Java, se abogará por una nomenclatura descriptiva, clara y consistente, para hacer más liviano el entendimiento del XML de configuración. Para el "id" de los beans se utilizará el mismo nombre de la interfaz pero con letra inicial en minúscula, alternativamente se hará lo mismo con el nombre de los atributos de negocio, de esta forma se hará más intuitiva la configuración y programación dentro del framework. Con el siguiente ejemplo se ilustraría mejor lo que se

quiere:

```
<bean id="ciudadanoDao" class="org.module.process.impl.CiudadanoDaoImpl">

</bean>

<bean id="ciudadanoFacade" class="org.module.process.impl.CiudadanoFacade">
   <property name="ciudadanoDao" ref="ciudadanoDao"/>
</bean>
```

Figura 12 Convención de nomenclatura.

Con sus respectivas implementaciones:

```
public interface CiudadanoDao {
   ...
}

public interface CiudadanoFacade {
   ...
   ...
}
```

```
public class CiudadanoDaoImpl {
    ...
}

public class CiudadanoFacadeImpl {
    private CiudadanoDao ciudadanoDao;
    ...
}
```

Figura 13 Implementación de nomenclatura.

3.3.1.2.4. Otros aspectos

También se deberá tener en cuenta que cada vez que se realicen cambios en un objeto de negocio del sistema, no se puede olvidar actualizar los archivos de configuración en caso de que sea necesario. No se puede pasar por alto que dentro del trabajo con esta herramienta, el archivo XML de configuración es parte del código, y un aspecto crítico dentro del sistema, incluso la mayoría de las veces será necesario leer tantos los archivos XML de configuración como el código Java para entender cómo funciona algún componente. Asimismo también habrá que tener presente que no se puede abusar de la inyección de dependencia, habrá que analizar cuando hay que beneficiarse de su uso, esto no es tan trivial como parece y pasarlo por alto podría ser un grave error. La falta de control sobre el crecimiento de los *beans* definidos puede incidir en que se sobrecarguen los archivos de configuración, lo que los hará más complicados y por ende más difíciles de manejar.

3.4 Validar solución.

Las pruebas de software, son los procesos que permiten verificar y revelar la calidad de un producto de software. Básicamente es la fase del desarrollo de software donde se prueban las aplicaciones construidas con el objetivo de identificar posibles fallos de implementación, calidad, o usabilidad de un sistema, demostrando así la validez de la hipótesis planteada. (9)

El modelo de pruebas que se concibió para los componentes incluye:

- Pruebas unitarias: para asegurar que el módulo **común** funcione correctamente. Como parte de estas pruebas se aplicará el método de caja blanca y se utilizó la herramienta *JUnit*.
- Pruebas de integración: tienen como objetivo fundamental probar el software cuando los módulos individuales de software son combinados como un grupo, para probar el correcto funcionamiento de los mismos en su conjunto. Para realizar esta comprobación se utilizó la herramienta *JUnit.*
- Prueba para comprobar la funcionalidad de los servicios ofrecidos por los componentes desarrollados. Para estas pruebas se aplicara el método de caja negra.

3.4.1 Pruebas unitarias con la herramienta *JUnit*

Cuando se implementa un software, resulta recomendable comprobar que el código que se ha escrito funciona correctamente, para ello se implementan pruebas que verifican que el programa genera resultados esperados esta pruebas se conocen como pruebas de unidad o unitarias.

Una prueba de unidad pretende probar cada función en un archivo de programa, las pruebas de unidad no sustituyen a las pruebas funcionales de calidad, pero a la larga reducen la cantidad de errores en futuras versiones del producto, influenciando positivamente en el diseño del programa. Las pruebas de unidad realizadas sobre los componentes que exponen los servicios comunes permitieron detectar errores en la implementación de algunas funcionalidades como se muestra en el siguiente diagrama. (16)

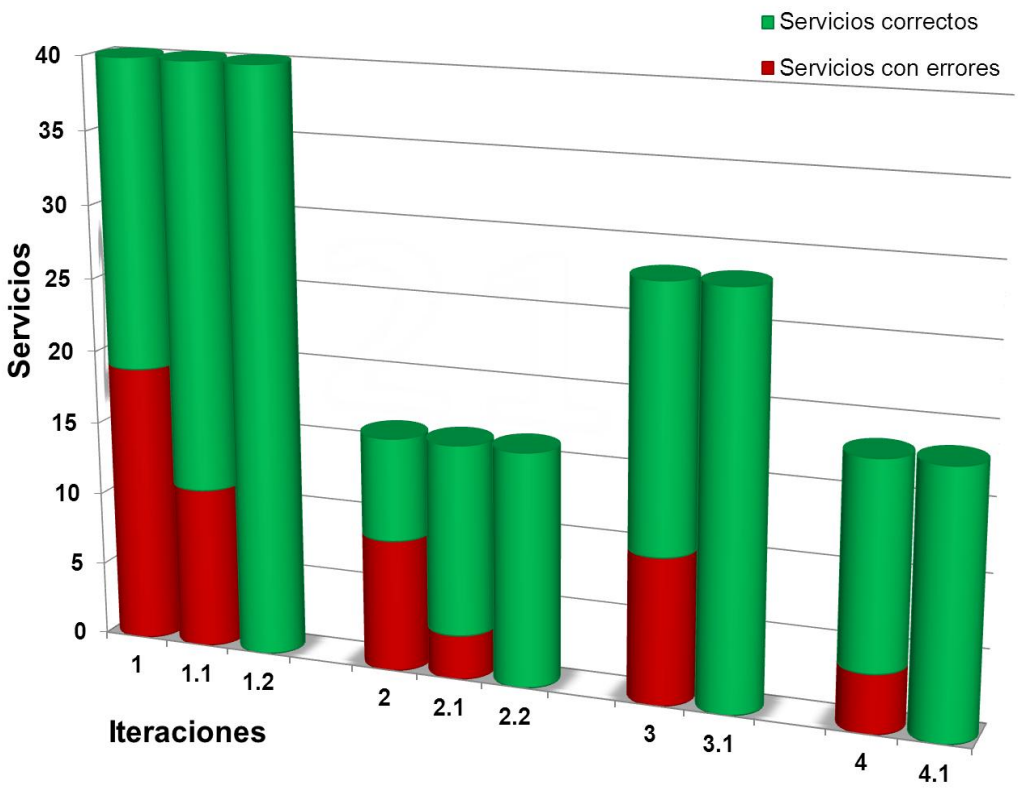

Figura 14 Pruebas unitarias realizadas a las funcionalidades.

3.4.2 Método de caja blanca

Las pruebas de caja blanca denominadas también como pruebas de cobertura o pruebas de caja transparente, la cobertura es un número porcentual que indica cuanto código del programa se ha probado.

Básicamente la idea de pruebas de cobertura consiste en diseñar un plan de pruebas en las que se vaya ejecutando sistemáticamente el código hasta que haya corrido todo o la gran mayoría de él.

En las pruebas de caja blanca, se definen la prioridad de cada uno de los métodos de acuerdo a la significación que tenga la funcionalidad que implementa. Una vez definidos los métodos más importantes para el funcionamiento de la aplicación, serán creados los casos de prueba asociados a éstos, definiendo los valores a los que deberá responder correctamente el sistema. (22)

De acuerdo a la porción de código correspondiente al rasgo **Registrar una solicitud**, perteneciente a la clase SolicitudFacadeImpl, del componente solicitud, dentro del módulo

común del paquete Solicitud, se le realizó la prueba de caja blanca:

```
public String registrarSolicitudDocumento(SolicitudDocumento solicitud,                    1
AccesoUsuario usuario) {
        String identificador = null;                                                       1

        if (usuario != null && usuario.getOrganismo() != null) {                           2

                OrganismoSolicitanteOrganizacion organismoSolicitante = new                3
                OrganismoSolicitanteOrganizacion(usuario.getOrganismo(),usuario);

                solicitud.setOrganismoSolicitanteOrganizacion(organismoSolicitante);       3

                        identificador = solicitudDocumentoDao.save(solicitud);             3
                }

                return identificador;                                                      4
        }
```

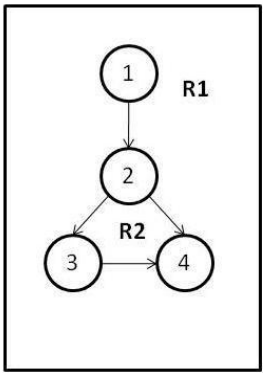

Figura 15 Grafo del caso de prueba registrar solicitud

Complejidad Ciclomática:

Es una medida que proporciona una idea de la complejidad lógica de un programa.

- ✓ La complejidad ciclomática coincide con el número de regiones del grafo de flujo.
- ✓ La complejidad ciclomática, V(G), de un grafo de flujo G, se define como **V(G)** = Aristas - Nodos + 2

V (G)= 4 – 4 + 2

V (G)= 2

Caminos:

A partir del valor de la complejidad ciclomática se obtiene el número de caminos independientes, que dan un valor límite para el número de pruebas que se tienen que diseñar.

Posibles caminos: 1-2-3-4, 1-2-4

Camino: 1-2-3-4

Caso de prueba: Registrar solicitud

Entrada: Recibe una solicitud válida, y un usuario válido.

Resultado: Se chequea que el usuario sea válido, se obtienen el organismo solicitante, por último se registra (o inserta) la solicitud, devolviendo el identificador con el cual fue guardado en la base de datos.

Condiciones: Haber pasado por parámetro un usuario y una solicitud.

Camino: 1-2-4

Caso de prueba: Registrar solicitud

Entrada: Recibe una solicitud nula o no válida, o un usuario inválido.

Resultado: Se chequea y como la solicitud no es válida o el usuario tampoco lo es devuelve **null.**

Condiciones: Haber pasado por parámetro una solicitud y un usuario.

Se le realizó además la prueba de caja blanca a los métodos **Obtener planificación semanal** de la clase CitaFacadeImpl del módulo común del paquete útil y a la porción de código correspondiente al rasgo **Registrar un ciudadano**, perteneciente a la clase CiudadanoFacadeImpl, del componente ciudadano, dentro del módulo común el paquete Identidad, presentadas en el **Anexo 6.**

3.4.3 Método de prueba de caja negra

A la mayor parte de los usuarios de programas extensos no les interesa los detalles de su funcionamiento; lo único que desean es conseguir respuestas. Es decir, desean tratar el programa como una caja negra a la cual le introducen datos de entrada y obtienen de ella los datos de salida que esperan. De ahí el nombre de este método. (28)

Para comprobar la funcionalidad de los servicios ofrecidos por los componentes

desarrollados se realizaron pruebas de caja negra sobre el sistema donde se examinó el correcto funcionamiento del módulo común exponiendo los servicios comunes utilizados por la capa de procesos.

También se pudo comprobar mediante estas pruebas como respondió el software ante situaciones en las cuales se le entraban datos incorrectos y se le pedía hacer algo para lo cual no fue programado, obteniendo resultados satisfactorios. Referenciar los casos de prueba del proyecto.

3.4.3.1 No conformidades halladas

No conformidad se aquella desviación o incumplimiento de los requisitos especificados. Las no conformidades se realizaron utilizando la base de datos del Sistema de Emisión de Pasaportes Diplomáticos conjuntamente la aplicación de escritorio desarrollada en el proyecto.

Este grafico se hace con la intención de ser más explicativos en cuanto a los errores encontrados en las pruebas que se realizaron en el sistema concerniente a los servicios comunes del negocio del mismo, con el objetivo de facilitar el progreso de las revisiones y evidenciar la terminación de las acciones correctivas.

Para ver el registro de no conformidades detectadas consultar **anexo 7**

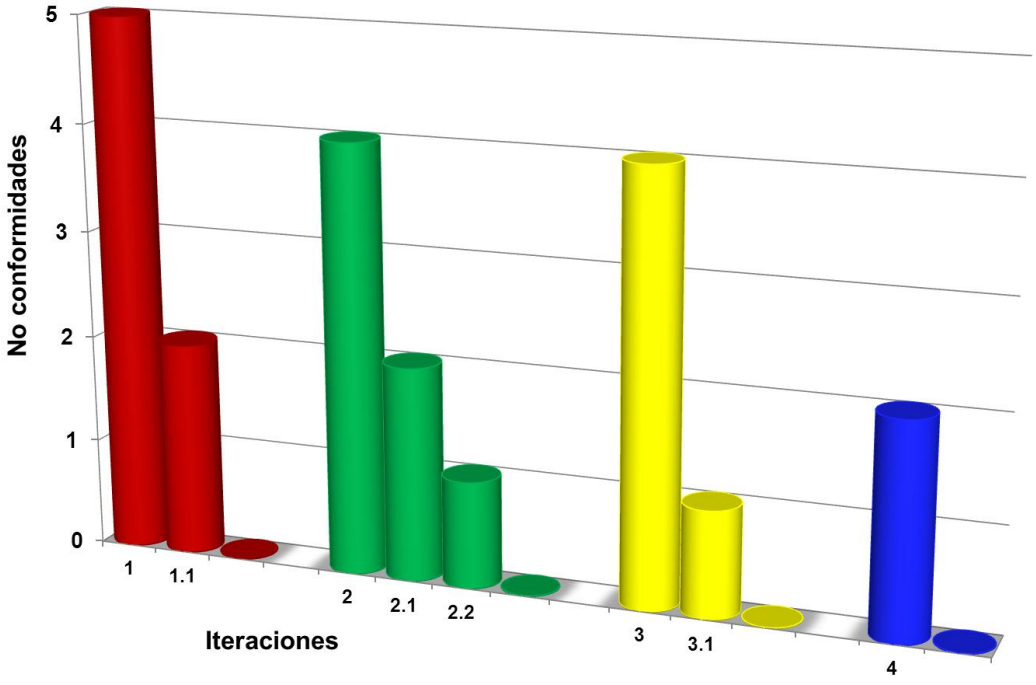

Figura 16 No conformidades detectadas en las pruebas de caja negra realizadas sobre el sistema.

3.4.3.2 Usabilidad

Usabilidad se define en el estándar ISO 9241 como “el grado en el que un producto puede ser utilizado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un determinado contexto de uso”. La usabilidad no puede definirse como un atributo simple de un sistema. (18)

La usabilidad aumenta con el uso de gráficos simples y comprensibles, métodos estandarizados de formas y accesos, y otro tipo de facilidades, algunas imágenes y gráficos se colocaron en la parte superior izquierda ya que estudios científicos demuestran que el primer lugar donde ve el usuario según el patrón de modelo F. Permite el trabajo colectivo, a través de la participación de todos los miembros, además que tiende a tener una gran capacidad para ser recordado por el usuario y posee una facilidad de aprendizaje, no presenta errores catastróficos, mejorando considerablemente las condiciones de trabajo. (24)

3.4.4 Pruebas de integración

Las pruebas de integración buscan comprobar la mezcla de las distintas partes de la aplicación, para determinar si funcionan correctamente en conjunto. Se realizan después de concluir las pruebas unitarias.

La necesidad de realizar las pruebas de integración viene dada por el hecho de que los módulos que forman a un sistema suelen fallar cuando trabajan de forma conjunta, aunque previamente se haya demostrado que funcionan correctamente de manera individual.

Para realizar dicha prueba se escoge una porción de código que incluye alguna llamada a otro método perteneciente a diferente módulo, se le realiza la prueba de caja blanca y además se comprueba mediante la herramienta *JUnit*, demostrando así la integración exitosa entre componentes.

3.4.4.1 Herramienta JUnit

Marco de trabajo para pruebas unitarias creado por Erich Gamma y Kent Beck. Herramienta de código abierto que se ha convertido en el estándar para las pruebas unitarias en *Java* y que es soportado por la mayoría de los Entorno de Desarrollo Integrado (IDEs), como *Eclipse* o *Net-Beans*.

Tiene Licencia Pública Común (CPL), que permite usarlo de forma gratuita para cualquier aplicación, sea esta comercial o de código abierto. Posee una comunidad mucho mayor que el resto de los marcos de trabajo de pruebas en *Java*.

Consiste en un conjunto de clases, fácil de usar y aprender que les permite a los programadores escribir sus pruebas unitarias en el lenguaje *Java*.

A continuación se muestra una prueba realizada con *JUnit* al componente NomencladorFacade, obteniendo como resultado el éxito de la prueba.

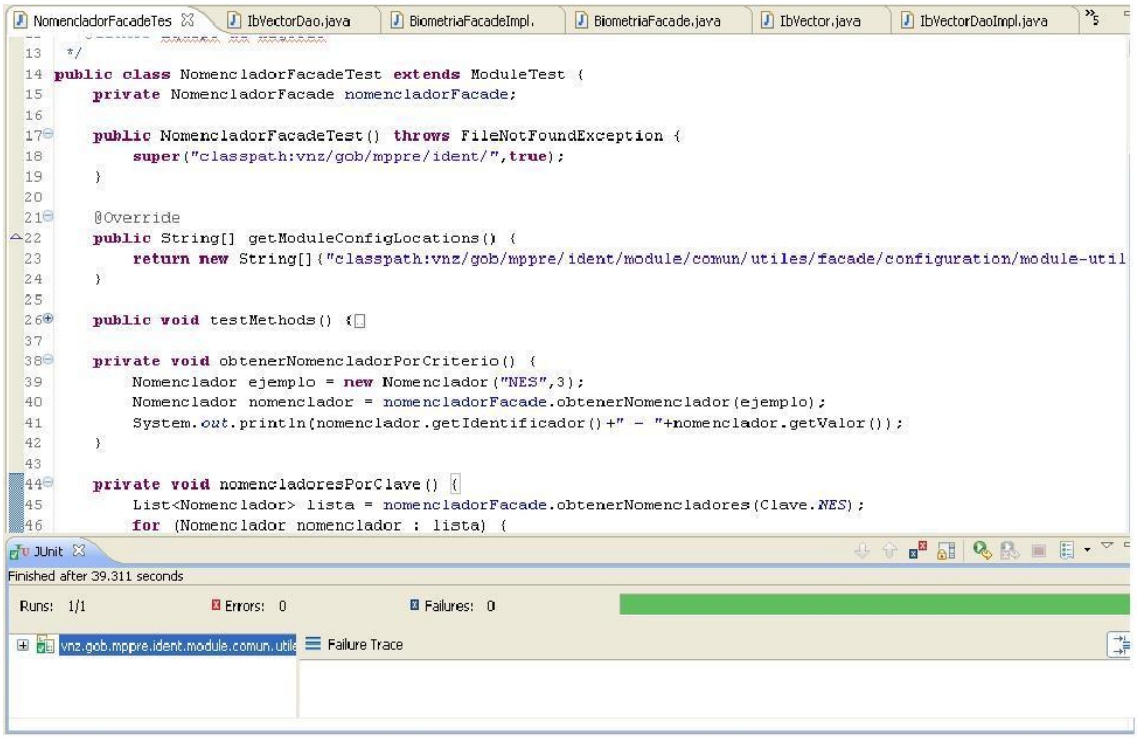

Figura 17 Prueba *JUnit* satisfactoria

3.4.5 Nivel de acoplamiento

Se ha hecho una buena descomposición del sistema, obteniendo así la menor cantidad de conexiones posibles entre los módulos por lo que habrá menos oportunidad de que aparezca el efecto onda (un defecto de un módulo, puede aparecer afectando a otro). Logrando tener la posibilidad de cambiar un módulo con el mínimo riesgo de tener que cambiar otro, se trata de que cada cambio realizado afecte lo menos posible a otros módulos. Mientras se esté manteniendo un módulo, no hay necesita de preocuparse en los detalles internos (código) de cualquier otro módulo.

Dependiendo del número de parámetros que se intercambian para la comunicación se clasifica el nivel de acoplamiento del módulo como acoplamiento normal, dentro del cual se ve mejor definido en el acoplamiento de datos donde se establece la comunicación básica por medio de elementos. Ejemplo:

Figura 18 Ejemplo de acoplamiento de datos

Este nivel de acoplamiento se considera como el mejor nivel de acoplamiento que puede existir dentro de un sistema informático.

3.4.5.1 Reutilización de código

La reutilización de código se refiere al comportamiento y a las técnicas que garantizan que una parte o la totalidad de un programa informático existente se puedan emplear en la construcción de otro programa. De esta forma se aprovecha el trabajo anterior, se economiza tiempo, y se reduce la redundancia dejando el código reusable en un único lugar, y llamándolo desde otros programas.

Según la comprobación estadística que se le realizó a los componentes que exponen los servicios comunes para evaluar el nivel de reutilización de código que existe se pudo percibir como los componentes se utilizaban más de una vez en todos los casos en uno de ellos fue utilizado por todos los módulos existentes.

		COMPONENTES					
	Submódulos	Administración	Biometría	Identidad	Solicitud	Trámite	Útiles
MÓDULOS	Acreditación – Solicitud	X		X	X		X
	Acreditación – Enrolamiento		X	X		X	X
	Acreditación – Común						X
	Pasaporte – Solicitud	X		X	X		X
	Pasaporte – Enrolamiento		X	X		X	X
	Pasaporte – Común						X
	Externos – MPPRE			X			X
	Externos – SAIME			X			X
	Total de reutilización	**2**	**2**	**6**	**2**	**2**	**8**

Tabla 6 Reutilización de componentes.

3.5 Conclusiones

En este capítulo la realización del diagrama de clases del diseño apoyó la implementación del módulo común. De las pruebas unitarias a los componentes que exponen los servicios con la utilización de la herramienta *JUnit* y las pruebas de caja negra permitieron comprobar que los componentes funcionan correctamente, exponiendo los servicios requeridos por los procesos de negocio, evidenciándose la calidad del producto final. Por lo que queda probada la hipótesis planteada en la investigación.

Conclusiones Generales

Con el análisis que se realizó a los elementos teóricos, técnica y tecnologías, además del ambiente de desarrollo propuesto por el proyecto se llegó a la conclusión de utilizar la metodología y herramientas que se consideraron más idóneas para el desarrollo de los servicios comunes. Luego de efectuar el estudio de los procesos de negocio se realizó una propuesta de solución basada en los rasgos funcionales identificados a partir del modelo conceptual de dominio elaborado. Además se definió la estructura de los componentes y el diagrama de clases del diseño de los mismos.

Con el desarrollo de este módulo se comprobó la gran importancia dentro del sistema puesto que logró la definitiva interoperabilidad e integración de los servicios necesarios aportando beneficios a la arquitectura y permitió unificar diversos servicios disminuyendo considerablemente el tamaño del proyecto. Se concibió un modelo de pruebas que permitió conocer de algunas no conformidades viendo la necesidad de realizar varias iteraciones de pruebas para perfeccionar el sistema validando así la hipótesis planteada.

Recomendaciones

Se recomienda continuar con el desarrollo de los componentes en base a la posible incorporación en un futuro de nuevos requisitos o cambios que puedan surgir en los ya existentes. Además se propone aplicar este tipo de soluciones en otras áreas relacionadas con procesos de identificación, emisión de documentos, y en sistemas con arquitecturas complejas y modelo de dominio extenso.

Referencias Bibliográficas

1. **Ramiro Machaca Honorio.** Scribd.com. [En línea] 2008. [Citado el: 10 de febrero de 2011.] http://www.scribd.com/doc/20966946/1095.

2. Acoplamiento. [En línea] http://www.ongei.gob.pe/publica/metodologias/Lib5081/CAP0641.HTM.

3. ADO.Net Entity Framework. [En línea] http://msdn.microsoft.com/es-es/library/bb399572.aspx.

4. Definicion.de. [En línea] http://definicion.de.

5. **Proyecto Pasaporte Diplomático.** *Estandar de Codificación.* La Habana : s.n., 2011.

6. **Dirección Provincial Cultura, Matanzas, Cuba.** LA ENSEÑANZA DEL SOFTWARE LIBRE EN CUBA, RETOS Y PERSPECTIVAS. [En línea] abril de 2009. http://www.atenas.cult.cu/?q=node/6880.

7. **Alejandro Aguilar Sierra.** Las Metodologías Ágiles en la Enseñanza de la Ingeniería de Software. [En línea] sptiembre de 2003. http://www.programacionextrema.org/ponencias/enc03pres.pdf.

8. Modelado de sistema con UML. [En línea] http://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/doc-modelado-sistemas-UML/multiple-html/x219.html.

9. Técnicas de Pruebas. [En línea] [Citado el: 25 de enero de 2011.] http://indalog.ual.es/mtorres/LP/Prueba.pdf.

10. WorkFlow-Tecnología para la Integración y Orquestación de Procesos . [En línea] http://personales.alumno.upv.es/~roferdi1/trabajoweb/1TextoIntroduccion.htm.

11. **Buschamann, F., y otros, y otros.** *Pattern-Oriented Software Architecture.* Inglaterra : s.n., 1996.

12. **Cerami, Ethan.** *Web Servics Essential.* 2002.

13. **Chapell, David y Jewell, Tyler.** Java Web Services. [En línea] marzo de 2002. http://std.kku.ac.th/4830401462/Ebooks/webservice/O%27Reilly%20-%20Java%20Web%20Services%20%282002%29.pdf.

14. **Cristián, Maximiliano.** Introduccion a la Arquitectura de Software. [En línea] 2007. http://www.willydev.net/InsiteCreation/v1.0/willycrawler/2008.05.01.articulo.introduccion%20a%20la%20arquitectura%20de%20soft.pdf.

15. **de la Torre LLorente, Cesar, y otros, y otros.** Guia de Arquitecura N-Capas orientada al

dominio con .Net. [En línea] 2010. http://es.scribd.com/doc/39222105/Guia-Arquitectura-N-Capas-DDD-NET-4-Borrador-Marzo-2010-SinPass#ad_unit=Doc_Sideboard_MediumRectangle_BTF_300x250&url=http%3A//es.scribd.com/doc/39222105/Guia-Arquitectura-N-Capas-DDD-NET-4-Borrador-Marzo-2010-SinPass%23a.

16. **Díaz, J.F.** Metodos de prueba de programas. [En línea] http://www.galeon.com/neoprogramadores/met_test.htm.

17. **Eloi.** MCV-Modelo Vista Controlador. [En línea] abril de 2007. http://www.programacionweb.net/articulos/articulo/?num=505.

18. **Ferré Grau, Xavier.** Principios Básicos de Usabilidad para Ingenieros Software. [En línea] http://is.ls.fi.upm.es/xavier/papers/usabilidad.pdf.

19. **Fortunato Branford, Melba y Vega de la Cruz, Erick.** Desarrollo de componentes de negocio para el piloto emisión de pasaportes. La Habana : s.n., 2010.

20. **Guineas de Salas, Alejandro.** Arquitectura SOA para la integración entre software libre y software propetario en entronos mixtos. . [En línea] 2007. http://www.sigte.udg.es/jornadassiglibre2007/comun/1pdf/13.pdf.

21. **Hernández Sampieri, Roberto, Fernández Collado, Carlos y Baptista Lucio, Pilar.** *Metodología de la Investigación.* D.F. : Programas Educativos S.A. de C.V., 1998.

22. **Huanca, Daniel.** Tipos de prueba. *Pruebas de caja blanca.* [En línea] [Citado el: 15 de 02 de 2011.] http://herrorsoft.zxq.net/pruebacajablanca.html.

23. **Jiménez Sanzano, Fidel .** SISTEMA DE GESTIÓN DE CAPITAL HUMANO PARA EL SISTEMA GESTIÓN INTEGRAL CEDRUX. [En línea] http://www.informaticahabana.cu/node/762.

24. **Nielsen, Jakob .** F-Shaped Pattern For Reading Web Content. [En línea] abril de 2006. http://www.useit.com/alertbox/reading_pattern.html.

25. **Prof Soto, Lauro .** Diseño de la Arquitectura de software. *Tecnologico .* [En línea] http://www.mitecnologico.com/Main/Dise%F1oArquitecturaDelSoftware.

26. **Shaw, M. y Garlan, D.** *Introduction to software Architectures. New perspectives.* Prentice Hall : s.n., 1996.

27. **Thompson, Ivan.** Promonegocios.net. [En línea] agosto de 2006. http://www.promonegocios.net/mercadotecnia-servicios/definicion-servicios.html.

28. **Torres, Manolo.** Prueba.pdf. [En línea] http://indalog.ual.es/mtorres/LP/Prueba.pdf.

Bibliografía Consultada

Acoplamiento. [En línea] http://www.ongei.gob.pe/publica/metodologias/Lib5081/CAP0641.HTM.

Definicion.de. [En línea] http://definicion.de.

Proyecto Pasaporte Diplomático. *Estandar de Codificación.* La Habana : s.n., 2011.

Alejandro Aguilar Sierra. Las Metodologías Ágiles en la Enseñanza de la Ingeniería de Software. [En línea] sptiembre de 2003. http://www.programacionextrema.org/ponencias/enc03pres.pdf.

Modelado de sistema con UML. [En línea] http://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/doc-modelado-sistemas-UML/multiple-html/x219.html.

Técnicas de Pruebas. [En línea] [Citado el: 25 de enero de 2011.] http://indalog.ual.es/mtorres/LP/Prueba.pdf.

Buschamann, F., y otros, y otros. *Pattern-Oriented Software Architecture.* Inglaterra : s.n., 1996.

Díaz, J.F. Metodos de prueba de programas. [En línea] http://www.galeon.com/neoprogramadores/met_test.htm.

Fortunato Branford, Melba y Vega de la Cruz, Erick. Desarrollo de componentes de negocio para el piloto emisión de pasaportes. La Habana : s.n., 2010.

Hernández Sampieri, Roberto, Fernández Collado, Carlos y Baptista Lucio, Pilar. *Metodología de la Investigación.* D.F. : Programas Educativos S.A. de C.V., 1998.

Huanca, Daniel. Tipos de prueba. *Pruebas de caja blanca.* [En línea] [Citado el: 15 de 02 de 2011.] http://herrorsoft.zxq.net/pruebacajablanca.html.

Nielsen, Jakob . F-Shaped Pattern For Reading Web Content. [En línea] abril de 2006. http://www.useit.com/alertbox/reading_pattern.html.

Torres, Manolo. Prueba.pdf. [En línea] http://indalog.ual.es/mtorres/LP/Prueba.pdf.

Calisoft. *Acta de Liberación_App_PD firmada.* La Habana : s.n., 2011.

Calisoft. *Pruebas de liberación de pasaporte Diplomático.* La Habana : s.n., 2011.

Glosario de términos

Arquitectura: Organización de los diversos elementos constitutivos de un sistema informático.

AFIS: Sistema automático de identificación de huellas digitales.

Componentes: Todo aquel recurso desarrollado para un fin concreto y que puede formar solo, o junto con otros, un entorno funcional requerido por cualquier proceso predefinido.

Framework: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

Hardware: Corresponde a todas las partes físicas y tangibles de una computadora.

JBoss: Es un servidor de aplicaciones J2EE de código abierto implementado en Java puro.

JBPM: Es un motor de flujo de trabajo escrito en Java que puede ejecutar procesos descritos en su proceso de lenguaje de definición JPDL.

Paradigma: Es un modelo o patrón en cualquier disciplina científica.

Procesos del negocio: Ordenación lógicamente interrelacionada de tareas desarrolladas en tiempo y espacio (con comienzo y fin, con entradas y salidas definidas) y que se orienta al logro de un objetivo de negocio, generando un output de valor (total o parcial) para el cliente del proceso.

SAIME: Servicio Autónomo de Identificación, Migración y Extranjería

Software: Se refiere al equipamiento lógico o soporte lógico de una computadora digital.

UML: Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modelling Language) es el lenguaje de modelado de sistemas de software más conocido en la actualidad.

Workflow: Un Flujo de Trabajo constituye un esquema de tareas (simples o complejas) definidas.

Anexos

Anexo 1 Operacionalización de las variables

VARIABLES	INDICADORES	UNIDAD DE MEDIDA
Servicios comunes	Desarrollo	Alto Medio Bajo
	Integración	Buena Normal Mala
Aceptación por el usuario	Usabilidad	Alta Media Baja
Nivel de acoplamiento	Efecto onda	Alta Media Baja
	Dependencia	Alta Media Baja

Tabla 7 Operacionalización de variables.

Anexo 2 Modelos de los procesos del negocio

1. Proceso de Emisión de pasaportes diplomáticos y de servicio.

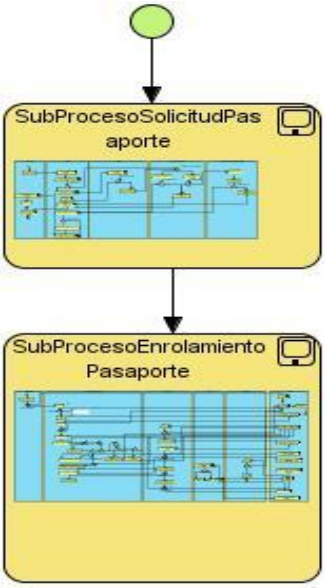

Figura 19 Modelo del proceso de emisión de pasaporte diplomáticos y de servicio.

2. Proceso de Emisión de acreditaciones.

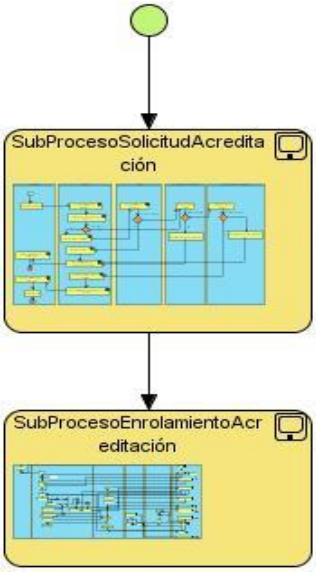

Figura 20 Proceso de emisión de acreditaciones.

Anexo 3 Modelos de los subprocesos del negocio

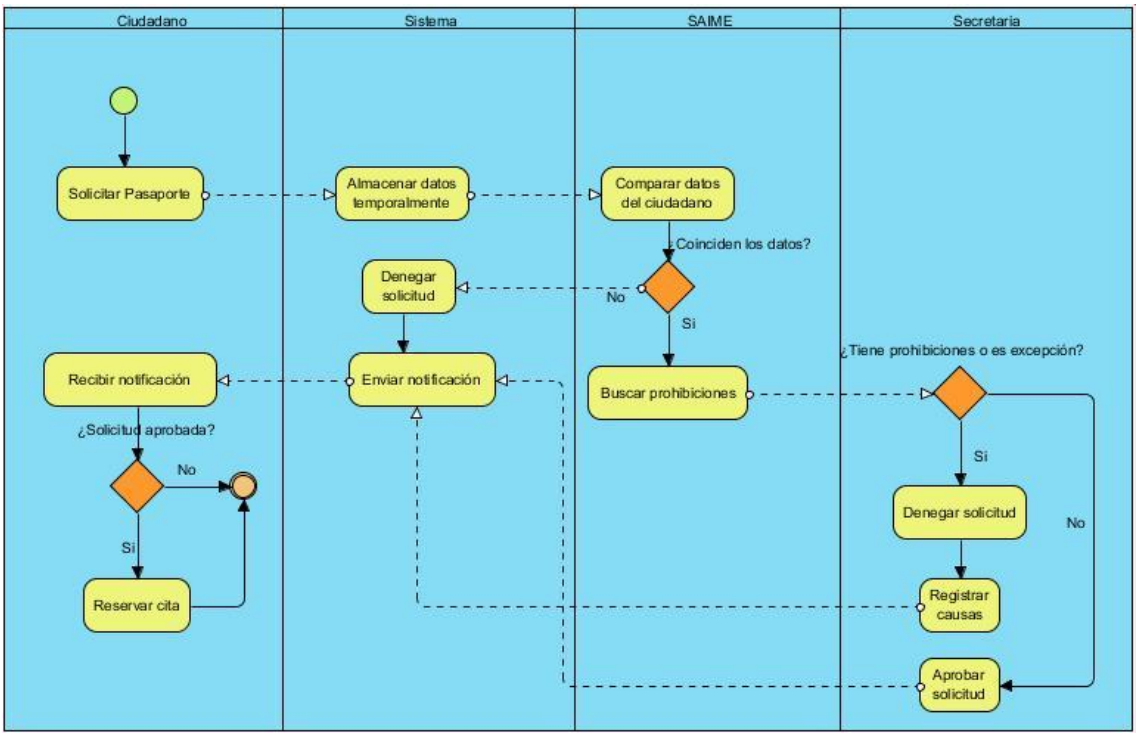

Figura 21 Modelo del subproceso solicitud de pasaportes.

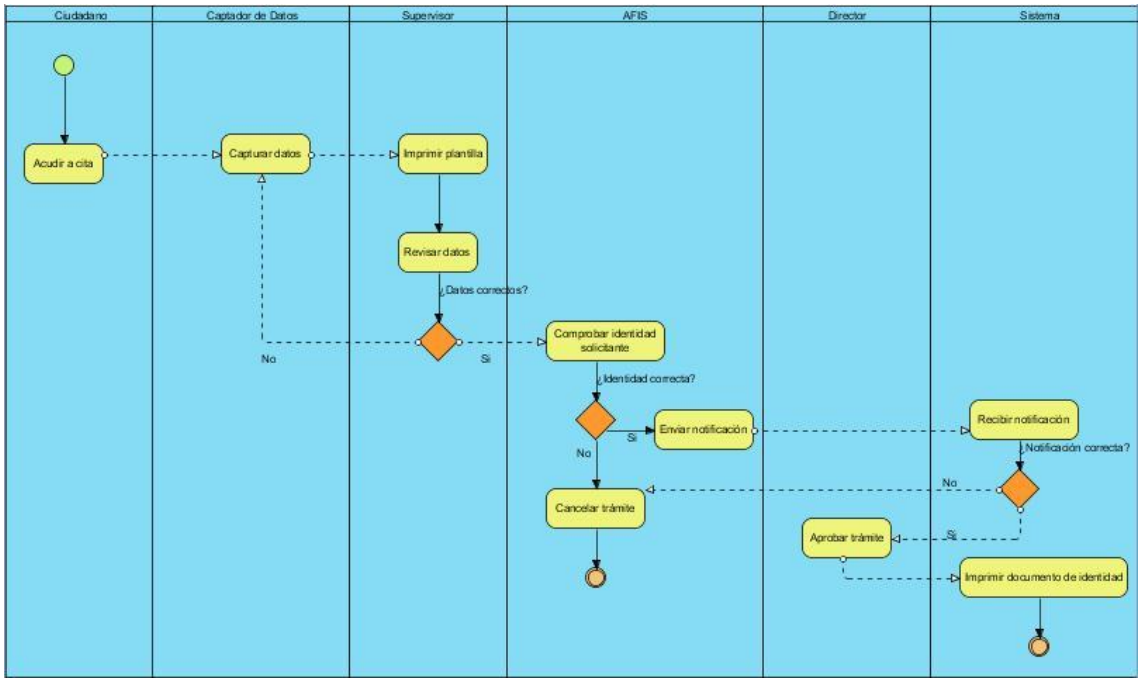

Figura 22 Modelo del subproceso enrolamiento de pasaportes.

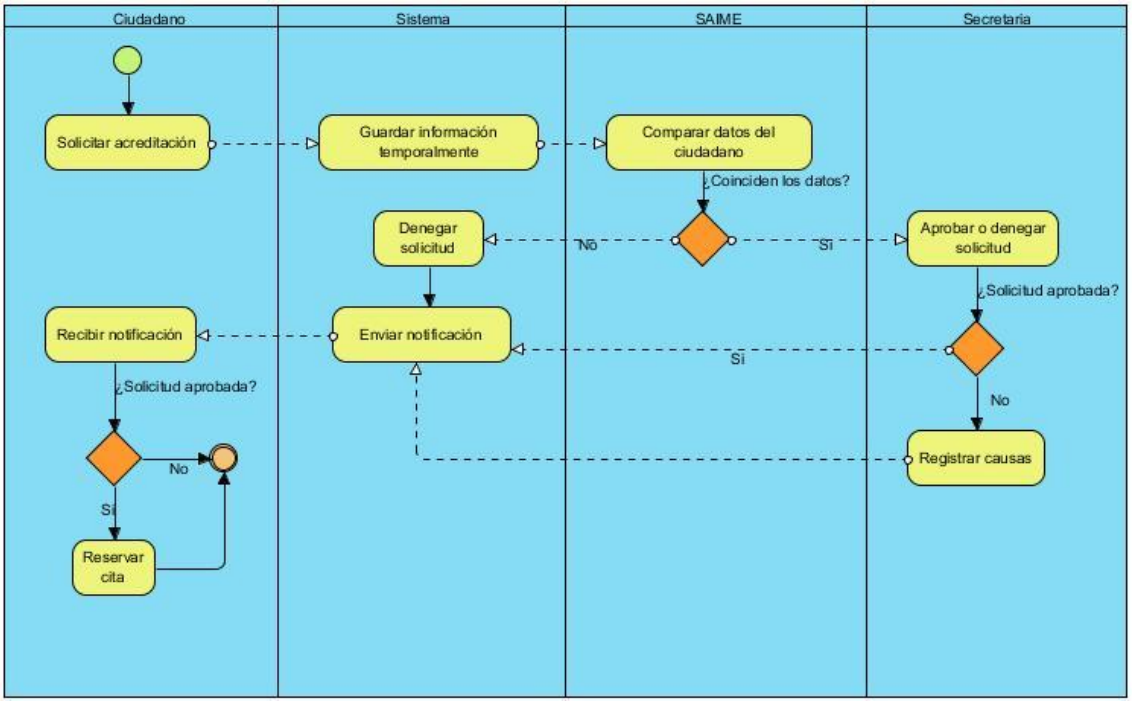

Figura 23 Modelo del subproceso solicitud de acreditación.

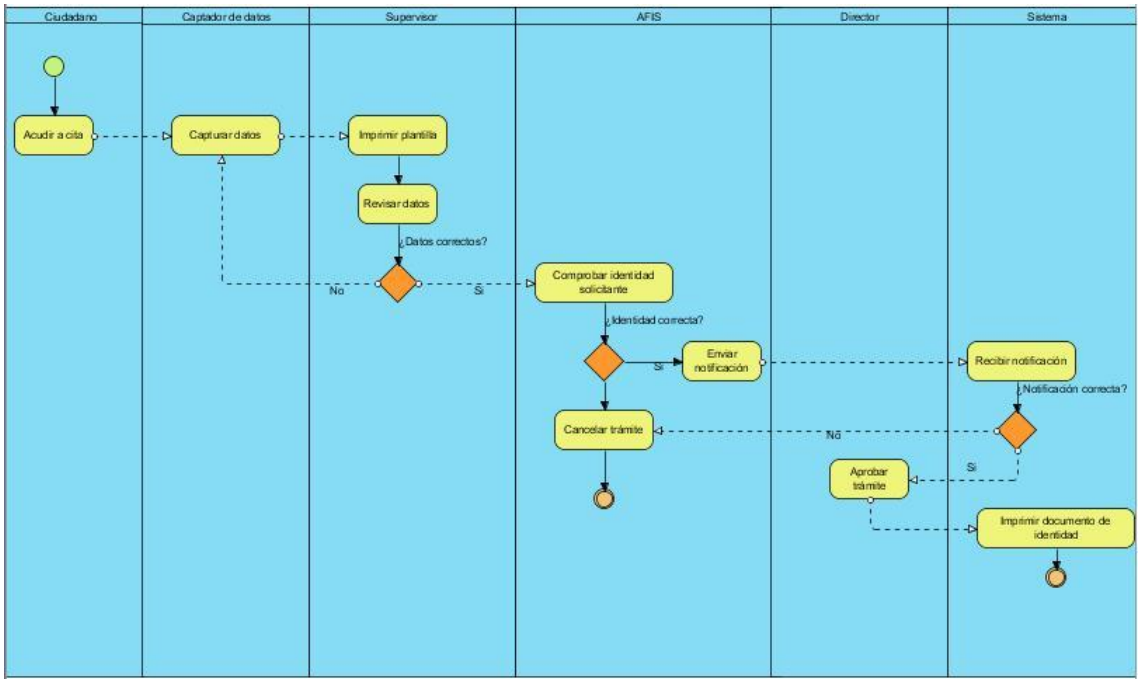

Figura 24 Modelo del subproceso de enrolamiento de acreditación.

Anexo 4 Planeación de las iteraciones por rasgos.

Iteración 2

GESTIONAR TRÁMITES.

Rasgo	Modelo General	Diseño	Construcción	Prueba
Registrar un nuevo trámite.	31/01/2011	1/02/2011	3/02/2011	9/02/2011
Registrar recaudos.	31/01/2011	1/02/2011	4/02/2011	9/02/2011
Modificar datos de un trámite.	31/01/2011	2/02/2011	7/02/2011	10/02/2011
Cancelar un trámite.	31/01/2011	2/02/2011	8/02/2011	10/02/2011

Tabla 8 Planeación del rasgo gestionar tramites.

GESTIONAR CITAS

Rasgo	Modelo General	Diseño	Construcción	Prueba
Registrar datos de una nueva cita	29/01/2011	10/02/2011	11/02/2011	16/02/2011
Modificar datos de una cita	29/01/2011	10/02/2011	15/02/2011	16/02/2011
Eliminar cita	29/01/2011	10/02/2011	15/02/2011	16/02/2011

Tabla 9 Planeación del rasgo gestionar cita

Iteración 3

GESTIONAR DATOS DE UN USUARIO

Rasgo	Modelo General	Diseño	Construcción	Prueba
Registrar un nuevo usuario.	17/02/2011	22/02/2011	24/02/2011	1/03/2011
Modificar datos de un usuario.	17/02/2011	22/02/2011	27/02/2011	1/03/2011

Eliminar un usuario.	17/02/2011	23/02/2011	27/02/2011	2/03/2011
Registrar características físicas.	17/02/2011	23/02/2011	28/02/2011	2/03/2011

Tabla 10 Planeación del rasgo gestionar datos de un usuario.

ENVIAR NOTIFICACIÓN

Rasgo	Modelo General	Diseño	Construcción	Prueba
Enviar notificación de aprobación de trámites.	18/02/2011	3/03/2011	4/03/2011	7/03/2011
Enviar notificación de denegación de una solicitud.	18/02/2011	3/03/2011	4/03/2011	7/03/2011

Tabla 11 Planeación del rasgo enviar notificación.

Iteración 4

AUTENTICAR USUARIO

Rasgo	Modelo General	Diseño	Construcción	Prueba
Adicionar un nuevo usuario	8/03/2011	10/03/2011	14/03/2011	17/03/2011
Modificar un usuario	8/03/2011	10/03/2011	15/03/2011	17/03/2011
Eliminar un usuario	8/03/2011	10/03/2011	15/03/2011	18/03/2011
Adicionar una contraseña.	8/03/2011	11/03/2011	16/03/2011	18/03/2011
Permitir el cambio de la contraseña.	8/03/2011	11/03/2011	16/03/2011	21/03/2011

Tabla 12 Planeación del rasgo autenticar usuario.

GESTIONAR ROL

Rasgo	Modelo General	Diseño	Construcción	Prueba

Adicionar un nuevo rol.	9/03/2011	22/03/2011	23/03/2011	25/03/2011
Modificar un rol.	9/03/2011	22/03/2011	24/03/2011	26/03/2011
Eliminar rol.	9/03/2011	22/03/2011	24/03/2011	26/03/2011

Tabla 13 Planeación del rasgo gestionar rol.

Anexo 5 Diseño de los componentes.

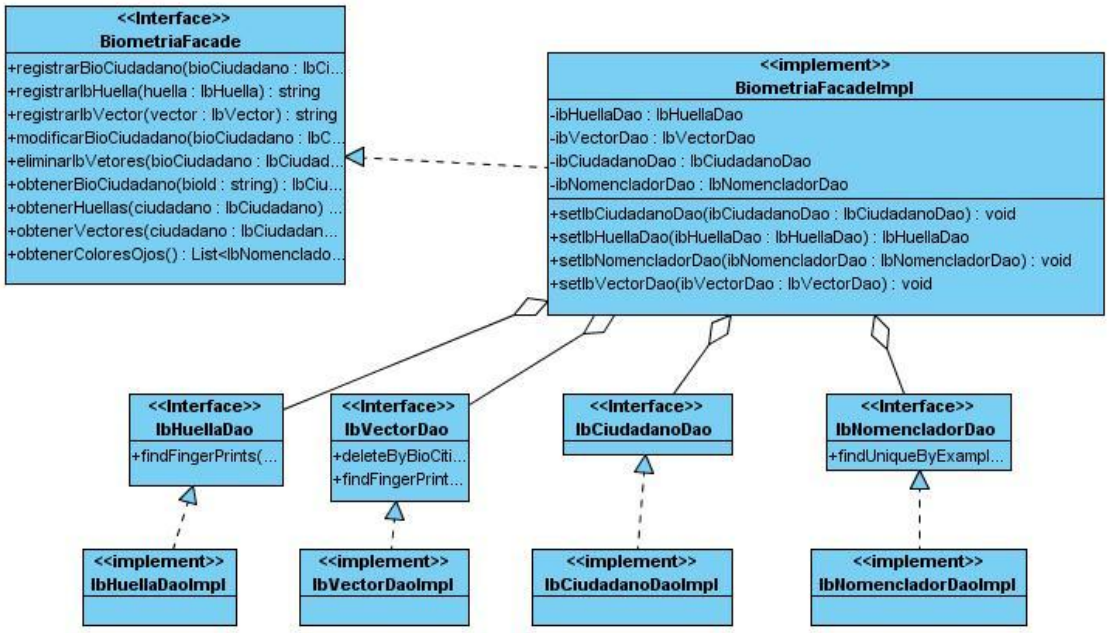

Figura 25 Diagrama de clases del diseño del componente de Biometría

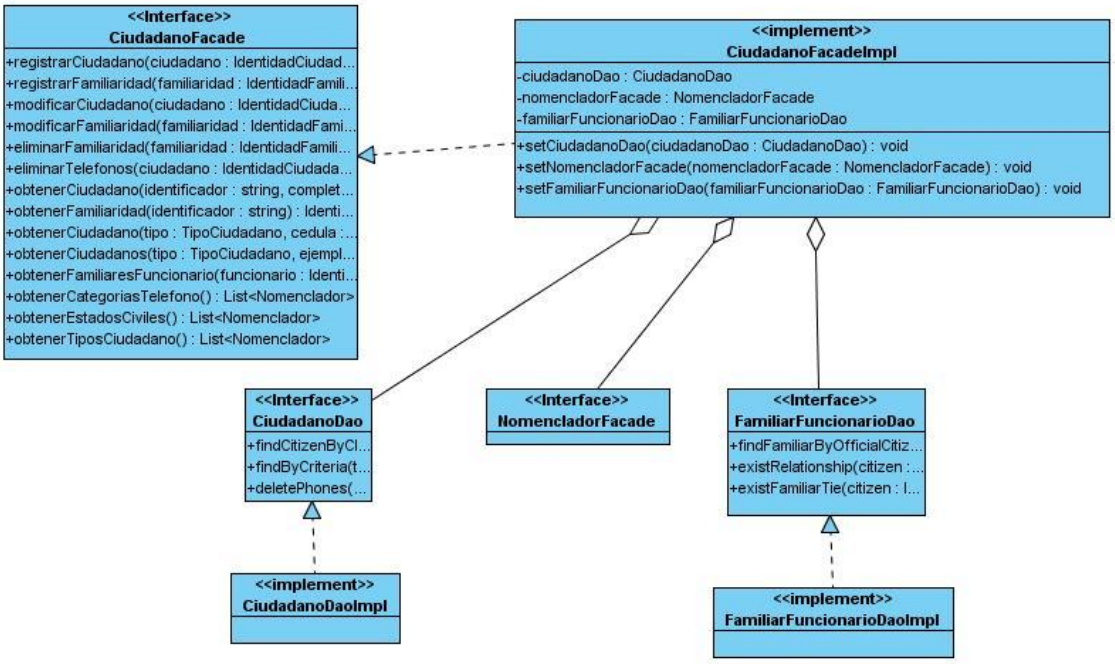

Figura 26 Diagrama de clases del diseño del componente de Ciudadano

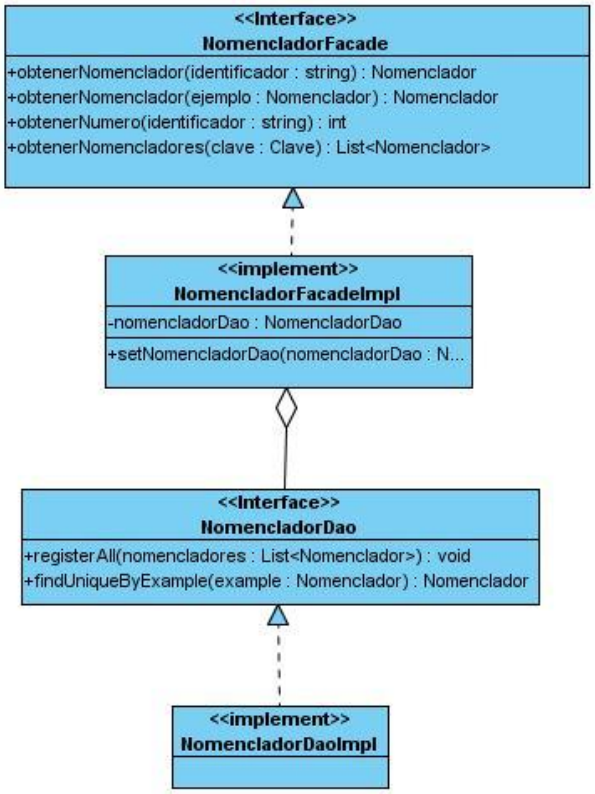

Figura 27 Diagrama de clases del diseño del componente nomenclador

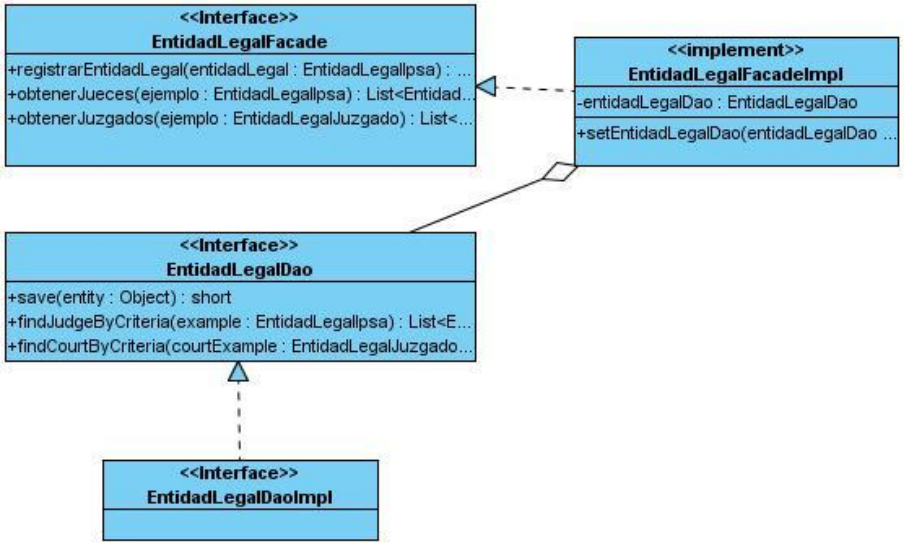

Figura 28 Diagrama de clases del diseño del componente entidad legal.

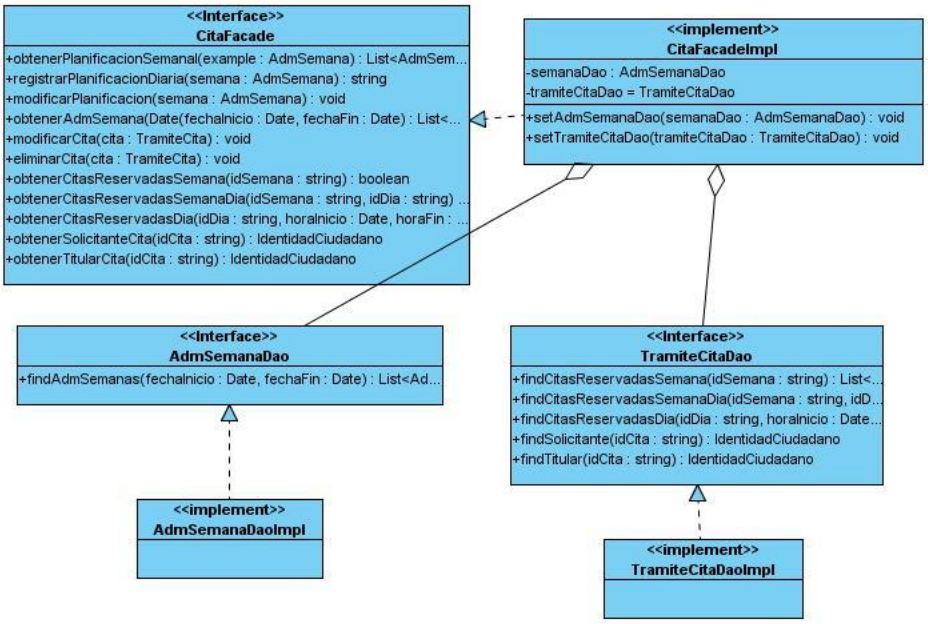

Figura 29 Diagrama de clases del diseño del componente cita

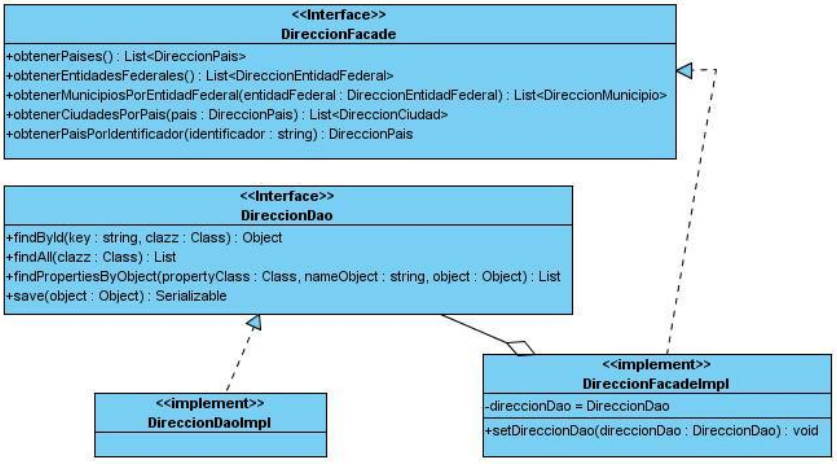

Figura 30 Diagrama de clases del diseño del componente dirección.

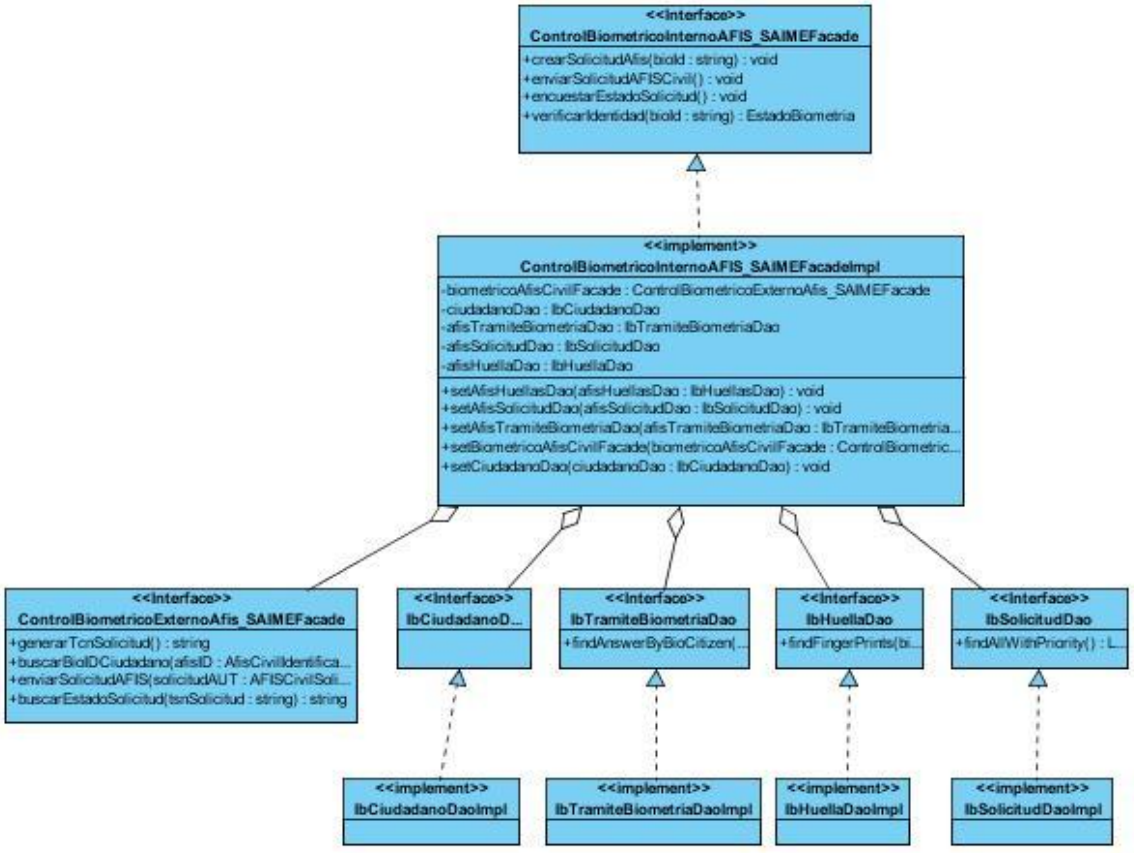

Figura 31 Diagrama de clases del diseño del componente control biométrico interno AFIS_SAIME.

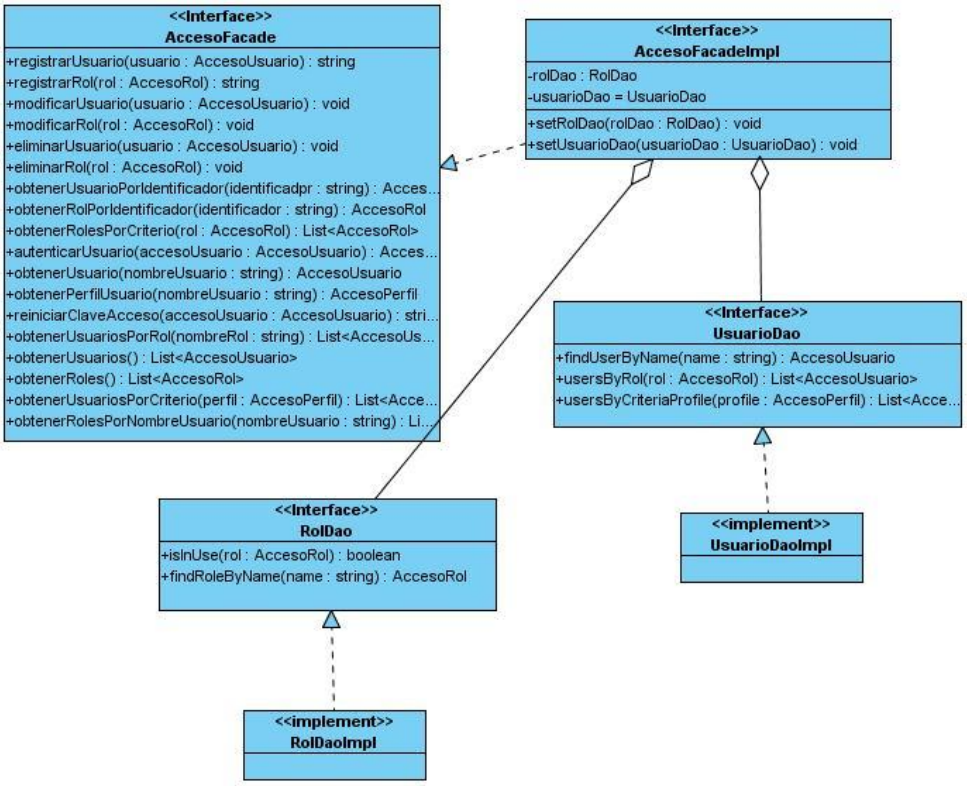

Figura 32 Diagrama de clases del diseño del componente acceso legal.

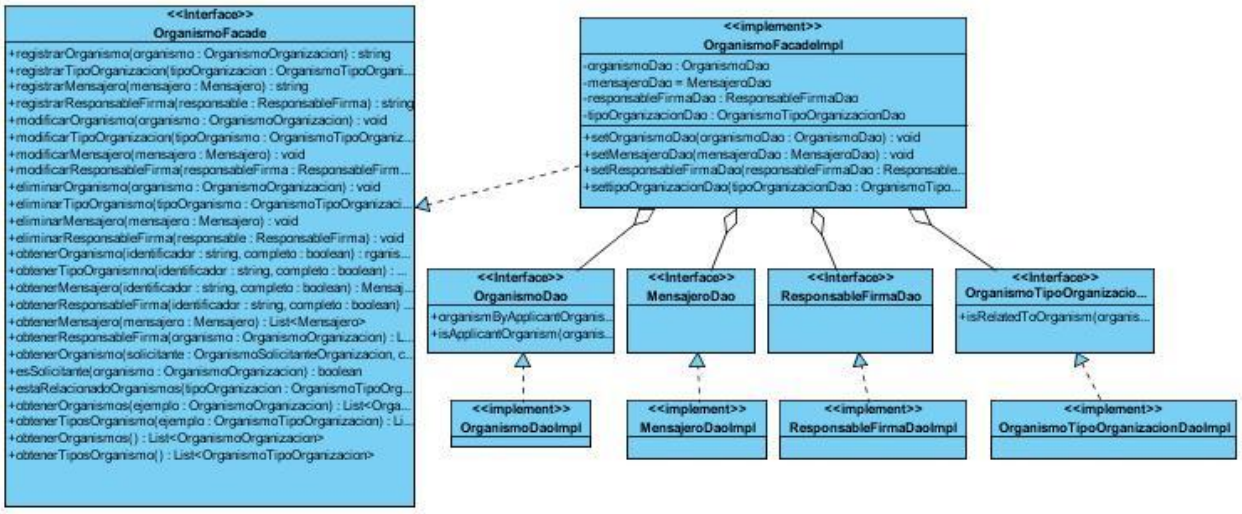

Figura 33 Diagrama de clases del diseño del componente organismo.

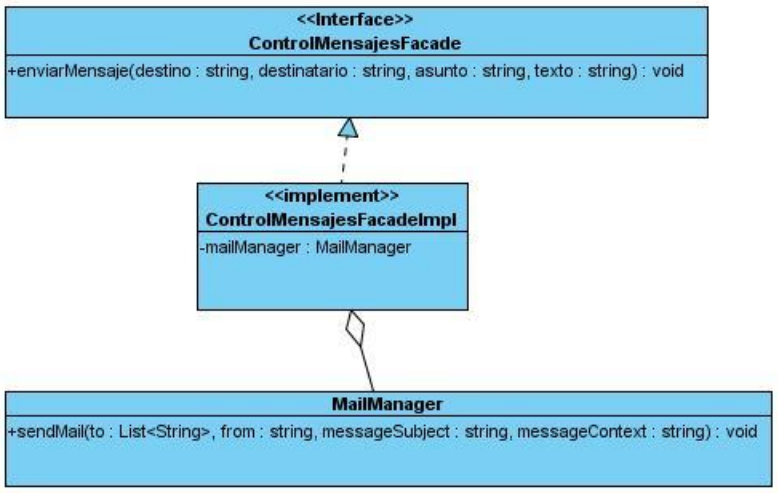

Figura 34 Diagrama de clases del diseño del componente control de mensajes.

Anexo 6 Descripción de los componentes.

Componente		
Nombre	**Tipo**	**Descripción**
CiudadanoFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con los ciudadanos.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
registrarCiudadano	String	Registra los datos del ciudadano, retorna el identificador con el que quedo registrado en la base de datos.
registrarFamiliaridad	String	Registra los datos de la familiaridad, retorna el identificador con el que quedo registrado en la base de datos.
modificarCiudadano	void	Modifica (actualiza) los datos del ciudadano especificado.
modificarFamiliaridad	void	Modifica (actualiza) la familiaridad entre dos funcionarios especificados.
eliminarFamiliaridad	void	Elimina una familiaridad específica de un ciudadano.
eliminarTelefonos	boolean	Elimina los teléfonos de un ciudadano específico.
obtenerCiudadano	IdentidadCiudadano	Retorna el ciudadano al que le pertenece en la base de datos el identificador especificado. Se especifica el modo de recuperación que se quiere hacer del objeto.
obtenerFamiliaridad	IdentidadFamiliarFuncionario	Retorna la familiaridad al que le pertenece el familiar en la base de datos el identificador especificado.

obtenerCiudadano	IdentidadCiudadano	Retorna el ciudadano al que le pertenece en la base de datos. Se especifica el modo de recuperación que se quiere hacer del objeto.
obtenerCiudadanos	List<IdentidadCiudadano>	Devuelve un listado con todos los ciudadanos cuyos atributos son iguales a los atributos contenidos dentro del especificado.
obtenerFamiliaresFuncionario	List<IdentidadFamiliarFuncionario>	Retorna un listado con los familiares pertenecientes al funcionario especificado.
obtenerCategoriasTelefono	List<Nomenclador>	Devuelve un listado con todos los nomencladores "Categoría teléfono".
obtenerEstadosCiviles	List<Nomenclador>	Devuelve un listado con todos los nomencladores "Estado civil".

Tabla 14 Descripción de la clase CiudadanoFacade

Componente		
Nombre	**Tipo**	**Descripción**
BiometriaFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con los datos biométricos de un ciudadano.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
registrarBioCiudadano	String	Registra la información biométrica de un ciudadano retorna el identificador con el que quedo registrado en la base de datos.
registrarIbHuella	String	Registra la huella digital de un ciudadano retorna el identificador con el que quedo registrado en la base de datos.
registrarIbVector	String	Registra los datos de las huellas de los ciudadanos
modificarBioCiudadano	void	Modifica o cambia la información biométrica de un ciudadano.
eliminarIbVetores	boolean	Elimina los datos de las huellas de los ciudadanos
obtenerBioCiudadano	IbCiudadano	Se obtiene el objeto que pertenece a la información biométrica del ciudadano.
obtenerHuellas	List<IbHuella>	Devuelve una lista de objetos de tipo huellas.
obtenerVectores	List<IbVector>	Obtiene los datos de las huellas de los ciudadanos
obtenerColoresOjos	List<IbNomenclador>	Devuelve una lista con todos los tipos de color que pueden tener los ojos.
obtenerColoresCabello	List<IbNomenclador>	Devuelve una lista con todos los tipos de color que pueden tener el cabello.

Tabla 15 Descripción de la clase BiometriaFacade

Componente		
Nombre	**Tipo**	**Descripción**
NomencladorFacade	Interfaz	Interfaz es la encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con los nomencladores.
Atributos		

Nombre	Tipo

Funciones		
Nombre	**Retorno**	**Descripción**
obtenerNomenclador	Nomenclador	Devuelve el objeto Nomenclador al que pertenece en la base de datos el identificador especificado por el cual se quiere buscar al nomenclador.
obtenerNomenclador	List<Nomenclador>	Devuelve una lista de nomencladores que tenga la misma clave que la pasada por parámetros.
obtenerNomenclador	Nomenclador	Devuelve el nomenclador que contenga los valores especificados por el ejemplo, los valores nulos no se tomaran como punto de comparación y solos los valores atómicos serán comparables.

Tabla 16 Descripción de la clase NomencladorFacade

Componente		
Nombre	**Tipo**	**Descripción**
NotificacionFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con las notificaciones.

Atributos	
Nombre	**Tipo**

Funciones		
Nombre	**Retorno**	**Descripción**
registrarNotificacion	String	Registra la notificación y devuelve el identificador.

Tabla 17 Descripción de la clase NotificacionFacade

Componente		
Nombre	**Tipo**	**Descripción**
EstadoCiudadanoFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con los ciudadanos (la parte de Estado).

Atributos		
Nombre	**Tipo**	**Descripción**

Funciones		
Nombre	**Retorno**	**Descripción**
modificarCiudadano	void	Modifica (actualiza) los datos del ciudadano especificado.
eliminarCiudadano	void	Elimina los datos del ciudadano especificado.
obtenerCiudadano	EstadoIdentidadCiudadano	Retorna el ciudadano al que le pertenece en la base de datos el identificador especificado. Se especifica el modo de recuperación que se quiere hacer del objeto.
obtenerCiudadanoPorIdentificador	EstadoIdentidadCiudadano	Retorna el ciudadano al que le pertenece en la base de datos el identificador especificado. Se especifica el modo de recuperación que se quiere hacer del objeto.

Tabla 18 Descripción de la clase EstadoCiudadanoFacade

Componente

Nombre	Tipo	Descripción
EntidadLegalFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con las entidades legales.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
obtenerJueces	List<EntidadLegalIpsa>	Retorna un listado de todos los jueces cuyos datos coinciden con el ejemplo especificado.
obtenerJuzgados	List<EntidadLegalJuzgado>	Retorna un listado de todos los juzgados.

Tabla 19 Descripción de la clase EntidadLegalFacade

Componente		
Nombre	**Tipo**	**Descripción**
CitaFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con las citas.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
obtenerPlanificacionSemanal	List<AdmSemana>	Se obtiene una lista de la planificación semanal.
registrarPlanificacionDiaria	String	Se registra la planificaron del día y retorna el identificador con el que quedo registrado en la base de datos.
modificarPlanificacion	void	Modifica (actualiza) los datos de las citas de una semana.
obtenerAdmSemana	List<AdmSemana>	Buscar todas las citas que su fecha de inicio allá sido realizado en esa fecha o posteriormente
modificarCita	void	Modifica (actualiza) los datos de una cita.
eliminarCita	void	Elimina los datos de una cita especificada.
obtenerCitasReservadasSemana	boolean	Especifica si existe cita en la semana especificada.
obtenerCitasReservadasSemanaDia	List<TramiteCita>	Devuelve un listado de las citas del día especificado.
obtenerCitasReservadasDia	List<TramiteCita>	Devuelve un listado de las citas entre fechas especificando además si son de acreditaciones o pasaporte.
obtenerSolicitanteCita	IdentidadCiudadano	Devuelve la identidad del ciudadano que solicito la cita.
obtenerTitularCita	IdentidadCiudadano	Devuelve la identidad del titular de la cita.

Tabla 20 Descripción de la clase CitaFacade

Componente		
Nombre	**Tipo**	**Descripción**

DireccionFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con las direcciones (Países, Estados, Ciudades, Municipios).
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
obtenerPaises	List<DireccionPais>	Devuelve una lista con todos los países existentes en la base de datos
obtenerEntidadesFederales	List<DireccionEntidadFederal>	Devuelve una lista con todos las direcciones de entidades federales existentes en la base de datos
obtenerMunicipiosPorEntidadFederal	List<DireccionMunicipio>	Devuelve una lista con todos los municipios existentes en la base de datos de acuerdo a la entidad federal
obtenerCiudadesPorPais	List<DireccionCiudad>	Devuelve una lista con todos las ciudades existentes en la base de datos de acuerdo al país especificado
obtenerPaisPorIdentificador	DireccionPais	Devuelve el objeto de tipo país de acuerdo al identificador especificado

Tabla 21 Descripción de la clase DireccionFacade

Componente		
Nombre	**Tipo**	**Descripción**
ControlBiometricoInternoAFIS_SAIMEFacade	Interfaz	Interfaz encargada de lo correspondiente con el control biométrico interno con los sistemas AFIS y SAIME
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
crearSolicitudAfis	void	Crea un solicitud para luego enviarla al sistema AFIS
enviarSolicitudAFISCivil	void	Envía la solitud creada al sistema AFIS
encuestarEstadoSolicitud	void	Encuesta el estado de la solicitud
verificarIdentidad	EstadoBiometria	Verifica si la identidad del ciudadano especificado esta correcta.

Tabla 22 Descripción de la clase ControlBiometricoInternoAFIS_SAIMEFacade

Componente		
Nombre	**Tipo**	**Descripción**
AccesoFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con el acceso de los usuarios del sistema.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**

registrarUsuario	String	Registra la información de un usuario, retorna el identificador con el que quedo registrado en la base de datos.
registrarRol	String	Registra la información de un rol, retorna el identificador con el que quedo registrado en la base de datos.
modificarUsuario	void	Modifica (actualiza) los datos de un usuario específico.
modificarRol	void	Modifica (actualiza) los datos de un rol específico.
eliminarUsuario	void	Elimina los datos de un usuario específico.
eliminarRol	void	Elimina los datos de un rol específico.
obtenerUsuarioPorIdentificador	AccesoUsuario	Devuelve un usuario por el identificador especificado.
obtenerRolPorIdentificador	AccesoRol	Devuelve un rol por el identificador especificado
obtenerRolesPorCriterio	List<AccesoRol>	Devuelve un listado de roles especificando el criterio de búsqueda.
autenticarUsuario	AccesoUsuario	Permite autenticar un usuario en el sistema
obtenerUsuario	AccesoUsuario	Devuelve un usuario con el nombre especificado
obtenerPerfilUsuario	AccesoPerfil	Devuelve el perfil de un usuario con el nombre especificado
reiniciarClaveAcceso	String	Reinicia la clave de acceso de un usuario.
obtenerUsuariosPorRol	List<AccesoUsuario>	Devuelve un listado de usuario dado el nombre de rol
obtenerUsuarios	List<AccesoUsuario>	Devuelve el listado de todos los usuarios
obtenerRoles	List<AccesoRol>	Devuelve un listado con todos los roles.
obtenerUsuariosPorCriterio	List<AccesoUsuario>	Devuelve un listado de los usuarios dado un criterio dado.
obtenerRolesPorNombreUsuario	List<AccesoRol>	Devuelve un listado de los roles dado un criterio dado.

Tabla 23 Descripción de la clase AccesoFacade

Componente		
Nombre	**Tipo**	**Descripción**
OrganismoFacade	Interfaz	Interfaz encargada de la exposición de todas las funcionalidades correspondientes a todo lo concerniente con los organismos.
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
registrarOrganismo	String	Registra (guarda) los datos del organismo especificado, retorna el identificador con el que quedo registrado en la base de datos.
registrarTipoOrganizacion	String	Registra (guarda) los datos del tipo de organismo especificado, retorna el identificador con el que quedo registrado en la base de datos.
registrarMensajero	String	Registra (guarda) los datos del mensajero de organismo, retorna el identificador con el que quedo registrado en la base de datos.
registrarResponsableFirma	String	Registra (guarda) los datos del responsable de firma de organismo, retorna el identificador con el que quedo registrado en la base de datos.

modificarOrganismo	void	Modifica (actualiza) los datos del organismo especificado.
modificarTipoOrganizacion	void	Modifica (actualiza) los datos de un tipo de organismo especificado.
modificarMensajero	void	Modifica (actualiza) los datos de un mensajero de organismo especificado.
modificarResponsableFirma	void	Modifica (actualiza) los datos de un responsable de firma de un organismo especificado.
eliminarOrganismo	void	Elimina los datos del organismo especificado.
eliminarTipoOrganismo	void	Elimina los datos del tipo de organismo especificado.
eliminarMensajero	void	Elimina los datos de un mensajero de organismo especificado.
eliminarResponsableFirma	void	Elimina los datos de un responsable de firma especificado.
obtenerOrganismo	OrganismoOrganizacion	Retorna el organismo al que le pertenece en la base de datos el identificador especificado.
obtenerTipoOrganismno	OrganismoTipoOrganizacion	Retorna el tipo de organismo al que le pertenece en la base de datos el identificador especificado.
obtenerMensajero	Mensajero	Retorna el mensajero de un organismo al que le pertenece en la base de datos el identificador especificado.
obtenerResponsableFirma	ResponsableFirma	Retorna el responsable de firma al que le pertenece en la base de datos el identificador especificado.
obtenerMensajero	List<Mensajero>	Retorna un listado de mensajero de acuerdo al criterio especificado.
obtenerResponsableFirma	List<ResponsableFirma>	Retorna un listado de responsable de firma de acuerdo al criterio especificado.
obtenerOrganismo	OrganismoOrganizacion	Retorna un organismo de acuerdo a los criterios especificados
esSolicitante	boolean	Especifica si el organismo especificado es solicitante.
estaRelacionadoOrganismos	boolean	Especifica si los organismos están relacionados
obtenerOrganismos	List<OrganismoOrganizacion>	Retorna un listado de organizaciones de organismos de acuerdo con el criterio especificado
obtenerTiposOrganismo	List<OrganismoTipoOrganizacion>	Retorna un listado de tipos de organismos de acuerdo al criterio especificado
obtenerOrganismos	List<OrganismoOrganizacion>	Devuelve un listado con todos los organismos que están registrados en la base de datos.
obtenerTiposOrganismo	List<OrganismoTipoOrganizacion>	Devuelve un listado con todos los tipos de organismos que están registrados en la base de datos

Tabla 24 Descripción de la clase OrganismoFacade

Componente		
Nombre	**Tipo**	**Descripción**
ControlMensajesFacade	Interfaz	Interfaz encargada de lo correspondientes a los mensajes
Atributos		
Nombre	**Tipo**	**Descripción**
Funciones		
Nombre	**Retorno**	**Descripción**
enviarMensaje	void	Envía mensajes especificando el destinatario, el destino, el asunto.

Tabla 25 Descripción de la clase ControlMensajesFacade

Anexo 7 Prueba realizada con el método de caja blanca.

De acuerdo a la porción de código correspondiente al rasgo **Gestionar cita**, perteneciente a la clase CitaFacadeImpl, del componente Cita, dentro del módulo común el paquete útil, se le realizó la prueba de caja blanca.

```
public List<AdmSemana> obtenerPlanificacionSemanal(AdmSemana example) {        1
        List<AdmSemana> lista = null;                                           1
        if (example != null) {                                                  2
                lista = semanaDao.findByExample(example);                       3
        } else {
                lista = semanaDao.findAll();                                    4
        }
        return lista;                                                           5
    }
```

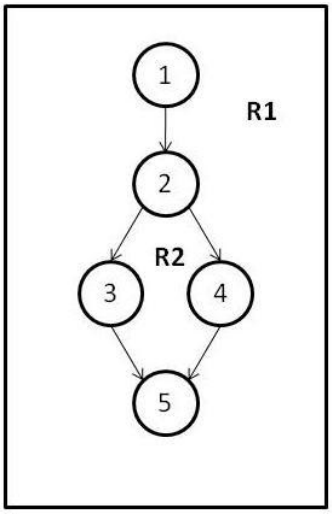

Figura 35 Grafo del caso de prueba Obtener planificación semanal

Complejidad Ciclomática:

V (G) = Aristas - Nodos + 2

V (G) = 5 – 5 + 2

V (G) = 2

Posibles caminos: 1-2-3-5, 1-2-4-5

Camino: 1-2-3-5

Caso de prueba: Obtener planificación semanal

Entrada: Recibe una semana.

Resultado: Se obtiene una lista con la planificación semanal

Condiciones: Haber pasado por parámetro una semana.

Camino: 1-2-3-5

Caso de prueba: Obtener planificación semanal

Entrada: Recibe una semana.

Resultado: Se obtiene una lista con toda la planificación

Condiciones: Haber pasado por parámetro una semana nula o no válida.

De acuerdo a la porción de código correspondiente al rasgo **Registrar un ciudadano**,

perteneciente a la clase CiudadanoFacadeImpl, del componente ciudadano, dentro del módulo común el paquete Identidad, se le realizó la prueba de caja blanca

```
public String registrarCiudadano(IdentidadCiudadano ciudadano) {                1
        if (ciudadano != null) {                                                 2
                if (ciudadano.getClasificador() == null) {                       3
                        this.generarClasificador(ciudadano);                     4
                }
                if (ciudadano.getTipoCiudadano() == null) {                      5
                        this.generarTipoCiudadano(ciudadano);                    6
                }
        }
        String identificador = ciudadanoDao.save(ciudadano);                     7
        return identificador;                                                    7
}
```

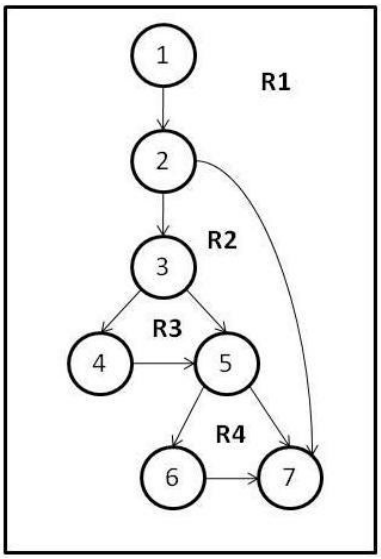

Figura 36 Grafo del caso de prueba Registrar Ciudadano

Complejidad Ciclomática:

V (G) = Aristas - Nodos + 2

V (G) = 9 – 7 + 2

V (G) = 4

Posibles caminos: 1-2-3-4-5-6-7, 1-2-3-5-6-7, 1-2-3-4-5-7, 1-2-7

Camino: 1-2-3-4-5-6-7

Caso de prueba: Registrar un ciudadano

Entrada: Recibe un ciudadano.

Resultado: Se adiciona un ciudadano a la base de datos devolviendo el identificador con el que fue registrado en la misma, asignándole un clasificador y organismo.

Condiciones: Haber pasado por parámetro un ciudadano.

Camino: 1-2-3-5-6-7

Caso de prueba: Registrar un ciudadano

Entrada: Recibe un ciudadano.

Resultado: Se adiciona un ciudadano a la base de datos devolviendo el identificador con el que fue registrado en la misma, asignándole organismo.

Condiciones: Haber pasado por parámetro un ciudadano.

Camino: 1-2-3-4-5-7

Caso de prueba: Registrar un ciudadano

Entrada: Recibe un ciudadano.

Resultado: Se adiciona un ciudadano a la base de datos devolviendo el identificador con el que fue registrado en la misma, asignándole el clasificador que no tiene.

Condiciones: Haber pasado por parámetro un ciudadano.

Camino: 1-2-7

Caso de prueba: Registrar un ciudadano

Entrada: Recibe un ciudadano nulo o no válido.

Resultado: Devuelve -1 como identificador

Condiciones: Haber pasado por parámetro un ciudadano.

Anexo 8 Registro de defectos y dificultades detectados mediante las pruebas de caja negra

Iteración 1

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Orden de las solicitudes	1	Las solicitudes deben aparecer ordenadas alfabéticamen-te.	Adicionar solicitud	Etapa de prueba	Media	Mostrar las solicitudes ordenadas alfabética-mente
Observaciones de las solicitudes	2	No adiciona las observaciones de las solicitudes	Adicionar solicitud	Etapa de prueba	Alta	Adicionar como parte de la solicitud las observacio-nes
Tipo de solicitud	3	No parecen los tipos de solicitudes	Adicionar solicitud	Etapa de prueba	Alta	Mostrar los tipos de solicitudes que existen
Botón de denegación de la solicitud	4	Aunque dice haber denegado la solicitud no aparece como denegada en la base de datos	Denegar solicitud	Etapa de prueba	Alta	Registrar la denegación de la solicitud correctamen-te
Obtener el identificar en la base de datos	2	No devuelve el identificador con el cual fue registrado en la base de datos	Adicionar solicitud	Etapa de prueba	Alta	Devolver el identificar con el cual fue guardado en la base de datos

Tabla 26 Registro de no conformidades detectadas en la iteración 1.

Iteración 1.1

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Interfaz de datos de la solicitud	1	Muestra la interfaz de registro de datos biométricos	Adicionar solicitud	Etapa de prueba 1.1	Alta	Mostrar la interfaz correspon-diente.
Página de solicitudes del sitio web	2	Muestra y permite modificar las solicitudes ya aprobadas	Aprobar una solicitud	Etapa de prueba 1.1	Alta	Las solicitudes aprobadas no deben ser mostradas

Tabla 27 Registro de no conformidades detectadas en la iteración 1.1.

Iteración 1.2

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Botón de denegación de la solicitud	1	No muestra el mensaje de confirmación de la denegación	Denegar solicitud	Etapa de prueba 1.2	Media	Mostrar mensaje de confirmación

Tabla 28 Registro de no conformidades detectadas en la iteración 1.2.

Iteración 2

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Botón de cancelar trámite	1	Lanza excepción al cancelar el trámite y en la base de datos no aparece cancelado	Cancelar tramite	Etapa de prueba	Alta	Cancelar el trámite correctamen-te en la base de datos

Interfaz recaudos de un menor	2	El recaudo acta de soltería no lo muestra en la interfaz correspondien-te	Realizar trámite	Etapa de prueba	Alta	Mostrar los recaudos correspon-diente con cada trámite.
Botón de reservar cita	3	No registra la cita como reservada	Registrar cita	Etapa de prueba	Alta	Mostrar ese día y hora como reservado para cita
Interfaz de recaudo de mayor celulado	4	No se muestra las interfaces de recaudos correspondien-te a un mayor celulado	Realizar trámite	Etapa de prueba	Alta	Mostrar interfaz correspon-diente

Tabla 29 Registro de no conformidades detectadas en la iteración 2.

Iteración 2.1

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Botón modificar trámite	1	Lo datos actualizados del trámite no se actualizan en la base de datos	Modificar trámite	Etapa de prueba 2.1	Alta	Actualizar los cambios correctamen-te en la base de datos
Interfaz recaudos de menor	2	Muestra el recaudo acta de matrimonio	Realizar trámite	Etapa de prueba 2.1	Alta	Mostrar los recaudos correspon-dientes.

Tabla 30 Registro de no conformidades detectadas en la iteración 2.1.

Iteración 2.2

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección	Importancia	Recomendación
Nombre de la enfermedad	1	Permite la entrada de caracteres extraños como $, %	Realizar trámite	Etapa de prueba 2.2	Alta	Validar la entrada de caracteres extraños

Tabla 31 Registro de no conformidades detectadas en la iteración 2.2.

Iteración 3

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
Excepción de validación del nombre del ciudadano.	1	La palabra ciudadano está escrito incorrectamen-te	Adicionar solicitud	Etapa de prueba	Media	Corregir la falta ortográfica
Validación del identificador del ciudadano	2	No se valida el identificador de un ciudadano	Adicionar solicitud	Etapa de prueba	Alta	Mostrar una excepción en caso que el identificador no sea…
Validación de la fecha de nacimiento	3	Acepta fechas de años futuros	Adicionar solicitud	Etapa de prueba	Alta	Mostrar una excepción impidiendo utilizar años futuros

Acompañante de viaje	4	Si el ciudadano con quien viaja no tiene cédula lanza una excepción	Adicionar ciudadano	Etapa de prueba	Alta	Permitir que el acompañan-te no tenga cédula

Tabla 32 Registro de no conformidades detectadas en la iteración 3.

Iteración 3.1

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
Recaudos de menor no cedulado	1	Muestra todos los recaudos para el titular menor no cedulado	Realizar trámite	Etapa de prueba 3.1	Alta	Mostrar los recaudos correspon-dientes a un menor no cedulado
Nacionalidad del ciudadano	2	Si la nacionalidad es venezolana permite que el ciudadano no tenga cédula	Adicionar ciudadano	Etapa de prueba 3.1	Alta	Si el ciudadano es venezolano tiene que tener cédula
Identificador nacional	3	Si el ciudadano es extranjero permite adicionarlo sin el identificador nacional	Adicionar ciudadano	Etapa de prueba 3.1	Alta	Cuando el ciudadano es extranjero tiene que tener identificador nacional

Tabla 33 Registro de no conformidades detectadas en la iteración 3.1.

Iteración 4

REGISTRO DE DEFECTOS Y DIFICULTADES DETECTADOS

Elemento	No	No conformidad	Aspecto correspondiente	Etapa de detección Código del CP	Importancia	Recomendación
Botón de acceso	1	Cuando no se introduce usuario y contraseña no muestra un mensaje de aviso	Acceso a la aplicación	Etapa de prueba	Alta	Mostrar una alerta de que tiene que entrar un usuario y una contraseña
Interfaz principal	2	Cuando es un usuario solicitante permite el acceso a los permisos de secretaria	Permisos del usuario solicitante	Etapa de prueba	Alta	Mostrar nada más que las funcionalida-des correspon-dientes al solicitante

Tabla 34 Registro de no conformidades detectadas en la iteración 4.