Universidad de las Ciencias Informáticas

Título: Desarrollo del módulo de control de inventario del subsistema de personalización del Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones.

Trabajo de Diploma para optar por el título de

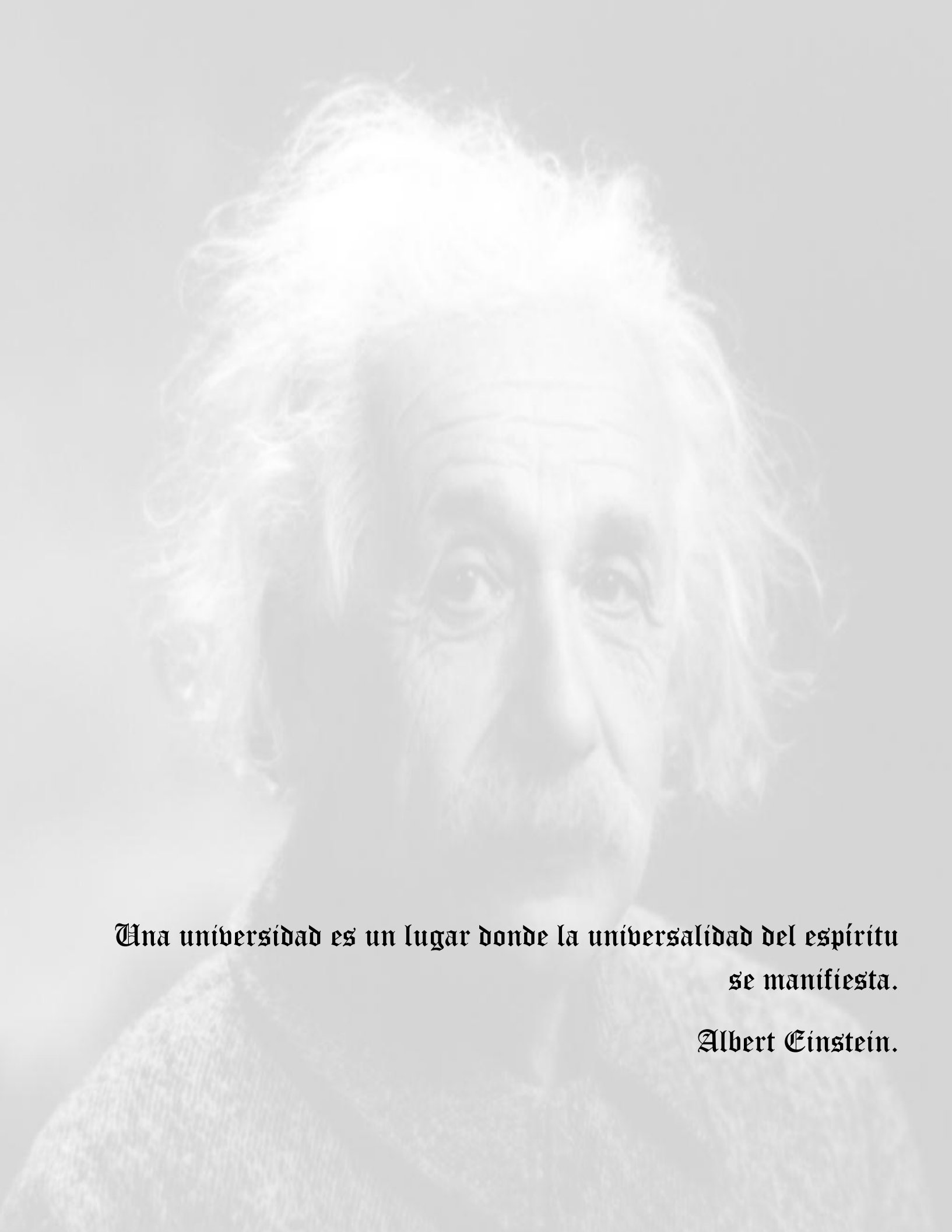
Ingeniero en Ciencias Informáticas

Autor: Ramón Alberto Hidalgo Reyes

Tutores: *Ing*. José Enrique Saura Guerra

Ing. Osmany Ramos Arias

La Habana, junio 2011



Una universidad es un lugar donde la universalidad del espíritu se manifiesta.

Albert Einstein.

Declaración de autoría

Declaro que soy el único autor del trabajo titulado: Desarrollo del módulo de control de inventario del subsistema de personalización del Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____
Ramón Alberto Hidalgo Reyes
(Autor)

_____
Ing. José Enrique Saura Guerra
(Tutor)

_____
Ing. Osmany Ramos Arias
(Tutor)

Datos de contacto

José Enrique Saura Guerra: Ingeniero en Ciencias Informáticas, instructor con 4 años de experiencia.

Correo electrónico: jsaura@uci.cu

Osmany Ramos Arias: Ingeniero en Ciencias Informáticas, instructor recién graduado con 1 año de experiencia.

Correo electrónico: oramos@uci.cu

Dedicatoria

Dedico este trabajo desde lo más profundo de mi corazón:

A mi mamá, que confió en mí desde el momento en que nací, que siempre me incitó a seguir adelante y nunca perdió su fe en mi. Quiero regalarte este momento por tanta dedicación y amor, espero que se sienta orgullosa por ver en lo que me convertido hoy.

A mami, mi abuelita Bella, a mis tías Marta y Mirtha, que han sido unas madres para mí, siempre dándome su amor incondicional, apoyándome y queriéndome en todo momento.

A mi abuelo Héctor, que aunque no camine por este mundo, siempre me quiso y apoyó.

A todos los que me quieren, me aprecian y están conmigo, les dedico este trabajo.

Agradecimientos

A Fidel y la Revolución, por haberme dado la posibilidad de estudiar en esta universidad donde me he formado como profesional.

A mi mamá, quien me dio la vida y me enseñó a querer, que me cuidó y me protegió desde el momento que supo de mi existencia, por cargarme tanto tiempo en su interior y por sus sonrisas para que no estuviese triste. Me quedaría sin palabras para explicar cuanto la quiero y le agradezco, me harían falta infinidades de vidas para describir un gesto de su corazón, pues de él solo brota luz, armonía y amor. Para tí mamá, lo tengo todo, y aun así no es suficiente. Te regalo este momento, así como me regalaste la vida.

A mami, mi abuelita del alma, cada arruga de su piel es una galaxia de amor, y el universo se queda pequeño para guardar todo el cariño, esmero y pasión que puso en mi corazón. Gracias por ayudarme a existir.

A mi tía Marta, quien me dormía y me sonreía, quien sembró tanto amor en mi vida. Si pudiera resumir en una palabra tu presencia en mi vida, sería fácil, "mamá", cuyo significado es infinito y su amor es invaluable.

A mi tía Mirtha, por soportar mis malacrianzas y velarme el sueño mientras crecía. Por su cariño incondicional, que aunque no sea mi mamá la veo como tal.

A mi primo Daykel, él ha sido el hermano que nunca tuve y el que siempre me ayudaba y acompañaba en cada locura y travesura de mi infancia.

A mi tío Fermín, por ayudarme todo este tiempo, que aunque no sea familia de sangre lo es de corazón.

A mi novia, que el amor de las novelas de Shakespeare no es comparable con el que me ha regalado. Gracias por existir y por formar parte de mi vida. Sin ti no hubiese logrado muchas cosas que tengo hoy, mua.

A mis tutores Osmany y Saura, que más que tutores se han convertido en grandes amigos, gracias por soportarme todo este tiempo.

A todas mis amistades que desde un comienzo sin proponérselo me enseñaron un poquito de cada uno y me permitieron compartir sus espacios durante estos 5 años, y hacer de esta universidad una estancia inolvidable.

A todos los que han contribuido a la realización de este trabajo de diploma, de corazón, muchas gracias.

Resumen

Mantener un adecuado control y organización sobre los materiales que circulan diariamente durante el proceso de personalización de documentos de identificación en el Ministerio del Poder Popular para Relaciones Exteriores es una tarea de suma importancia y de gran envergadura, ya que de su correcto funcionamiento depende la organización y administración de estos recursos. Dada la cantidad de registros que se conforman para la realización de estas actividades y lo complejo que se torna conocer la estructuración y ubicación de los documentos de identificación procesados, surge la necesidad de realizar un módulo que mantenga un inventario de todos los materiales que se utilizan para la emisión de los documentos de identificación en este ministerio.

La evasión de los problemas que conllevan a que la elaboración de productos, así como las tediosas búsquedas de información referente al desenvolvimiento de los materiales, provoca que sea necesario mantener un control de inventario. El principal propósito de dicho control es el de tener conocimiento instantáneo de datos cualitativos y cuantitativos, así como la localización de aquellos documentos que se emiten.

Con este trabajo se tiene como objetivo implementar y probar todo lo referente al control de inventario, así como los elementos necesarios para su total funcionamiento. Para realizar la implementación, se utilizaron herramientas libres para evitar el pago de costosas licencias, se utilizó Eclipse Galilieo como entorno de desarrollo, Java como lenguaje de programación y Apache Tomcat como contenedor de aplicaciones.

Palabras claves: Control de inventario, documentos de identificación, inventario.

Índice de tablas y figuras

Índice

Introducción

Las Tecnologías de la Información y las Comunicaciones (TICs) juegan un papel elemental en el manejo, control y preservación de la información. La transformación y sustitución de los viejos sistemas por los más novedosos, incluyendo adelantos tecnológicos ha traído como consecuencia que la información se gestione de una manera más efectiva y rápida. Las innovaciones tecnológicas ligadas a las metodologías de cada una de las empresas, incrementan la productividad, registro y administración de los medios y productos de una organización.

El éxito y el sustento de una organización se basan fundamentalmente en el control que esta pueda ejercer sobre su patrimonio. Para una mejor organización y seguimiento de los movimientos de cada recurso de la entidad se debe realizar un control de inventario, técnica que permite mantener la existencia de los bienes y pertenencias a un nivel deseado.

La necesidad del Ministerio del Poder Popular para Relaciones Exteriores (MPPRE) de la República Bolivariana de Venezuela de crear un sistema para la emisión de documentos de identificación que fortalezca la seguridad y reduzca el tiempo de ejecución de este proceso conllevó a la creación del proyecto Transformación y Modernización del Sistema de Emisión de Pasaportes Diplomáticos, de Servicio y Acreditaciones en el Centro de Identificación y Seguridad Digital.

Este proyecto forma parte de la solución de *software* del Contrato de Desarrollo de Solución Tecnológica Integral que consiste, específicamente, en la emisión de pasaportes diplomáticos y de servicio, así como las acreditaciones. Para que este proceso tenga el resultado esperado es necesario la utilización de recursos materiales tales como: documentos de identificación vírgenes[1], sobres para su protección y entrega, entre otros.

Actualmente, el MPPRE no cuenta con un sistema que le posibilite llevar el control de todos los recursos materiales que serán utilizados durante el proceso de personalización de documentos de identificación ya que solo se lleva el registro de aquellos pasaportes que son emitidos el cual se conforma y actualiza de forma manual, provocando esto retrasos al consultar cualquier información, además de inferirle poca

[1] Referente a documentos de identificación pre impresos.

confiabilidad a la misma ya sea por errores de escritura o información no legible. No se conoce la ubicación exacta de ningún documento de identificación por lo que se hace más tediosa su búsqueda. No existe asignación personas para el manejo y control de los materiales, ocasionando que sea poco probable conocer el responsable ante cualquier faltante de materiales o falla durante el proceso. Por todo lo anteriormente expuesto surge el siguiente **problema científico:** ¿Cómo lograr un control de los recursos materiales usados por el subsistema de personalización para el Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones?

El **objeto de estudio** consiste en el proceso de control de inventario cuyo **campo de acción** se enmarca en el proceso de control de inventario para el subsistema de personalización del Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones.

Como **objetivo general** se plantea desarrollar el módulo de control de inventario del subsistema de personalización derivándose los siguientes **objetivos específicos:**

- Efectuar el estudio de sistemas homólogos relacionados con el control de inventario.
- Efectuar el estudio herramientas para el diseño y desarrollo de la solución propuesta.
- Completar el análisis y diseño propuesto por el analista.
- Realizar la implementación de la capa de componentes del módulo de control de inventario.
- Realizar la implementación de la capa de proceso del módulo de control de inventario.
- Probar el funcionamiento del módulo mediante el uso de casos de prueba.

Para enfocar la investigación se plantea la siguiente **hipótesis:**

Si se desarrolla un módulo que realice un control de inventario se logrará controlar los recursos materiales usados por el subsistema de personalización del Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones (SEPDSA).

De la hipótesis antes mencionada se plantean las siguientes variables de la investigación:

Variable independiente (causa): Desarrollo del módulo de control de inventario.

Variable dependiente (efecto): Administrar los recursos materiales manejados por el subsistema de personalización del Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones.

La operacionalización de las variables se encuentra recogida en el **Anexo 1**.

Modelo metodológico

Estrategia de investigación:

La estrategia que se utiliza es la descriptiva, ya que se cuenta con la información suficiente como para plantear una hipótesis a nivel descriptivo.

Para garantizar el desarrollo de la investigación se hizo necesario el empleo de los siguientes métodos investigativos:

Métodos teóricos:

- **Analítico – sintético:** Facilita el entendimiento del fenómeno en el que se trabaja, es más útil la división de este en diferentes fases, y de esta forma descubrir sus características generales, lo que ayuda a seguir una correcta investigación.

  Este método se utilizó para entender y conocer las características del control de inventario y hacer una profundización de las mismas, dígase objetivos o deficiencias, así como también las funcionalidades para lograr resultados eficientes.

- **Histórico – lógico:** Permite estudiar de forma analítica la trayectoria histórica real de los fenómenos, evolución y desarrollo.

  Se utiliza este método para analizar la evolución de los sistemas existentes de control de inventario en empresas.

Métodos empíricos:

- **Entrevista:** Es una conversación planificada entre el investigador y el entrevistado para obtener información.

  Se realizan entrevistas con el objetivo de obtener la mayor información sobre la estructura y funcionalidades de la solución propuesta.

- **Experimento:** El experimento es el método empírico para el estudio de un objeto en el cual el investigador crea las condiciones o adapta las existentes para el esclarecimiento de las propiedades, leyes y relaciones del objeto, para verificar una hipótesis, una teoría o un modelo. El experimento puede ser transformador o comprobador.

  Se utiliza este método ya que uno de los objetivos propuestos fue el de garantizar el correcto funcionamiento de la propuesta de solución, por lo que es necesario la realización de pruebas para dar cumplimiento a dicho objetivo.

La **estructura del documento** está conformada por los siguientes capítulos:

Capítulo 1. Fundamentación teórica: Este capítulo contiene una base teórica para entender el problema planteado. Se describen los conceptos fundamentales para el dominio del problema, así como las tendencias, técnicas, tecnologías, metodologías y *software* usados. Además se incluye un estado del arte del tema a nivel nacional e internacional.

Capítulo 2. Descripción y análisis de la solución propuesta: Se presenta un estudio del funcionamiento y la información que se maneja para desarrollar el módulo, así como los artefactos necesarios para el completamiento del análisis y diseño propuesto por el analista, además se muestran los requerimientos de la propuesta de solución, así como los requerimientos no funcionales que debe cumplir.

Capítulo 3. Implementación y prueba: Se describe la implementación y las técnicas empleadas en la propuesta de solución, además, se describen los diagramas de componentes, así como las pruebas de caja blanca, unitarias y de integración aplicadas al módulo desarrollado.

Capítulo 1: Fundamentación teórica

1.1. Introducción

La informática ha alcanzado un lugar privilegiado en los diferentes sectores de la sociedad, desempeñando un papel protagónico en esta gran revolución de avance tecnológico. Insertándose en cada sector social, posibilitando así la aplicación de la misma a diferentes problemas reales como el control de recursos de entidades, organizaciones e individuos. En el presente capítulo se abordan temas relacionados con estudios de diferentes sistemas de control de inventario, así como metodologías y herramientas que facilitan el trabajo. También se aclaran conceptos y definiciones necesarias para el entendimiento del problema.

1.2. Conceptos asociados al dominio del sistema

Para la total comprensión de este trabajo se deben tener en cuenta una serie de definiciones y conceptos relacionados con el ambiente del trabajo, comenzando por saber que es un **inventario**.

Un **inventario** “es una provisión de materiales y de subcomponentes que tiene por objeto facilitar la producción o satisfacer la demanda de los clientes. Por lo general, los inventarios incluyen materia prima, productos en proceso y artículos terminados.” [1] También unificando varios conceptos se puede llegar a conclusiones y definir un inventario como una lista, relación o estado detallado de los bienes y pertenencias de una persona, organismo o entidad, por consiguiente, un **control de inventario** sería la “técnica que permite mantener la existencia de productos a los niveles deseado” [2], en otras palabras, se encarga de regular en forma óptima las existencias en los almacenes de herramientas, materias primas, productos en proceso y terminados, etc., protegiendo a la empresa de costos innecesarios por acumulamiento o falta de existencias en el almacén. En este caso, el control de inventario se realizaría a los materiales relacionados con el proceso de personalización de documentos de identificación.

Los **documentos de identificación o de identidad** “son los documentos que emite el estado para posibilitar la identificación personal de cada ciudadano” [3]. Todas las personas deben tener un documento de identidad que acredite quién es y que le permita acceder a los servicios estatales o de otro tipo. El proceso de emisión de este documento requiere un alto nivel de control y seguridad basados en

normas legislativas para que presente un valor legal. Los documentos de identificación a que se refiere en este trabajo son los pasaportes.

Una **acreditación** es un documento que provee a una persona de la facultad para desempeñar un rol o misión específica.

Un **pasaporte** es “un tipo de documento que se requiere para viajar internacionalmente y cruzar fronteras entre diferentes tipos de países o regiones” [4]. También puede verse como un documento legal expedido por las autoridades para autorizar la salida e ingreso de las personas a una nación. El proceso de emisión es complejo y al tratarse de documentos que tienen efectos legales e identifican a una persona, este requiere un alto nivel de seguridad y control. Existen diferentes categorías de pasaportes, las concernientes a este trabajo de tesis son:

El **pasaporte diplomático** “es un documento de viaje y de identificación concedido a quienes desempeñan determinadas funciones o cargos de alta dignidad, de responsabilidad nacional en muchos de los casos cumplen misiones oficiales en el exterior ya sea Embajador, Cónsul, etc.”[5]. Entre las ventajas para los que dispongan de este documento según la convención de Viena para las Relaciones Internacionales, se destacan: la inviolabilidad de la propiedad, la excepción del pago de impuestos de cualquier índole, la imposibilidad de ser sancionado por las fuerzas policiales, entre otras.

El **pasaporte de servicio** es el documento entregado al personal de las representaciones diplomáticas tales como agentes diplomáticos (son los acreditados con este pasaporte y que gozan de estatuto diplomático), miembros del personal administrativo (gozan de ciertas inmunidades y privilegios aun no siendo acreditados como personal diplomático) y miembros del personal de servicio de una misión diplomática.

1.3. Inventario

Como ha sido definido anteriormente, un inventario viene siendo como una relación entre materiales y subcomponentes cuyo objetivo es facilitar la producción de una empresa o satisfacer la demanda de los clientes. Generalmente, los inventarios incluyen materia prima, productos en proceso y artículos terminados, por lo que se hace necesario un seguimiento de todos estos componentes para mantener un

mejor funcionamiento y control sobre el patrimonio de las entidades de una manera más segura y confiable.

1.3.1. Tipos de inventarios

Los inventarios son importantes para los fabricantes en general, los cuales varían ampliamente entre los diferentes grupos de industrias. La constitución de esta parte del activo no es más que una gran variedad de artículos, y es por eso que se han clasificado de acuerdo a su utilización en los siguientes tipos:

- "Inventarios de materia prima.
- Inventarios de producción en proceso.
- Inventarios de productos terminados.
- Inventarios de materiales y suministros."[6]

Inventarios de materia prima:

Este tipo de inventario comprende los elementos básicos o principales que ingresan en la elaboración de los productos. En toda actividad industrial concurren una variedad de artículos (materia prima) y materiales, los cuáles serán sometidos a un proceso para la obtención de un artículo terminado o acabado. A todos los materiales que intervienen en mayor nivel en la producción se les considera "Materia Prima", ya que su uso se hace en cantidades lo suficientemente importantes del producto final. En resumen, la materia prima, es aquel o aquellos artículos se someten a un proceso de fabricación para convertirse en un producto terminado.

Inventarios de productos en proceso:

El inventario de productos en proceso consiste en todos los artículos o elementos que se utilizan en el proceso de producción actual. Es decir, son productos parcialmente terminados que se encuentran en un grado intermedio de producción y a los cuáles se les aplica la labor directa y gastos indirectos inherentes al proceso de producción en un momento dado.

Una de las características de los inventarios de producto en proceso es que va aumentando el valor a medida que este es transformado de materia prima a producto terminado como consecuencia del proceso de producción.

Inventarios de productos terminados:

Comprende los artículos transferidos por el departamento o local de producción al almacén por haber alcanzado su grado de terminación total y que a la hora de la toma física de inventarios se encuentren aun en los almacenes, es decir, los que todavía no han sido vendidos o dados de baja.

Inventarios de materiales y suministros:

En el inventario de materiales y suministros se incluyen:

• Materias primas secundarias, sus especificaciones varían según el tipo de industria, un ejemplo; para el sistema de emisión de documentos de identificación: sobres para la protección de los documentos.

• Artículos de consumo destinados para ser usados en la operación de la industria. Dentro de estos artículos de consumo, los más importantes son los destinados a las operaciones, y están formados por los combustibles y lubricantes (estos no son registrados por el módulo de inventario pero forman parte de este tipo de inventario).

• Los artículos y materiales de reparación, mantenimiento de las maquinarias, equipos operativos y los artículos de reparación por su gran volumen necesitan ser controlados adecuadamente. La existencia de estos varían en relación a sus necesidades (estos no son registrados por el módulo de inventario pero forman parte de este tipo de inventario).

Existe otro tipo de inventario pero que no regula como tal los artículos o materiales, es el llamado **inventario de seguridad**. Este tipo de inventario es utilizado para impedir la interrupción en el aprovisionamiento causado por demoras en la entrega o por el aumento imprevisto de la demanda durante un período de reabastecimiento, la importancia del mismo está ligada al nivel de servicio, la fluctuación de la demanda y la variación de las demoras de la entrega.

1.3.2. Sistemas de inventario

Existen básicamente dos sistemas para el control de inventario: **el sistema de inventario perpetuo** o continuo y **el sistema de inventario periódico**.

En el **sistema de inventario perpetuo**, la entidad mantiene un registro continuo para cada artículo del inventario por lo que estos están disponibles todo el tiempo. Este método permite que el negocio pueda determinar el costo del inventario final y el costo de las mercancías vendidas (en dependencia del tipo de entidad) directamente de las cuentas sin tener que contabilizar el inventario.

El sistema perpetuo ofrece un alto grado de control, porque los registros de inventario están siempre actualizados. Anteriormente, los negocios utilizaban el sistema perpetuo principalmente para los inventarios de alto costo unitario[2], hoy en día con este método, los administradores pueden tomar mejores decisiones acerca de las cantidades a comprar, los precios a pagar por el inventario, la contabilización y ubicación de cada material entre otras cosas, en otras palabras, el conocimiento de la cantidad disponible ayuda a proteger el inventario.

Este sistema indicará oportunamente la disponibilidad de dicho producto además de alertar a la entidad de cualquier problema con su patrimonio. Los registros que proporciona este sistema muestran el inventario final existente de manera tal que no es necesario un conteo físico de los artículos.

En el **sistema de inventario periódico** la empresa no mantiene un registro continuo del inventario disponible, más bien, al final de un período determinado, esta hace un conteo físico del inventario disponible y aplica los costos unitarios para determinar el costo del inventario final. Se utiliza también para calcular el costo o detallar un informe de los activos que se dan de baja o las mercancías vendidas.

El sistema periódico es conocido también como sistema físico, porque se apoya en el conteo físico real del inventario y es utilizado generalmente para contabilizar los artículos del inventario que tienen un costo unitario bajo. Los artículos de bajo costo pueden no ser lo suficientemente valiosos para garantizar el

[2] Valúa las existencias que aparecen en el balance general y estado de pérdidas y ganancias en los renglones de los inventarios de producción en proceso y productos terminados.

costo de llevar un registro al día del inventario disponible. Para usar el sistema periódico con efectividad, el propietario debe tener la capacidad de controlar el inventario mediante la inspección visual.

1.3.3. Métodos utilizados en inventarios

Métodos de primeras entradas, primeras salidas (PEPS):

Bajo el método de primeras entradas, primeras salidas, la entidad debe llevar un registro del costo de cada unidad comprada del inventario. El costo de la unidad utilizado para calcular el inventario final, puede ser diferente de los costos unitarios utilizados para calcular el costo de las mercancías vendidas o dadas de baja. Bajo PEPS, los primeros costos que entran al inventario son los primeros costos que salen al costo de las mercancías vendidas, a eso se debe el nombre de Primeras Entradas, Primeras Salidas. El inventario final se basa en los costos de las compras o adquisiciones más recientes. Este inventario es también llamado por las iniciales que lo identifican en inglés FIFO (*First In First Out*).

Métodos de últimas entradas, primeras salidas (UEPS):

El método últimas entradas, primeras salidas depende también de los costos por compras de un inventario en particular. Bajo este método, los últimos costos que entran al inventario son los primeros costos que salen al costo de mercancías vendidas o dadas de baja. Este método deja los costos más antiguos (aquellos del inventario inicial y las compras primeras del período) en el inventario final. La filosofía de este método consiste en dar salida primero a los costos y materiales a los que se hicieron las últimas compras o adquisiciones. Esto trae como consecuencia que los inventarios que van quedando, estarán valorados a los materiales de las primeras compras. Este inventario es también llamado por las iniciales que lo identifican en inglés LIFO (*Last In First Out*).

Método de costo de promedio ponderado:

Aunque poco usado, es este otro método de valuación de los inventarios. Para determinar el costo promedio aritmético ponderado se procede como sigue:

- Se suman las unidades de que se ha dispuesto en el período es decir, el saldo inicial más las compradas, después de deducir las devoluciones.

- Se suman los respectivos costos.
- Se divide el costo total entre el total de unidades.

Método de promedio simple:

Este método de costear el inventario es también poco usado, a continuación se hace un bosquejo de cómo funciona.

Consiste en la determinación de un costo unitario promedio calculado como sigue:

- Se suman los costos unitarios tanto del inventario inicial como de las diferentes compras hechas en un período.
- El total así obtenido, se divide entre el número de partidas sumadas.

Después se procede a determinar el costo del inventario final y el correspondiente costo de ventas del período.

Método de promedio móvil:

Este método de control de inventarios tiene como características fundamentales las siguientes:

- Cada vez que entra en el almacén un lote de mercancía, el costo unitario del saldo resultante, debe ser recalculado.
- La existencia física es presentada en un solo total, en vez de estar segregado en lotes según el orden de entrada.
- El coste de las unidades que van saliendo, se hace en base al costo promedio calculado en saldo inmediato anterior.

1.4. Soluciones informáticas existentes

Actualmente existen varios sistemas para realizar el control de inventario en una empresa o entidad. La gran mayoría de estos sistemas poseen una amplia gama de funcionalidades que facilitan esta tarea en almacenes y empresas.

Entre las muchas soluciones informáticas existentes que realizan alguna tarea de este tipo, se encuentran los sistemas ERP (*Enterprise Resource Planning*), el cual consiste en un conjunto de programas integrados que apoya las principales actividades organizacionales como producción, logística, finanzas, contabilidad, ventas y recursos humanos. A continuación se analizarán algunos de estos sistemas:

1.4.1. BIOSS ERP

El *software* ***BIOSS*** en un sistema multi-empresa de ERP que integra módulos de Ventas y Distribución, Manufactura y Finanzas mediante tecnología basada en la Internet para optimizar el costo total de operación y mantenimiento de los sistemas de su empresa. Fue desarrollado y pensado para el mercado mexicano por IntraNets, Inc. con oficinas en Estados Unidos de América y México.

Esta es una solución cliente-servidor donde la estación de trabajo opera con cualquier versión de Windows y no requiere instalar ningún *software*. Para enlazarse al servidor solo se requiere el Microsoft Internet Explorer que viene incluido en Windows. El servidor utiliza Windows 2003 y Microsoft SQL Server. Los reportes han sido desarrollados con *Crystal Reports*, la herramienta estándar de la industria, conocida por la mayoría de los profesionales de sistemas.

Ofrece más de 10 módulos adaptables a una empresa. El módulo de Control de Inventarios tiene que ver con las transacciones que involucran entradas y salidas de almacén y las políticas que rigen los niveles de existencias.

Casi todos los módulos de ***BIOSS*** afectan inventarios. Existen conceptos y políticas de inventario generales que se aplican a todos los módulos. Además para cada módulo se definen operaciones de inventario específicas al módulo particular.

1.4.2. Exactus ERP

Exactus ERP es una solución de *software* empresarial diseñada específicamente para cumplir con los requerimientos de la empresa mediana en Latinoamérica. Es un sistema de gestión totalmente integrado y escalable que abarca las áreas Financiera - Contable, Comercial (Ventas e Inventario), Industrial (Manufactura) y Recursos Humanos y Nómina, con altos estándares de calidad.

Este *software* se puede integrar con soluciones en el área de Inteligencia de Negocios, CRM (por sus siglas en inglés ***C****ustomer* ***R****elationship* ***M****anagement*), Aplicaciones Web (Portal de Personal y Comercio Electrónico) y Aplicaciones Móviles (Facturación en Ruta y Administración de Almacén). Es un sistema integrado, accesible, de rápida implementación y que crece con su empresa y lo más importante, a un costo fijo, es decir funciona bajo un modelo de operación pre – definido para cada paquete comercial ofrecido. Esta opción se diseña mediante una serie de entregables y actividades específicas acordadas con el cliente, antes de arrancar el proyecto, las cuales se detallan en un cronograma de trabajo.

Presenta entre sus componentes, un módulo para el control de inventario que ha sido diseñado pensando en las necesidades de las empresas manufactureras, de distribución y servicio que requieren de un control estricto de sus existencias de artículos de consumo interno y para la venta.

Control de inventarios integra todos los movimientos transaccionales que tienen relación con consumos, producción, ventas, ingresos, entre otros, realizados directamente en él o desde los demás módulos con los que interactúa.

1.4.3. Microsoft Dynamics NAV

Desde 1984, Microsoft Dynamics NAV se ha establecido como opción ideal para organizaciones de tamaño medio que buscan una solución completa de planificación de recursos empresariales (ERP) que sea rápida de implantar, fácil de configurar y sencilla de usar. Más de un millón de usuarios han utilizado Microsoft Dynamics NAV para simplificar y optimizar sus procesos empresariales altamente especializados y, en la actualidad, está disponible en versiones para más de 42 países. Microsoft Dynamics NAV ofrece una adaptabilidad rápida, así como una personalización simplificada, lo que permite a las compañías añadir fácilmente funcionalidades y aplicaciones personalizadas. Contar con un ERP como Microsoft Dynamics NAV en una institución se podrá:

- Automatizar los procesos críticos para la empresa para aumentar su propia eficacia y la del personal.
- Ayuda a reducir los costes de compra e inventario.
- Visibilidad en los procesos de inventarios.

Este sistema no es multiplataforma ya que solamente corre sobre el sistema operativo Windows y requiere una versión Windows XP SP 2 o superior. Además, es un *software* privativo por lo que hay que pagar para su adquisición.

1.4.4. CONDOR

Sistema que ofrece soluciones óptimas adaptadas a las normas y principios de la Contabilidad General aceptados en Cuba conforme a las normas y procedimientos establecidos por los Ministerios de Finanzas y Precios y de Informática y las Comunicaciones.

Es una aplicación informática diseñada acorde con los principios actuales de la ingeniería de *software* que cumple con todos los criterios de trabajo en red simultáneo o multi-instancias, que posibilita un trabajo mucho más organizado y eficiente sobre la información, pues todos los operadores del sistema tributan sobre la misma fuente de datos, siempre con su debida distinción de usuarios en cuanto a funcionalidades a las que tienen acceso.

El sistema está compuesto por módulos y permite el intercambio entre ellos automáticamente y por opciones. Entre sus módulos se encuentra el de control de inventario, concebido para el control de las entradas y salidas de los almacenes, contabilización de los movimientos que se realizan con los productos almacenados, control de herramientas en uso y facturación.

El Sistema CONDOR se puede comercializar como módulos aislados. La concesión de la Licencia de Uso no le otorga derechos de venta o regalo del mismo; esta se otorga por períodos anuales, previo contrato. La licencia de uso del módulo de control de inventario tiene un costo de 2779.74 MN.

1.4.5. EMIPAS

El sistema EMIPAS está concebido para la personalización y emisión de pasaportes. En su funcionamiento cumple con las normas internacionales de la Organización de la Aeronáutica Civil (OACI), para pasaportes de lectura mecánica.

Uno de los pilares fundamentales del EMIPAS es lograr que la información que se encauza sea confiable y no pueda ser adulterada, durante el procesamiento. Esta filosofía está implícita en todo el proceso de

diseño y funcionamiento del sistema. Este sistema, implementado la plataforma .NET, cuenta con un servidor Oracle para la base datos, que se combina con: autentificación biométrica de los usuarios, registro de los equipos autorizados y transferencias encriptadas. Este sistema no presenta la ventaja de ser multiplataforma al requerir para su funcionamiento Windows 2000 o superior.

Para evitar los actos ilícitos de los funcionarios, el sistema cuenta con: un control de los permisos atribuidos a cada uno, un registro de las acciones que realicen los usuarios; de cualquier rol, con alertas automáticas sobre acciones sospechosas y un control de inventario de las libretas de pasaportes.

EMIPAS está compuesto básicamente por cuatro módulos: “Personalización”, “Administración”, “Arraigos” y “Reportes”. Precisamente en el módulo de administración es donde se realiza el control de inventario, aspecto principal donde se centra el análisis de este sistema.

1.4.6. SAIME

El SAIME es una solución de *software* que integra varios módulos y subproyectos como son Identificación, que incluye Cedulación y Emisión de Pasaportes, y Migración para el control migratorio en Aeropuertos, puertos, puntos fronterizos y Extranjería. Este sistema se hizo con el propósito de informatizar los procesos de identificación y control fronterizo y de extranjeros, en un país donde los documentos oficiales podían ser fáciles de falsificar, obteniéndose a altos precios.

Se integra con un Sistema Automatizado de Identificación de Huellas Dactilares (AFIS, de sus siglas en inglés *Automated Fingerprint Identification System*) Civil para el procesamiento de datos biométricos al ciudadano, y con productos para la emisión de pasaportes electrónicos contando además con módulos para la administración y el control de inventario, servicios a través de un portal web y un centro de llamadas para atención al cliente.

Principales beneficios para el cliente.

SAIME incorpora tecnologías actuales y cumple con los estándares internacionales. Entre los elementos fundamentales que lo componen se encuentran:

- Sistemas de Enrolamiento para la emisión de documentos de identificación.

- Sistemas de Control fronterizo.
- Sistema de Personalización de pasaportes de lectura mecánica (Pasaporte andino venezolano) y documentos para extranjeros (visas, permisos de residencia).

Integración con sistema de personalización de pasaportes electrónicos.

- Integración con solución AFIS.
- Sistema de Personalización de documentos para el control de extranjeros.

Impacto social a nivel de implantación.

La solución ha representado un gran aporte a la ciudadanía venezolana. Se han emitido hasta la fecha más de un millón de pasaportes electrónicos el cual es el primero desarrollado en Latinoamérica, y en general se registran miles de trámites a diario, lo cual ha hecho que exista un cambio total en la imagen de la organización para la que se desarrolló esta solución.

Entre los recursos utilizados por este sistema se encuentran:

1. Servidor: Sistema Operativo RedHat Enterprise Linux Advanced Server.

- Gestor de Bases de Datos Oracle 10.2.

2. Cliente: Sistema Operativo Windows XP.

- Aplicación utilizando Microsoft .Net *framework* v1.1.

Las soluciones analizadas a nivel nacional e internacional presentan muchas ventajas, pero latentes inconvenientes que las hacen soluciones no factibles para implantar en el MPPRE, puesto que el control de inventario que es el centro de atención en la investigación de estos sistemas no cumplen con los requisitos fundamentales, necesarios e indispensables para ser adaptado a las características y exigencias del cliente. Además de que son o utilizan herramientas privativas, por lo que hay que pagar costosas licencias para su adquisición o utilización.

1.5. Herramientas a utilizar en el proceso de desarrollo

1.5.1. IDE Eclipse Galileo v3.5

El *software* Eclipse es un Entorno Integrado de Desarrollo, del inglés *Integrated Development Environment* (IDE) de código abierto y multiplataforma, hecho principalmente para el desarrollo de aplicaciones Java. En sus inicios fue desarrollado por IBM, y actualmente es gestionado por la Fundación Eclipse.

A diferencia de otros IDE, donde todas las funcionalidades vienen incluidas, Eclipse utiliza módulos (conocidos también como plug-in[3]) que hacen de este *software* una herramienta extensible, pues a través de los mismos, pude incluir nuevas funcionalidades para programas en C/C++, PHP, Python y otros más. Entre las facilidades que brinda se encuentra la gestión de proyectos mediante la incorporación de sistemas de control de versiones (CVS, del inglés *Concurrent Versions System*) permitiendo así mantener un registro de los cambios y trabajos realizados en el código fuente[4] de un proyecto contribuyendo a la colaboración entre los desarrolladores.

1.5.2. Apache Tomcat

Apache Tomcat es un servidor HTTP y un contenedor de *servlets*[5]/JSP. Es la implementación de referencia de las especificaciones de *servlets* (2.4) y de JSP (2.0). Es *software* libre bajo la licencia Apache 2.0 y es gestionado por la fundación Apache. Puede funcionar como servidor HTTP o conectado a otro servidor HTTP como Apache HTTP Server o IIS (Internet Information Services) y puede ejecutar servicios web mediante Apache Axis.

1.5.3. Spring

Spring es un *framework* de aplicación de código abierto desarrollado para aplicaciones escritas en el lenguaje de programación Java. Es una potente herramienta de gestión de configuración basada en JavaBeans[6], aplicando los principios de Inversión de Control (IoC) y una Programación Orientada a

[3] Ver glosario de terminos.

[4] El formato entendible por las personas de las instrucciones que conforman un programa.

[5] Ver glosario de términos.

[6] Ver glosario de términos.

Aspectos *(Aspect Oriented Programming*). Esto hace que la configuración de aplicaciones sea rápida y sencilla.

Spring integra herramientas tales como EJB (*Enterprise JavaBeans*), *Servlets*, y JSP (*Java Server Pages*) en un solo paquete para desarrollar aplicaciones J2EE (*Java 2 Enterprise Editions*) y brindar una estructura más sólida y ofrecer un mejor soporte para este tipo de aplicaciones.

"Spring no intenta "reinventar la rueda" sino integrar las diferentes tecnologías existentes, en un solo *framework* para el desarrollo más sencillo y eficaz de aplicaciones J2EE portables entre servidores de aplicación (Johnson, 2005)."[7]

1.5.4. JUnit

JUnit es un *framework* de código abierto que se utiliza para la automatización de las pruebas tanto unitarias, como de integración, en los proyectos de desarrollo de *software*. Este *framework* provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad.

"JUnit tiene diversos paquetes: *framework* como paquete básico de marcos, *runner* para algunas clases abstractas y para la ejecución de pruebas, *textui* y *swingui* para interfaces de usuario y *extensions* para algunas contribuciones prácticas al marco."[8]

1.5.5. Control de versiones

Parte importante en la implementación de un sistema informático es el control de versiones. Un sistema de control de versiones (o sistema de control de revisiones) es una combinación de tecnologías y prácticas para seguir y controlar los cambios realizados en los ficheros del proyecto, en particular en el código fuente, en la documentación y en las páginas web. Este ayuda virtualmente en todos los aspectos al dirigir un proyecto: comunicación entre los desarrolladores, manejo de los lanzamientos, administración de fallos, estabilidad entre el código y los esfuerzos de desarrollo experimental, atribución y autorización en los cambios de los desarrolladores.

En este caso la herramienta utilizada para realizar el control de versiones es **TortoiseSVN**, esta herramienta es un cliente gratuito de código abierto para el sistema de control de versiones *Subversion*. TortoiseSVN maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios.

1.6. Lenguajes utilizados

1.6.1. Java

Java es un lenguaje de Programación Orientada a Objeto (POO) creado por la compañía *Sun Microsystems* a mediados de los años 90. Es un lenguaje que soporta el encapsulamiento, la herencia y el polimorfismo (los 3 pilares del paradigma de la POO) y está siendo muy usado en el mundo.

Java posee muchas características que lo hacen un lenguaje muy utilizado por los desarrolladores entre las cuales se encuentran:

"**Lenguaje simple:** Posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir los *applets*[7], interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de la gente los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo."[9]

Produce applets: Java puede ser usado para crear dos tipos de programas: aplicaciones independientes y *applets*. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador web *HotJava*, escrito íntegramente en Java. Por su parte, los *applets* son pequeños programas que aparecen embebidos en las páginas web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

[7] Ver glosario de término.

"Orientado a objeto: Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia del futuro, a la que Java se suma, apunta hacia la POO, especialmente en entornos cada vez más complejos y basados en red. Distribuido: Permite abrir sockets[8], establecer y aceptar conexiones con los servidores o clientes remotos; facilita la creación de aplicaciones distribuidas ya que proporciona una colección de clases para aplicaciones en red.

Robusto: Java fue diseñado para crear *software* altamente 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." [10]

Seguro: Tiene políticas que evitan que se puedan codificar virus con este lenguaje. Existen muchas restricciones, especialmente para los *applets*, que limitan lo que se puede y no puede hacer con los recursos críticos de una computadora.

Indiferente a la arquitectura: Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan heterogénea, el compilador de Java genera *bytecodes*[9] para transportar el código eficientemente a múltiples plataformas *hardware* y *software*. El resto de los problemas los soluciona el intérprete de Java.

1.6.2. XML

XML es un Lenguaje de Marcas Extensible, de sus sigla en inglés eXtensible Markup Language, es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). XML describe los datos de manera fácil de analizar y que no está atado a un lenguaje de programación en

[8] Ver glosario de terminos.

[9] Ver glosario de términos.

específico, esto posibilita transferir información de manera segura, fiable y fácil entre aplicaciones, incluso si estas aplicaciones residen en diferentes sistemas operativos o son escritos en diferentes lenguajes de programación. XML provee múltiples vistas para los datos por eso su uso es bien variado, se pueden encontrar en bases de datos, editores de texto, hojas de cálculo entre otros.

XML no es, como su nombre puede sugerir, un lenguaje de marcado, en realidad es un metalenguaje que permite definir lenguajes de marcado adecuados a usos específicos. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores.

1.7. Metodología de desarrollo utilizada

Actualmente existe un gran número de metodologías que guían el proceso de desarrollo del *software* para que este se realice con la mayor transparencia y calidad. Existen fundamentalmente dos variantes o tipos de metodologías entre las cuales se encuentran las metodologías ágiles o ligeras (SCRUM[10], XP[11], FDD[12]) y las llamadas pesadas o tradicionales (RUP[13], MSF[14]). El uso que se les da a las metodologías ágiles hoy en día para el desarrollo de *software* es muy común, debido a las ventajas que las mismas brindan con respecto a las metodologías pesadas.

En este trabajo se centra la atención en las metodologías ágiles, específicamente en la metodología FDD, por la habilidad de responder de forma versátil al cambio para maximizar los beneficios.

1.7.1. Feature Driven Development

Desarrollo basado en funcionalidades, del inglés **F**eature **D**riven **D**evelopment (FDD) es una metodología ágil e iterativa diseñada por Peter Coad, Eric Lefebvre y Jeff DeLuca. Esta metodología forma parte de la organización Ágil Alliance (organización sin fines de lucro que busca promover el conocimiento y la

10 SCRUM: Metodología ágil para la gestión y desarrollo de software.
11 XP: eXtreme Programming.
12 FDD: Feature Driven Development.
13 RUP: Rational Unified Process.
14 MSF: Microsoft Solution Framework

discusión de todos los métodos ágiles) y fue diseñada originalmente para un proyecto de 50 personas y con un tiempo de desarrollo relativamente corto (alrededor de 1 año).

Es una metodología escalable ya que a mayor tamaño de código y/o equipo mayor es la necesidad del cliente. A pesar de ser una metodología ágil, tiene un proceso con un peso intermedio en el sentido de que genera más documentación que otras metodologías ágiles como XP, además de que entrega bastante libertad a los desarrolladores pero siempre bajo cierto orden marcado por una jerarquía. El aseguramiento de la calidad de FDD no se basa en formalismos, sino en controles propios y comunicación fluida con el cliente. Esta metodología establece cinco procesos para el desarrollo de un *software*:

1. **Desarrollo de un modelo global:**

Cuando comienza el desarrollo, los expertos del dominio están al tanto de la visión, el contexto y los requerimientos del sistema a construir. Luego se divide el dominio global en áreas que son analizadas detalladamente. Posteriormente los desarrolladores construyen un diagrama de clases o de objetos por cada área y se construye un modelo global del sistema.

2. **Construcción de una lista de funcionalidades:**

Una funcionalidad es un ítem útil a los ojos del cliente. Se elabora una lista de funcionalidades que resuma la funcionalidad general del sistema. La lista es elaborada por los desarrolladores y es evaluada por el cliente. Se divide la lista en subconjuntos según la afinidad y la dependencia de las funcionalidades. La lista es finalmente revisada por los usuarios y los responsables para su validación y aprobación.

3. **Planeación por funcionalidad:**

En este punto se procede a ordenar los conjuntos de funcionalidades conforme a su prioridad y dependencia, y se asigna a los programadores jefes.

4. **Diseño por funcionalidades:**

Se selecciona un conjunto de funcionalidades de la lista. Se procede a diseñar las funcionalidades que en conjunto con el proceso de construcción por funcionalidades se realiza mediante un proceso iterativo. Una iteración puede tomar de unos pocos días a un máximo de dos semanas.

5. Construcción por funcionalidades:

Se procede a construir las funcionalidades diseñadas. Una iteración puede tomar de unos pocos días a un máximo de dos semanas. El proceso iterativo incluye inspección de diseño, codificación, pruebas unitarias, integración e inspección de código.

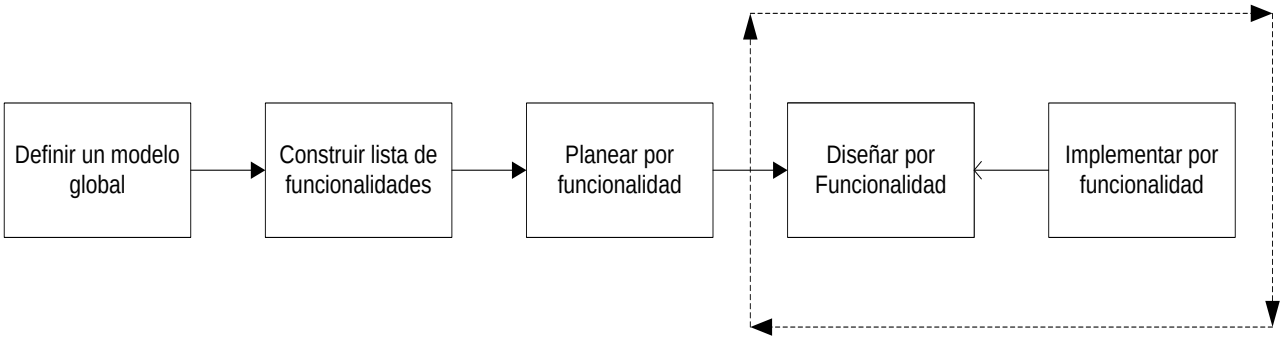


Figura 1: Fases de la metodología FDD

1.7.2. Lenguaje de modelado

Para dar soporte a la metodología seleccionada se utilizó el Lenguaje Unificado de Modelado (UML). UML es un lenguaje gráfico utilizado para entender, diseñar, construir, documentar y especificar un sistema de *software*. Entre las características que este posee se encuentran:

- Viabilidad en la corrección de errores.
- Participación del cliente en todas las etapas del proyecto.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- Utilizando UML se pueden modelar sistemas bajo el paradigma orientado a objetos (OO).

- 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.
- UML permite la creación de los diferentes modelos que ofrecen las vistas necesarias para la construcción de un *software* de calidad.

1.7.3. Herramienta CASE: Visual Paradigm v3.4

La herramienta utilizada para realizar el modelado del sistema fue Visual Paradigm. Este *software* proporciona una rápida y eficiente construcción de aplicaciones con calidad y a un menor costo. Permite diseñar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Una de sus principales ventajas es que es robusto, presenta gran usabilidad y portabilidad. Entre las ventajas que esta ofrece también se encuentran:

- Una rápida construcción de aplicaciones con calidad y a un menor costo.
- Ofrece capacidades de ingeniería directa e inversa.
- 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.
- 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.

1.8. Base de datos

Una base de datos es una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular. En otras palabras, es un sistema de archivos electrónico para adicionar, actualizar, eliminar y consultar información siempre que se necesita.

Los **S**istemas de **G**estión de **B**ases de **D**atos (**SGBD)** en inglés *database management system*, abreviado DBMS. Estos sistemas son un tipo de *software* muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Entre los SGBD más utilizados se encuentran PostgresSQL y SQLite entre los denominados *software* libre, y como *software* privativos los gestores Oracle y Microsoft SQL Server.

1.8.1. PostgreSQL v8.4

PostgreSQL es un gestor de base de datos relacional muy potente y de código abierto creado en la Universidad de Berkeley en el año 1977 y bautizado con el nombre Ingres y cuenta ya con más de 15 años de desarrollo. Este gestor presenta una amplia documentación que es pública y libre además de que corre sobre los principales sistemas operativos: Linux, Windows, Mac Os, Unix, etc. Entre las ventajas que presenta se encuentran:

- Soporte nativo para los lenguajes más populares: Java, PHP, C, C++, Python, etc.
- Soporte de todas las características de una base de datos profesional (*triggers*, *store procedures* – funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuarios, vistas, vistas materializadas, etc.).
- Soporte de protocolo de comunicación encriptado por SSL (***S**ecure **S**ocket **L**ayer*).
- Es altamente extensible pues soporta operadores, funcionales métodos de acceso y tipos de datos definidos por el usuario.

- Usa tecnología MVCC, o Control de Concurrencia Multi-Versión (***M**ulti-**V**ersion **C**oncurrency **C**ontrol*) para evitar bloqueos innecesarios.

1.8.2. Hibernate

Hibernate es una herramienta de mapeo objeto – relacional (***O**bject-**R**elational **M**apping*, ORM) para entornos Java (disponible también para .NET con el nombre NHibernate). El término ORM, se refiere a técnica de mapear una representación de datos desde un modelo de objeto a un modelo de datos relacional con un esquema basado en SQL, además de facilitar la consulta y recuperación de datos.

Esta herramienta o *framework* es *software* libre bajo la licencia GNU/LGPL, además de que ofrece un lenguaje propio de consultas de datos llamado HQL (*Hibernate Query Languaje*). Hibernate podrá generar bases de datos en cualquiera de los **S**istemas **G**estores de **B**ases de **D**atos **R**elacionales (SGBDR) entre los cuales se encuentra: PostgreSQL, MySQL, Oracle, entre otros.

1.9. Conclusiones parciales

Este capítulo se realizó una base teórica para entender el tanto el problema planteado como el ambiente en el cual se desenvuelve el mismo. Se realizó un estado del arte del tema tanto a nivel nacional como internacional y se determinó que era necesaria la implementación de un módulo para el control de inventario del subsistema de personalización. Se utilizarán las herramientas definidas en el proyecto; Java como lenguaje de programación, Eclipse Galileo como entorno de desarrollo, Spring para trabajar el negocio, Hibernate para el acceso a datos, PostgreSQL como gestor de bases de datos y para guiar el proceso de desarrollo la metodología ágil FDD.

Capítulo 2: Descripción y análisis de la solución propuesta

2.1. Introducción

En este capítulo se explican las funcionalidades que debe presentar la aplicación, así como una valoración crítica del diseño propuesto por el analista, los patrones de diseño y arquitectónicos utilizados, también se explica la estructura de la aplicación, se describen las principales clases desarrolladas, la descripción de la solución propuesta y los diagramas de clases del diseño, por paquetes y despliegue, además de los estándares de codificación usados para la implementación.

2.2. Descripción de la solución propuesta

La solución propuesta consiste en el desarrollo de los servicios a nivel de proceso del módulo de control de inventario del subsistema de personalización del Sistema de Emisión de Pasaporte Diplomático, de Servicio y Acreditaciones. Dicho servicio contará con las tareas o funcionalidades que consumirá la capa de interfaz o aplicación cliente, las cuales son generalmente funcionalidades de gestión. En este módulo se gestionan todos los locales que tienen algún vínculo con el proceso de personalización, así como de las estructuras y otros locales que estos pueden contener dentro de sí. La materia prima que se guarda y traslada por los mismos, consiste en lotes que contienen documentos de identificación de los cuales se registran todos sus datos, fecha de entrada, salida y localización para facilitar la organización y control de los documentos.

2.3. Propuesta del analista

Haciendo uso de la metodología FDD los analistas del proyecto realizaron una propuesta de solución que satisface las necesidades del cliente pero no es lo suficientemente explícita para que los desarrolladores puedan realizar una total implementación del módulo.

Como artefacto resultante tras el análisis y diseño de la solución se encuentra la lista de funcionalidades, que es entregada para que el desarrollador lleve a cabo la implementación. La lista de funcionalidades cuenta con una descripción de las acciones o actividades que debe cumplir el sistema y las interfaces correspondientes para la solución de las mismas. Se realiza una descripción por cada característica

principal del sistema, reemplazan un gran documento de requisitos y presiden la creación de las pruebas de aceptación.

En el diseño propuesto no se definen las clases que se necesitan para un correcto funcionamiento del sistema, extendiéndose el tiempo en que pudiera implementarse dicho módulo. Tampoco existe un diseño de los diagramas de componentes, despliegue y clases del diseño, por lo que se hizo necesario el modelado de los mismos por el desarrollador durante la etapa de implementación.

2.3.1. Requisitos funcionales (RF)

"Según Somerville, los requisitos no funcionales definen propiedades y restricciones del sistema, - aunque la palabra restricciones puede ser ambigua, porque en otros textos, una restricción es algo impuesto por la organización, desde el punto de vista de *software*." [11]

Los analistas del proyecto definieron para el módulo una serie de requisitos funcionales, dentro de los que están:

Para el administrador de inventario (Permisos: Gestionar locales, gestionar tipos de estructuras y asignar responsables y roles a locales).

RF1 Gestionar local

1.1. Registrar datos de un local.
- Nombre del local.
- Número del local.
- Tipo de local (Almacén, Personalización, Entrega).

1.2. Mostrar listado de locales existentes.
- Nombre del local.

1.3. Mostrar datos de un local específico.

1.4. Modificar datos de un local.

1.5. Eliminar un local.

RF2 Gestionar tipo de estructura

2.1. Registrar datos de un tipo de estructura.
- Nombre del tipo de estructura.

2.2. Mostrar listado de tipos de estructuras.
- Nombre del tipo de estructura.

2.3. Mostrar datos de un tipo de estructura específico.

2.4. Modificar datos de un tipo de estructura.

2.5. Eliminar un tipo de estructura.

2.6. Buscar tipo de estructura por nombre del tipo de estructura.

RF3 Asignar responsable a local

3.1. Mostrar listado de locales existentes.
- Nombre del local.

3.2. Permitir la selección de un local.

3.3. Mostrar listado de responsables del local seleccionado.
- Nombres y apellidos.

3.4. Mostrar listado de roles del sistema.
- Nombre.

3.5. Buscar usuario por nombre de usuario.

3.6. Mostrar datos de usuario.
- Nombres y apellidos.
- Rol.

3.7. Permitir la asignación de un usuario con un rol que contenga el local.

3.8. Permitir la asignación de un rol a un local.

3.9. Eliminar la asignación de un usuario a un local.

Para el almacenista (Permisos: Permitir la configuración de las estructuras del local, registrar pasaportes a acreditar, registrar materias primas, permitir la clasificación de lotes, permitir la asignación de preimpresos a locales, recibir documentos anulados de personalización, recibir documentos terminados de personalización, enviar documentos a entrega, destruir documentos, reubicar lote).

RF4 Permitir la configuración de las estructuras del local

4.1. Listar locales a los que el almacenista tiene permisos de acceso.

4.2. Permitir la selección de un local.

4.3. Mostrar estructura del local seleccionado.

- Nombre de estructura.
- Tipo de estructura.

4.4. Adicionar datos de estructura.

- Nombre de estructura.
- Tipo de estructura.
- Clasificación.

4.5. Modificar datos de estructura.

4.6. Eliminar estructura.

RF5 Permitir el recibo de materias primas

5.1. Registrar datos de un lote.

- Lote.
- Fecha de caducidad.
- Estado.
- Documento.
- Serie inicial.
- Cantidad.
- Serie final.

5.2. Listar lotes que hayan sido registrados en el sistema.

- Lote
- Fecha de caducidad.
- Documento.
- Serie inicial.
- Cantidad.
- Serie final.

RF6 Permitir el recibo de pasaportes por sellar

6.1. Listar trámites de pasaportes por sellar.

- Nombres y apellidos.
- № Pasaporte.

- Nacionalidad.

6.2. Permitir el registro de lotes de pasaporte por sellar que se genere automáticamente.

6.3. Permitir la selección de un local.

6.4. Mostrar la estructura del local.

6.5. Asignar lotes de pasaportes por sellar a un local.

RF7 Clasificar lotes de documentos

7.1. Listar lotes con estado virgen en el sistema.

- Descripción (Lote, Cantidad).

7.2. Permitir la selección de uno o varios lotes.

7.3. Listar documentos contenidos en los lotes.

7.4. Permitir el reclamo de documentos.

7.5. Reconfeccionar lotes.

7.6. Asignar ubicación de lotes en el almacén.

RF8 Permitir la asignación de materias primas

8.1. Mostrar listado de lotes de documentos vírgenes.

8.1.1. Permitir la selección del tipo de documento.

8.2. Reconfeccionar lotes de acuerdo a una cantidad específica.

8.3. Listar los locales de tipo personalización.

8.4. Asignar lotes a un local.

RF9 Recibir documentos anulados

9.1. Listar lotes de documentos anulados.

- Descripción (Lote, cantidad).
- Documentos.

9.2. Permitir la selección de local.

9.3. Mostrar estructura del local.

9.4. Asignar documentos anulados a una estructura del local seleccionado.

RF10 Recibir documentos terminados

10.1. Listar lotes de documentos anulados.

- Descripción (Lote, cantidad).
- Documentos.

10.2. Permitir la selección de local.

10.3. Mostrar estructura del local.

10.4. Asignar documentos terminados a una estructura del local seleccionado.

RF11 Enviar lotes al área de entrega

11.1. Listar lotes de documentos con estado terminado en almacén.

- Lote.
- Documentos.

11.2. Permitir la selección de local.

11.3. Asignar documentos terminados al local de entrega.

RF12 Dar baja a documentos anulados

12.1. Actualizar el estado del lote a destruido.

RF13 Reubicar lotes

13.1. Permitir la búsqueda de un lote.

13.2. Mostrar ubicación origen.

13.3. Asignarle una nueva ubicación.

2.3.2. Requisitos no funcionales (RnF)

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

“Los requerimientos no funcionales forman una parte significativa de la especificación. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con la funcionalidad requerida, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación (Jacobson, 2000).” [12]

Fiabilidad

RnF1 Se encontrará disponible de lunes a viernes, el horario comprendido será desde las 0:00h del lunes hasta 23:59h del viernes, lo cual hace un total del 60h semanales. El mantenimiento se realizará en horario de 0:00h del sábado hasta las 11:59 del domingo.

RnF2 Tiempo medio entre fallos: Dispondrá de un tiempo medio entre fallos de 6:00h.

RnF3 Errores del sistema:

a. Menores: Fallos sencillos en el sistema. Ejemplo: referencia errónea, problema de codificación, etc.

b. Significativos: Fallos de importancia media que no imposibilitan el desenvolvimiento del sistema. Ejemplo: caída del módulo de citas en la administración del subsistema.

c. Críticos: Errores graves que imposibilitan la ejecución del sistema.

Soporte

RnF4 Dispone de un acceso desde las oficinas del área destinada para la ejecución del trámite.

RnF5 El módulo de control de inventario se identifica como *resourcemodule*.

RnF6 El paquete *facade* identifica la lógica de negocios dentro de un módulo.

RnF7 El paquete *dataaccess* identifica la persistencia dentro de un módulo.

Restricciones de diseño

RnF8 El módulo de control de inventario se desarrollará en lenguaje java.

RnF9 Cada componente de *software* está formado por una capa de negocio y de persistencia.

RnF10 El IDE de desarrollo y mantenimiento del sistema es Eclipse Galileo (v 3.5).

RnF11 Los artefactos de diseño del *software* se realizan en Visual Paradigm.

RnF12 Las tareas generadas por el motor de procesos se exponen mediante un servicio web en axis2[15] para el acceso de las aplicaciones clientes.

2.4. Modelo de dominio

Un modelo de dominio es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción. El modelo de dominio presenta uno o más diagramas de clases que no contienen conceptos propios de un sistema de *software* sino de la propia realidad física. Se utiliza para capturar y expresar el entendimiento ganado en un área bajo análisis, como paso previo al diseño de un sistema. Teniendo en cuenta que el problema a resolver no está determinado a partir de procesos bien definidos que permitan modelar el funcionamiento de la misma, se ha procedido a crear un modelo de domino.

2.4.1. Modelo conceptual de dominio

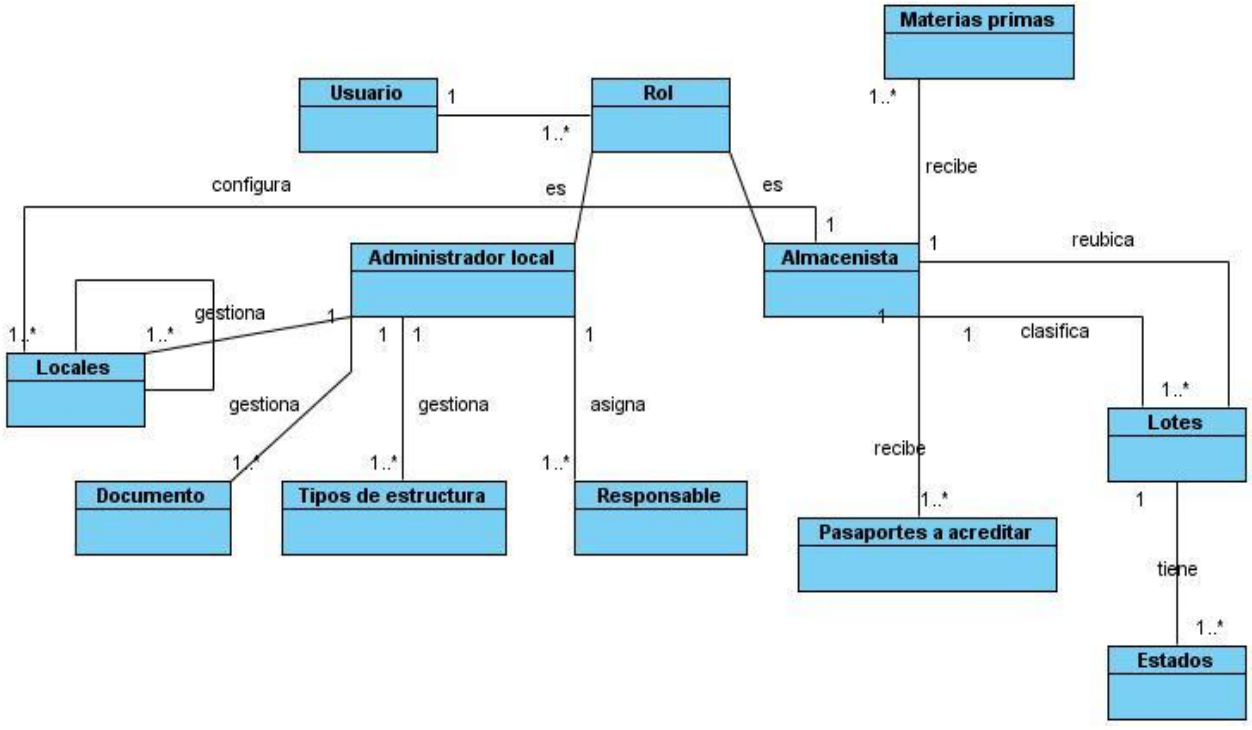


Figura 2: Modelo conceptual de dominio

[15] Ver glosario de términos.

Para el entendimiento del modelo conceptual de dominio para el módulo de control de inventario, es necesario e importante conocer el significado de los siguientes términos:

Lote: Paquete que contiene documentos de identificación y posee información general sobre los documentos que contiene.

Local: Es un local que puede estar formado por varios locales y puede almacenar o contener estructuras y lotes.

Documento: Refiere a los documentos de identificación registrados en el sistema y que tienen una ubicación en un lote determinado.

Responsable: Es el usuario registrado en el sistema que posee capacitación y conocimiento suficiente para administrar un local.

2.5. Patrones de diseño implementados

Los patrones son principios generales basados en la experiencia que aplican ciertos conocimientos y estilos que guían la creación de un *software*. Cada patrón especifica un problema recurrente en el diseño y la implementación de un *software*. Los patrones de diseño procuran en esencia suministrar elementos reutilizables en el diseño de los sistemas de *software*.

2.5.1. Patrones 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 solo con quien lo necesita para realizar sus procedimientos (o métodos). Como ejemplo se tiene que: para realizar las acciones de la tarea “gestionar tipo de estructura”, la clase **GestionTipoEstructuraFTaskImpl** recurre a la interfaz **GestionTipoEstructuraFacade** solamente, pues es esta la que contiene los procedimientos de los cuales se apoya para realizar los suyos.

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 las capas de negocio y servicio, 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. Al igual que el patrón anterior, ejemplos de este están plasmados en todas las clases de las capas de negocio y servicio. Un ejemplo sería la clase **GestionLocalFacadeImpl**, la cual es la responsable de realizar todas las operaciones acerca de un local del negocio, por lo que los métodos relacionados con las operaciones básicas sobre los locales estarán en la misma.

Creador: Permite decidir cuáles serán las clases creadoras de otras clases. Este patrón se tiene muy en cuenta a la hora de estructurar las clases según la arquitectura, donde cada clase de una capa superior crea su similar en la inferior (en conjunto con el *framework*, ya que Spring es quién maneja los objetos de negocio), asimismo se tiene de ejemplo que la clase **GestionTipoEstructuraFTaskImpl** es quien concibe (y en conjunto con el *framework*) crea la clase **GestionTipoEstructuraFacade**, la cual utilizará para sus operaciones.

2.5.2. Patrón Gang of Four (GoF)

Patrón Fachada (Facade): El patrón de diseño fachada se empleó para proveer de una interfaz unificada y sencilla, que haga de intermediaria entre el cliente y la interfaz o grupo de interfaces más complejas, permitiendo así una mayor flexibilidad en el desarrollo del sistema. Permite reducir la complejidad y minimizar las dependencias.

2.6. Arquitectura base

Según un reconocido autor Paul Clements: “La arquitectura de *software* es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.” [13]

2.6.1. Arquitectura cliente/servidor

La arquitectura cliente servidor también llamada modelo cliente – servidor, es una forma de dividir o separar al cliente (en este caso la interfaz gráfica) del servidor. El cliente es aquel que inicia el diálogo o comunicación solicitando información y el servidor es el que procesa las solicitudes del cliente y retorna una respuesta.

Para que los clientes y los servidores puedan establecer una comunicación se necesita una infraestructura de comunicaciones o protocolo de comunicación, estos brindan los mecanismos básicos para el direccionamiento y transporte de los datos.

Esta arquitectura presenta disímiles ventajas ya que la aplicación cliente pueden actuar tanto como una sola entidad que como entidades separadas y las funciones de cada uno pueden ejecutarse sobre plataformas diferentes, además de que la aplicación cliente no necesita conocer la lógica del servidor, ya que solo interactúa con esta a través de una interfaz y los cambios que ocurran en el servidor implican pocos o ningún cambio en el cliente.

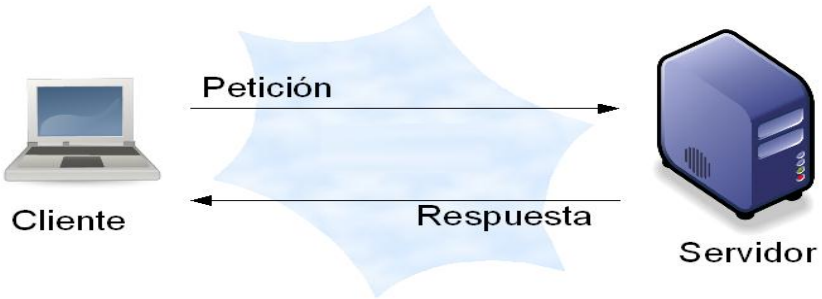


Figura 3: Arquitectura cliente - servidor

Esta arquitectura está estructurada en 3 capas: la capa de presentación o de interfaz en la cual se encuentra la aplicación cliente que hace peticiones al servidor, la capa de procesos de negocio donde se realizan las automatizaciones de los procesos de negocio identificados en el sistema y por último la capa de componentes que es donde se implementan las funcionalidades utilizadas por la capa de procesos de negocio. Esta última, está formada por 3 subcapas, la capa de acceso a datos, que es donde se encuentran las clases para acceder a la base de datos, la capa lógica de negocio, en la cual se realizan todas las actividades del negocio y la capa de servicios que es donde se exponen los servicios a la capa de interfaces de usuario o de presentación.

Estas capas están distribuidas de manera tal que los componentes de una capa solo pueden acceder a los componentes de la capa inmediata inferior, de esta manera se reduce la dependencia entre capas, ya que las capas inferiores no tienen conocimiento sobre los detalles de las capas superiores.

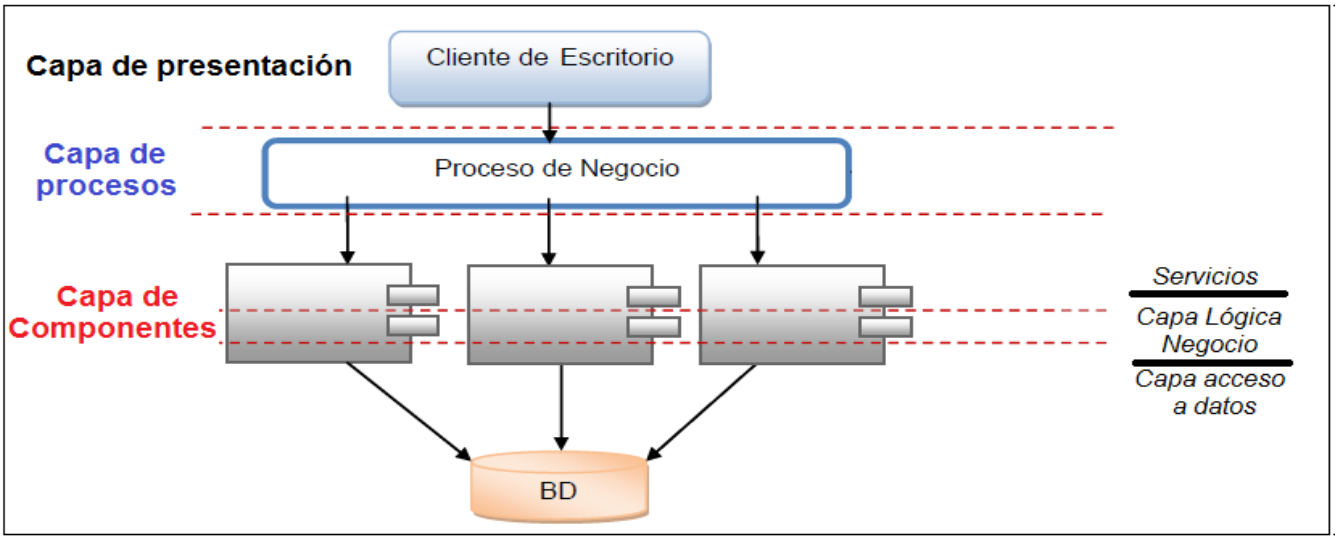


Figura 4: Estructura por un patrón de 3 capas

2.7. Descripción de los algoritmos no triviales a implementar

En este epígrafe se describen algoritmos que tienen una complejidad superior a la media, y a percepción de los implementadores, su explicación se hace necesaria para la mejor comprensión del código y constituye un aporte a futuras implementaciones de sistemas similares.

2.7.1. Registrar materias primas

Entradas:

- ✓ Nombre del lote.
- ✓ Tipo de documentos.
- ✓ No. de serie inicial.
- ✓ No. de serie final.
- ✓ Fecha de caducidad.
- ✓ Cantidad de documentos.
- ✓ Identificador del local (Almacén) destino.

Pasos (Cliente):

- ✓ Para crearlo primero se le asignan los datos descritos anteriormente en las entradas.
- ✓ Luego se presiona el botón recibir y se envía la información al servidor de aplicaciones.

Pasos (Servidor de aplicaciones):

- ✓ Se separan las letras y los números de la numeración de serie del lote.
- ✓ Se calcula la cantidad de documentos que contiene el lote de acuerdo al rango de números extraídos.
- ✓ Se registran los documentos en la base de datos con el número de serie correspondiente (la unión de las letras y la parte numérica).
- ✓ Se le adicionan a los datos del lote el estado de los documentos (virgen) y la fecha de creado.
- ✓ Se registra el lote en la base de datos.

Salida:

- ✓ Se muestra el mensaje “El lote fue registrado correctamente”.

Reglas:

- ✓ El nombre y la descripción del lote solo pueden presentar números y letras.
- ✓ No puede quedar ningún campo vacío.
- ✓ No debe existir un lote con el mismo nombre en el sistema.
- ✓ El local debe ser de tipo “Almacén”.

- ✓ La fecha de caducidad debe ser posterior a la fecha de registro del lote.

2.7.2. Asignar preimpreso

Entradas:

- ✓ Identificador del local (Personalización) destino.
- ✓ Lista de documentos.

Pasos (Cliente):

- ✓ Se introduce la cantidad de documentos que va a tener el nuevo lote.
- ✓ Se seleccionan los lotes de los cuales se van a extraer los documentos.
- ✓ Luego se presiona el botón asignar y se envía la información al servidor de aplicaciones.

Pasos (Servidor de aplicaciones):

- ✓ Se genera una cadena alfanumérica aleatoria para el nombre del lote, así como la fecha de ubicación y caducidad.
- ✓ Se ordenan los documentos de menos a mayor para obtener el primer número de serie del lote.
- ✓ Se registra el lote en la base de datos.
- ✓ Se actualizan los datos de los lotes de los cuales se extrajeron documentos para la creación del nuevo lote.
- ✓ Se eliminan de la base de datos los lotes que no poseen documentos.

Salida:

- ✓ Se muestra el mensaje “El lote fue asignado correctamente”.

Reglas:

- ✓ El local debe ser de tipo “Personalización”.
- ✓ Todos los documentos deben ser de un mismo tipo.

2.8. Estándares de codificación

Debido a que el lenguaje de programación seleccionado para el desarrollo del sistema fue Java, a continuación se detallan algunos aspectos en los que se quiere enfatizar a la hora de escribir el código en

dicho lenguaje. Muchos de los aspectos que se describirán a continuación se extrajeron del estándar de codificación que define SUN (u Oracle) para el uso de este lenguaje.

2.8.1. Principios generales de nomenclatura

Los nombres de cada uno de los elementos del programa, clase o interfaz deben ser significativos, su nombre debe explicar, siempre que sea posible, el uso o fin del elemento. La mayoría de los elementos se deben nombrar usando sustantivos (posiblemente compuestos) o formas verbales en imperativo. La forma de construir los nombres será colocando primero el verbo o sustantivo, seguido de cada uno de sus complementos con la primera letra en mayúscula. El idioma seleccionado para realizar la codificación será el inglés, únicamente cuando se tengan nombres o palabras claves en el negocio cuya traducción resulte engorrosa o poco intuitiva, se procederá a escribirlas en idioma español. En resumen, se guiará por la siguiente tabla:

Tipos de identificadores	Reglas para nombres	Ejemplos
Paquetes	El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel, actualmente **com**, **edu**, **gov**, **mil**, **net**, **org**, o uno de los códigos ingleses de dos letras que identifican cada país como se especifica en el" ISO Standard 3166", 1981. Los subsecuentes componentes del nombre del paquete variarán de acuerdo a las convenciones de nombres internas de cada organización, proyecto o módulo.	org.springframework.util com.apple.quicktime.v2 cu.vnz.mppre.pd
Clases	Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Se debe intentar mantener los nombres de las clases simples y descriptivos. Evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).	class Cliente class ImagenAnimada

Interfaces	Los nombres de las interfaces siguen la misma regla que las clases.	-
Métodos	Los nombres de los métodos deben ser verbos intuitivos que expresen que es lo que se quiere hacer, además empezarán siempre con minúscula. Cuando son compuestos, tendrán la primera letra del nombre en minúsculas, y la primera letra de las palabras siguientes en mayúsculas.	ejecutar(); realizarJugada();
Variables	Excepto las constantes, todas las variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en minúscula igual y mayúsculas las que le siguen. Los nombres de variables no deben empezar con los caracteres guion bajo "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje. Los nombres de las variables deben ser cortos pero con significado. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales. Nombres comunes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres.	int i; char c; float peso;
Constantes	Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un guión bajo ("_"). (Las constantes ANSI se deben evitar, para facilitar su depuración.)	static final float SIZE = 0.0;

Tabla 1: Nomenclatura.

2.8.2. Estándares de configuración para el *framework* Spring

Normas de configuración

En aras de lograr una mayor claridad durante la concepción 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 será como se muestra en la figura 5:

```
<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 5: Declaración de forma abreviada

Convenciones de nomenclatura

Siguiendo la misma filosofía que se utiliza 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[16]" 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*. En las figuras 6 y 7 se ilustran mejor lo que se quiere:

[16] Refiere un identificador.

```
<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 6: Declaración de los "id" de los beans

Con sus respectivas implementaciones:

```
public interface CiudadanoDao {
    ...
}

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

```
public class CiudadanoDaoImpl {
    ...
}

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

Figura 7: Declaración de las clases e interfaces correspondientes

2.9. Modelo de clases del diseño

2.9.1. Diagrama de clases del diseño

Los diagramas de clases del diseño son diagramas estáticos que muestran un grupo de clases, interfaces, atributos, así como la relación existente entre ellos. En las **figuras 8 y 9** se muestran los diagramas de clases del diseño con las clases más significativas para las capas de componente y proceso.

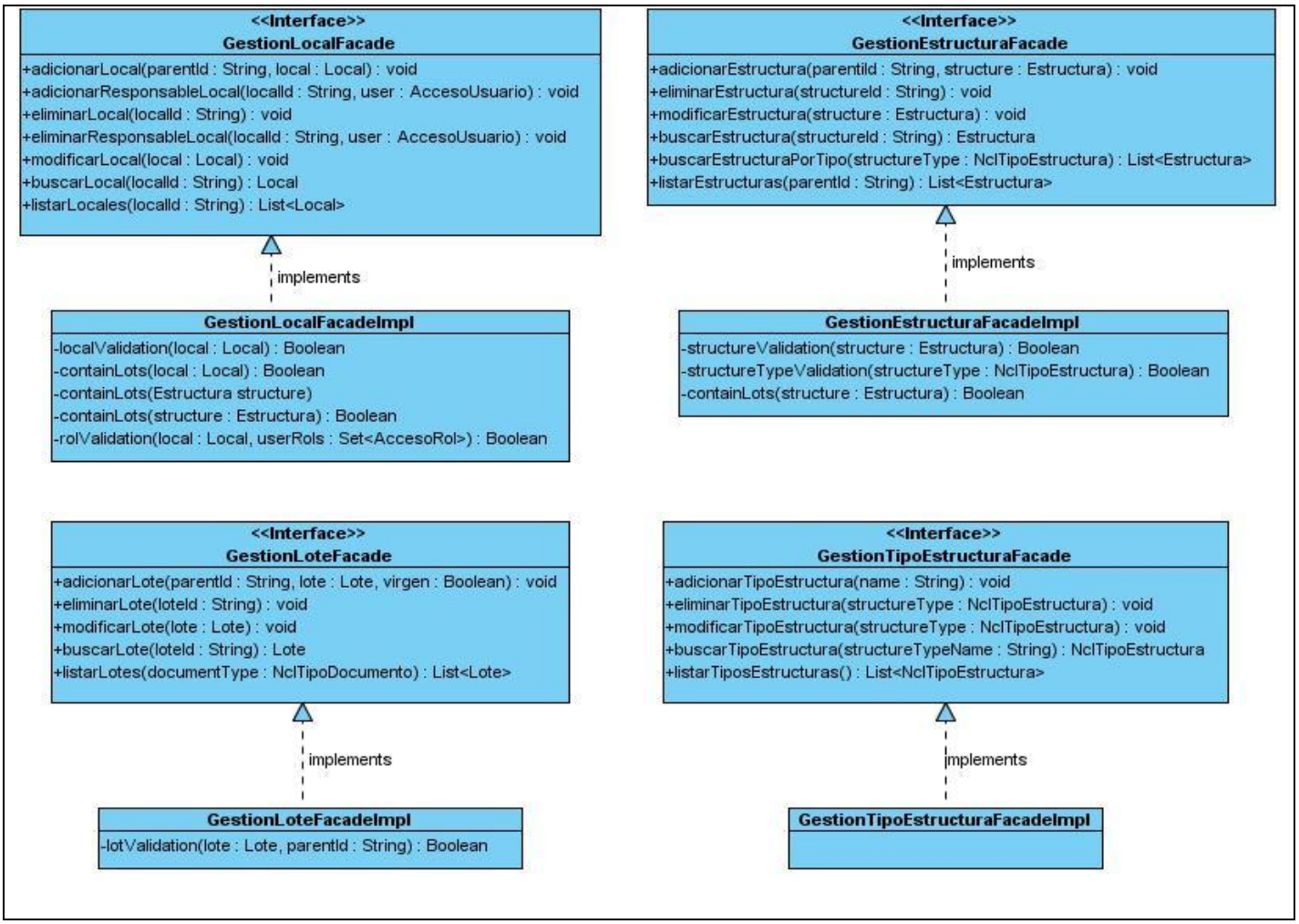


Figura 8: Diagrama de clases del diseño de la capa de componentes (Negocio)

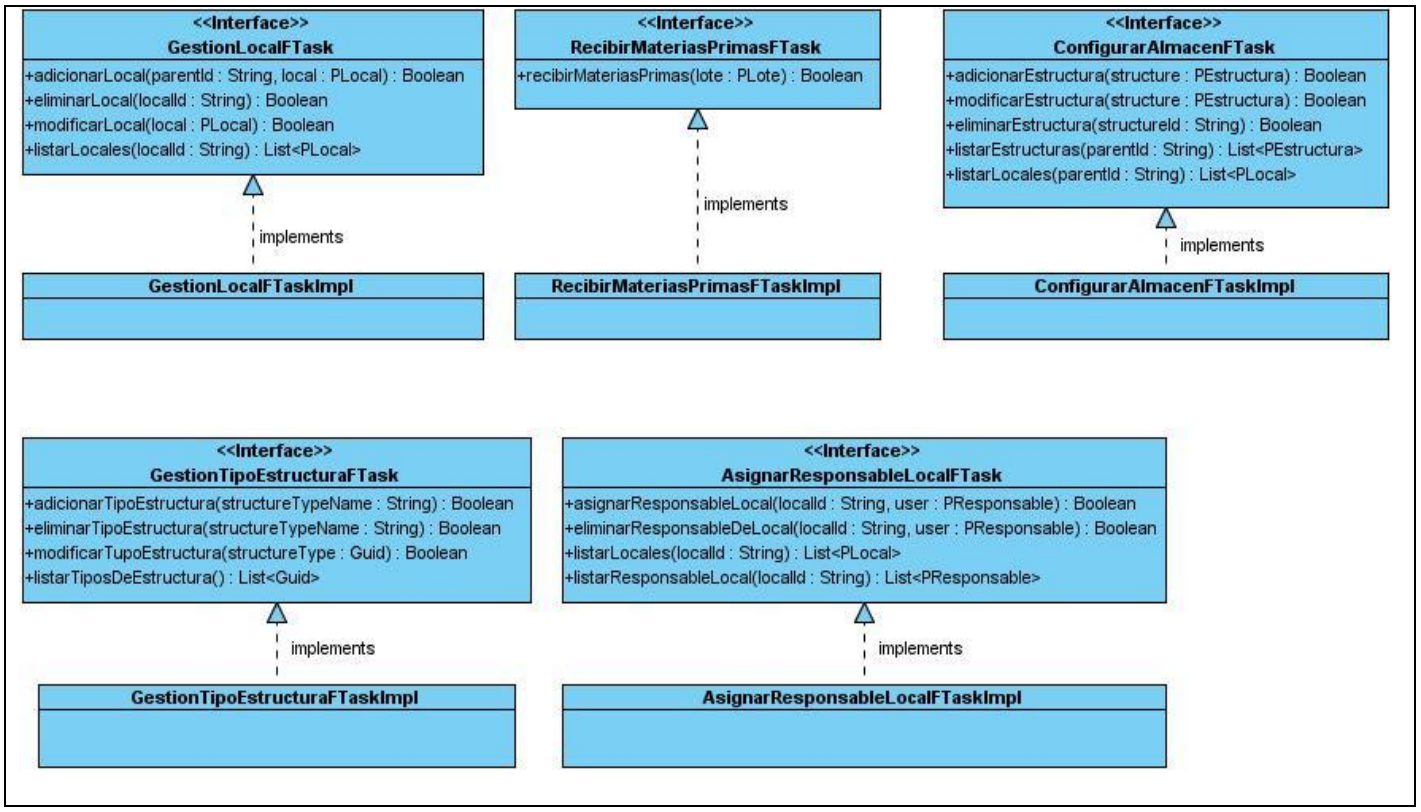


Figura 9: Diagrama de clases del diseño de la capa de procesos de negocio

2.9.2. Descripción de clases del diseño

A continuación se describe una de las clases más relevantes presentes en el diagrama de clases del diseño anterior. Para cada una de ellas se muestra su nombre, una breve descripción de su función. Se muestra además el objetivo y forma de funcionamiento de todas las operaciones contenidas por la misma, exceptuando aquellos métodos conocidos como "*get*" y "set". Las descripciones de las demás clases del diseño se encuentran registradas en los **anexos 4, 5 y 6**.

Nombre: GestionLocalFacade
Tipo de clase: Interfaz Está desarrollada para la gestión y manejo de todas las acciones básicas que se realizan sobre el objeto Local, es la que interactúa a través de DAOs[17] con la base de datos, por lo que es la que de alguna forma le da soporte a la capa de negocio correspondiente a sus funciones.

[17] DAO: Data Access Object (Objeto de Acceso a Datos)

Atributos:	**Tipo:**
-	-
Funciones:	
Nombre	adicionarLocal (String parentId, Local newLocal)
Descripción	Recibe como parámetro un identificador del local padre y un objeto de tipo Local, este método se encarga de buscar el local que contenga ese identificador y registra como hijo el nuevo local.
Nombre	adicionarRolLocal(String localId, List<String> rols)
Descripción	Recibe como parámetro una cadena y una lista de roles, este localiza el local que contenga dicha cadena como identificador y se le asignan los roles que recibe como parámetro.
Nombre	adicionarResponsableLocal (String localId, List<String> users)
Descripción	Recibe como parámetro una cadena y una lista de usuarios, este localiza el local que contenga dicha cadena como identificador y se le asignan los usuarios que recibe como parámetro.
Nombre	eliminarLocal (String localId)
Descripción	Busca el local que contenga como identificador la cadena recibida como parámetro y se encarga de eliminarlo en caso de que este pueda ser borrado de la Base de Datos.
Nombre	eliminarResponsableLocal (String localId, String user)
Descripción	Busca el local cuyo identificador concuerde con la cadena recibida como parámetros y elimina la relación que existe con el usuario introducido.
Nombre	modificarLocal (Local modifiedLocal)
Descripción	Se encarga de actualizar o modificar los datos del local recibido como parámetro.
Nombre	buscarLocal (String localId)
Descripción	Dada una cadena (recibida como parámetro) busca el local cuyo identificador coincida con la misma.
Nombre	listarLocales(String localId)
Descripción	Devuelve una lista de los locales cuyo padre posea como identificador la cadena recibida como parámetro.
Nombre	listarLocalesPorTipo(String localType)
Descripción	Devuelve una lista de los locales cuyo tipo de local coincida con la cadena recibida como parámetro.
Nombre	listarLocalesPorUsuario(String usuario)

Descripción	Devuelve una lista de los locales cuyo usuario responsable coincida con la cadena recibida como parámetro.
Nombre	listarLocalesCriterio(String localId, String user, String localType)
Descripción	Devuelve una lista de los locales cuyo usuario responsable, tipo de local e identificador del local padre coincidan con los recibidos como parámetro.
Nombre	listarTiposLocal()
Descripción	Devuelve una lista con los tipos de locales existentes en la base de datos.

Tabla 2: Descripción de la clase del diseño "GestionLocalFacade".

2.9.3. Estructura por paquetes de diseño

Estos diagramas se utilizan para describir la organización de los paquetes y los elementos que contienen. Además de mostrar las dependencias que existen entre cada una de estas agrupaciones, muestran cómo se dividen estos en agrupaciones lógicas.

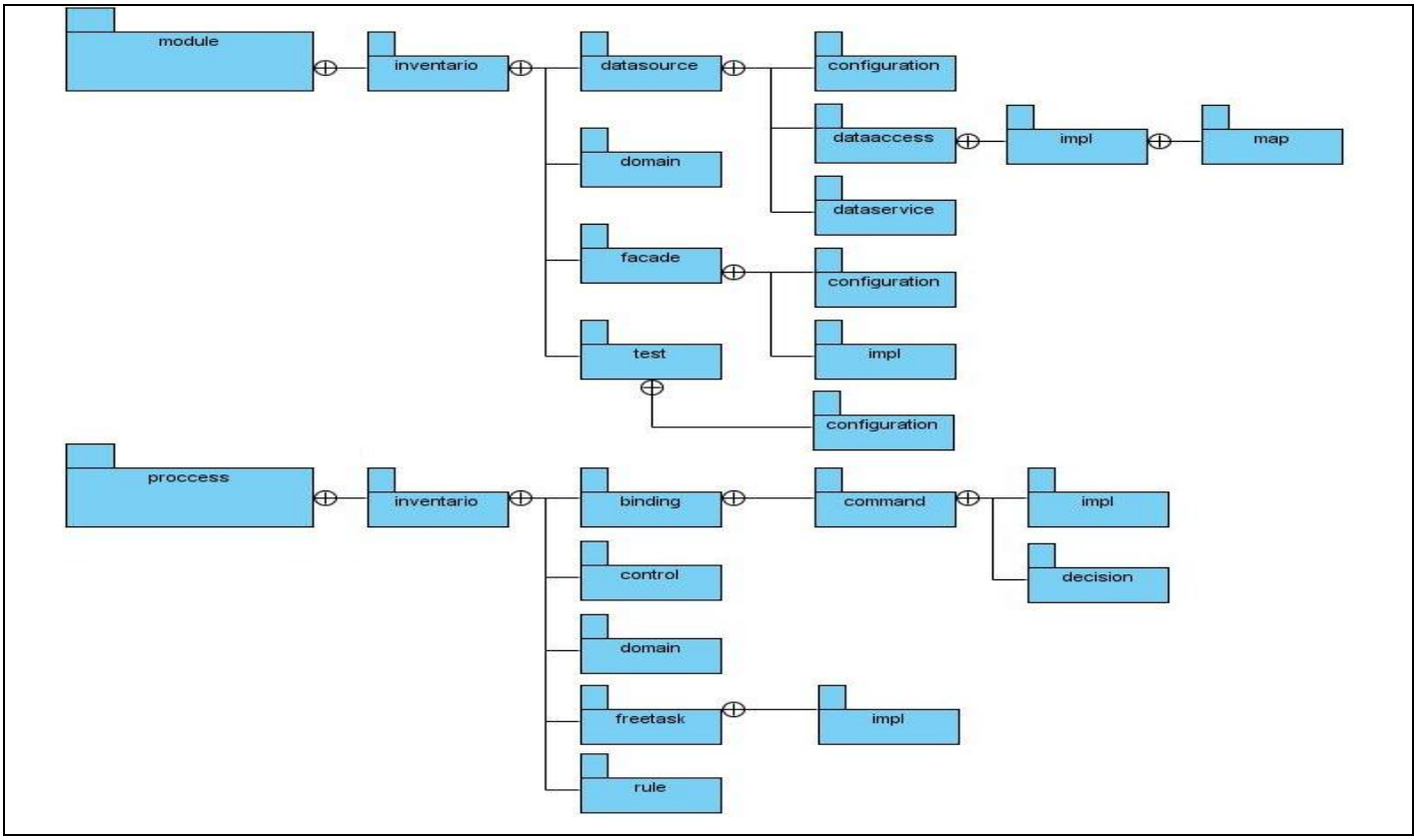


Figura 10: Diagrama por paquetes del módulo de control de inventario

A continuación se describen algunos de los principales paquetes que presenta el módulo de control de inventario. La completa descripción de los mismos y los elementos que estos contienen se encuentran en los anexos.

configuration: Contiene los archivos de configuración de los servicios web, conexión a la base de datos y demás propiedades que necesitan los componentes de negocio para su funcionamiento.

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

domain: Contiene todas las clases del dominio o persistentes con que se trabajan en el módulo.

map: Contiene todos los archivos de mapeo que genera hibernate correspondiente a cada clase del dominio.

2.10. Modelo de despliegue

El modelo de despliegue describe la distribución física del sistema, muestra la distribución de los componentes de *software* entre los distintos nodos de cómputo y permite comprender la correspondencia entre la arquitectura de *software* y la de *hardware*.

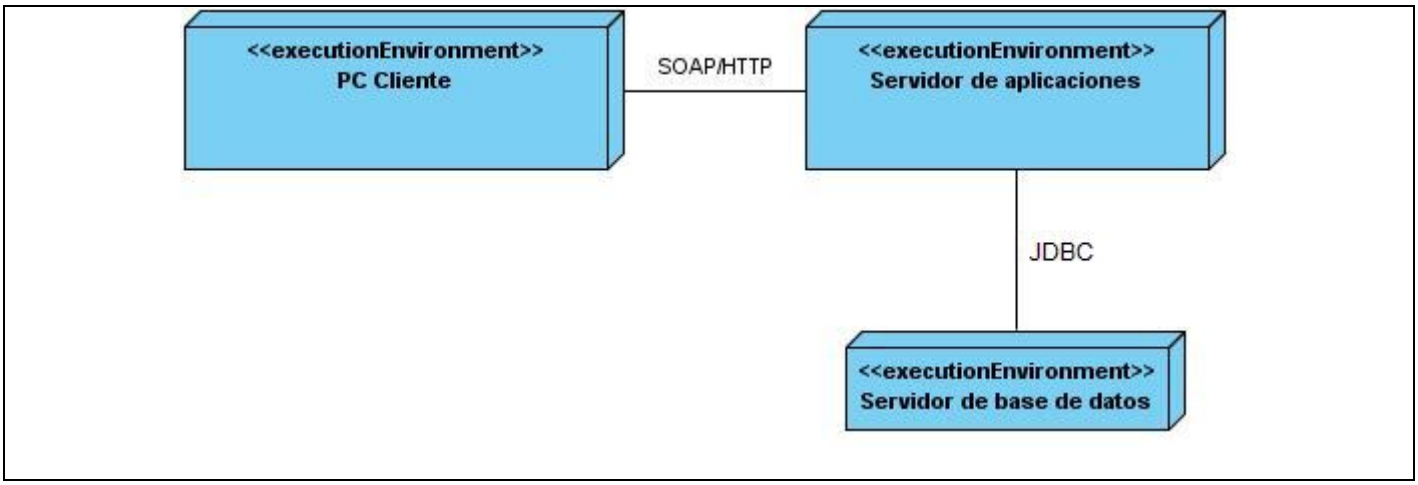


Figura 11: Diagrama de despliegue

En el nodo PC Cliente estará la aplicación con la cual interactúa el cliente, la cual consume los servicios del servidor de aplicaciones del Sistema de Emisión de Pasaportes Diplomáticos, de Servicios y

Acreditaciones mediante el protocolo SOAP/HTTP. En el nodo Servidor de aplicaciones se encuentran todos los componentes que implementas las funcionalidades del sistema, este a su vez, interactúa con el nodo Servidor de base de datos, utilizando el protocolo JDBC. Este protocolo de comunicación, es la API[18] de Java que utiliza la herramienta Hibernate para lograr la persistencia de los objetos. La aplicación debe tener acceso un controlador JDBC, el cual es que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API, JDBC y la base de datos real.

2.11. Conclusiones parciales

Se completó el análisis y diseño propuesto por el analista lo que dio como resultado un mejor entendimiento de las funcionalidades que se necesitan automatizar, así como los recursos que las mismas gestionan. Se generaron los artefactos faltantes en el análisis y diseño para argumentar y documentar de una manera más detallada la propuesta de solución y de forma tal que sea más factible la posterior implementación del módulo.

[18] Ver glosario de términos.

Capítulo 3: Implementación y pruebas

3.1. Introducción

Teniendo bien definida la propuesta de solución, con los requisitos y descripciones hechas por el analista y la elaboración de los demás artefactos generados, se puede realizar la implementación de la solución propuesta.

En el presente capítulo se documenta el diagrama de componentes realizado como artefacto que se genera en la fase de construcción. Además se define el modelo de pruebas que se llevará a cabo, se especifican y describen algunas de las pruebas a realizar, para así ir formando el programa global a medida que se comprueba cómo los distintos componentes interaccionan y se comunican libres de errores.

3.2. Diagrama de componentes

Los diagramas de componentes describen los elementos lógicos del sistema y sus relaciones además de mostrar las opciones que estos realizan, incluyendo código fuente, binario y ejecutable. Los componentes son la representación de todos los tipos de elementos de *software* que intervienen en la creación de aplicaciones informáticas, estos pueden ser archivos, paquetes, etc.

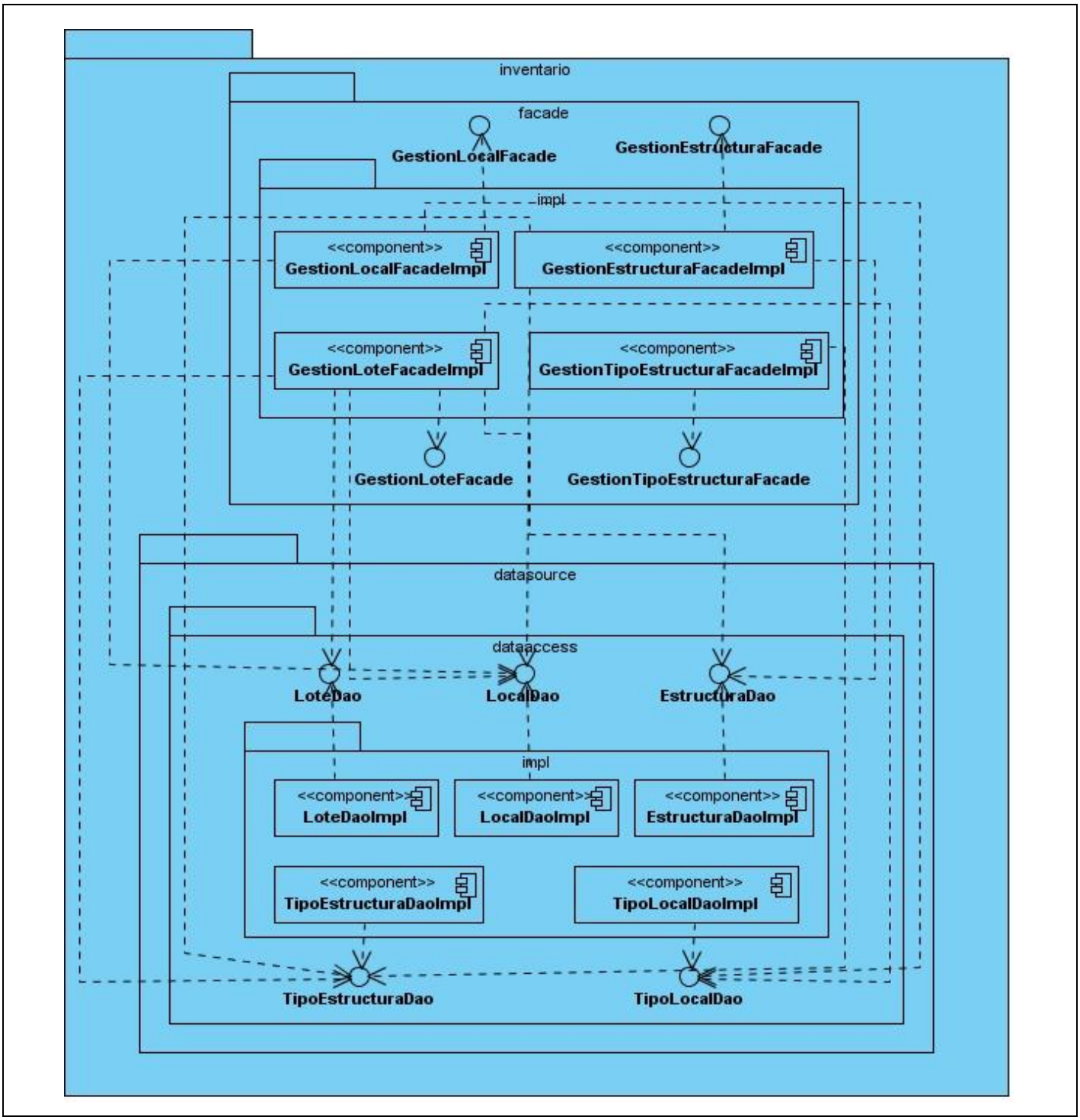


Figura 12: Diagrama de componentes del módulo de control de inventario

En la **figura 12** solo se muestran los componentes y paquetes más importantes de la capa de componentes, pues no se expresan en su totalidad los componentes y paquetes del módulo por la cantidad de componentes, interfaces y paquetes que presenta la estructuración del mismo. En el diagrama describe las relaciones que existen entre las principales clases de las subcapas de acceso a datos y capa de negocio, las cuales se relacionan a través de las interfaces de la capa de acceso a datos, para realizar las consultas a la base de datos, así como mantener la persistencia de los datos.

3.3. Métodos de prueba

El presente módulo fue sometido a diversas pruebas en cada una de las iteraciones por las que pasó. Pero primero es necesario conocer que las pruebas de *software* son los procesos que permiten verificar y revelar la calidad de un producto, y que si no encuentran deficiencias no son eficientes.

Pruebas de caja blanca: Son pruebas de *software* que se realizan sobre las funciones internas de un módulo. Se realizan primeramente sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas (integración).

Existen varias técnicas de prueba de caja blanca entre las que se pueden citar:

- Prueba de Condición: Método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- Prueba de Flujo de Datos: Selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- Prueba de Bucles: Técnica que se basa principalmente en la validez de las construcciones de bucles.
- Prueba del Camino Básico: Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática.

Los pasos que se siguen para aplicar esta técnica son:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- Se calcula la complejidad ciclomática del grafo.
- Se determina un conjunto básico de caminos independientes.
- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Se debe conocer que un camino es aquel para el cual puede efectuarse una traza a través del código desde el comienzo del programa hasta el final del mismo como ilustra la **figura 13**. Hay que tener en cuenta que dos caminos son diferentes si se ejecutan distintas sentencias o las mismas pero en diferente orden. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

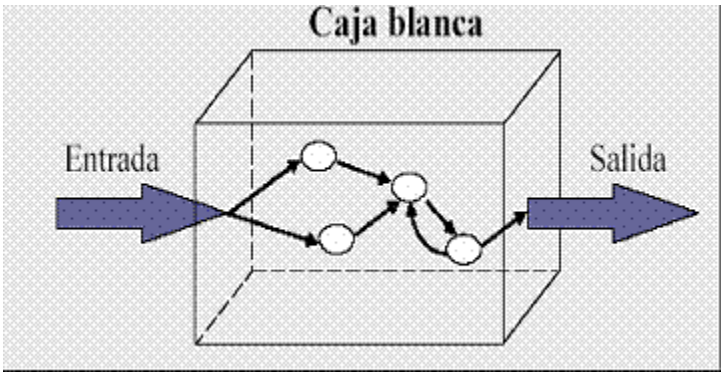


Figura 13: Pruebas de caja blanca

3.4. Niveles de prueba

Pruebas unitarias: Inician con las pruebas de cada módulo. Es una forma de probar el adecuado funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de estos funcione correctamente por separado. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el fragmento de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas, fomentan el cambio,

simplifica la integración, documenta el código, separa la interfaz del código y hace que los errores estén más acotados y sean fáciles de localizar.

Pruebas de integración: A partir del esquema del diseño, los módulos probados se vuelven a probar combinados para probar sus interfaces. Son aquellas que se realizan en el ámbito del desarrollo de *software* una vez que se han aprobado las pruebas unitarias. Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez. Consiste en realizar pruebas para verificar que un gran conjunto de partes de *software* funcionan juntos.

Todas las pruebas definidas serán con la ayuda de la herramienta JUnit, herramienta que se explica en el capítulo 1.

3.5. Casos de prueba

De acuerdo a la porción de código correspondiente a la funcionalidad eliminarResponsableLocal, perteneciente a la clase GestionLocalFacade, del componente inventario, módulo inventario, se le realizó la prueba de caja blanca:

```
public void eliminarResponsableLocal(String localId, String user) {                    1
        Local local = localDao.findCompleteLocal(localId);                              2
        AccesoUsuario accUser = usuarioDao.findUserByName(user);                        2
        if (local == null || user == null) {                                            3 y 4
                throw new RuntimeException("Usuario o local incorrecto");               5
        }
        if (local.getAccesoUsuarios().contains(accUser)) {                              6
                local.getAccesoUsuarios().remove(accUser);                              7
        } else {
                throw new RuntimeException("El usuario no es responsable de ese local");  8
        }
        localDao.update(local);                                                         9
}                                                                                       10
```

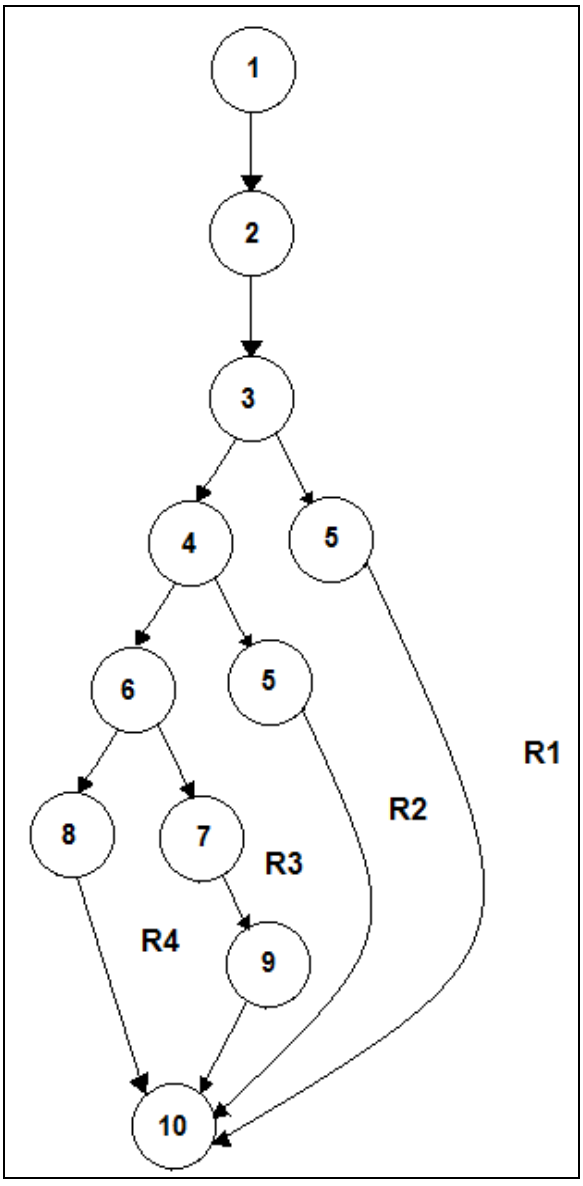


Figura 14: Grafo de caso de prueba Eliminar responsable de local

Complejidad ciclomática:

V (G): Número de regiones del grafo.

V (G) = A – N + 2

V (G) = P + 1

A: Número de aristas del grafo.

N: Número de nodos.

P: Número de nodos predicados.

V (G) = 4

Caminos: 1-2-3-5-10, 1-2-3-4-5-10, 1-2-3-4-6-7-9-10, 1-2-3-4-6-8-10.

Camino: 1-2-3-5-10.

Caso de prueba: Eliminar responsable de local.

Entrada: Recibe un identificador de local y un usuario.

Resultado: Se busca el local por su identificador y el responsable por su usuario, se verifica que el local sea nulo y se lanza una excepción.

Condiciones: Haber pasado o no por parámetro un identificador de local y un usuario.

Camino: 1-2-3-4-5-10.

Caso de prueba: Eliminar responsable de local.

Entrada: Recibe un identificador de local y un usuario.

Resultado: Se busca el local por su identificador y el responsable por su usuario, se verifica que el local no sea nulo, se verifica que el responsable sea nulo y se lanza una excepción.

Condiciones: Haber pasado o no por parámetro un identificador de local y un usuario.

Camino: 1-2-3-4-6-7-9-10.

Caso de prueba: Eliminar responsable de local.

Entrada: Recibe un identificador de local y un usuario.

Resultado: Se busca el local por su identificador y el responsable por su usuario, se verifica que el local no sea nulo, se verifica que el responsable no sea nulo, se verifica que el local contenga al responsable buscado, se elimina el responsable del local y se actualiza el local en la base de datos.

Condiciones: Haber pasado o no por parámetro un identificador de local y un usuario.

Camino: 1-2-3-4-6-7-9-10.

Caso de prueba: Eliminar responsable de local.

Entrada: Recibe un identificador de local y un usuario.

Resultado: Se busca el local por su identificador y el responsable por su usuario, se verifica que el local no sea nulo, se verifica que el responsable no sea nulo, se verifica que el local no contenga al responsable buscado y se lanza una excepción.
Condiciones: Haber pasado o no por parámetro un identificador de local y un usuario.

De acuerdo a la porción de código correspondiente a la funcionalidad **Destruir documentos**, perteneciente a la clase GestionLoteFacade, del componente inventario, módulo inventario, se le realizó la prueba de caja blanca:

```
public void destruirDocumentos(List<Lote> lots) {                          1
    if (lots.isEmpty()) {                                                  2
        throw new RuntimeException("No hay lote para eliminar");           3
    }
    List<DocumentoLote> docs = new ArrayList<DocumentoLote>();             4
    Date date = new Date();                                                4
    for (int i = 0; i < lots.size(); i++) {                                5
        docs = documentoLoteDao.listByLotId(lots.get(i).getPkLote());      6
        if (docs.isEmpty()) {                                              7
            throw new RuntimeException("Error. Lote ficticio");            8
        }
        for (int j = 0; j < docs.size(); j++) {                            9
            Documento doc = docs.get(i).getDocumento();                    10
            doc.setFechaDestruccion(date);                                 10
            documentoDao.update(doc);                                      10
        }
        loteDao.delete(lots.get(i));                                       11
    }
}                                                                          12
```

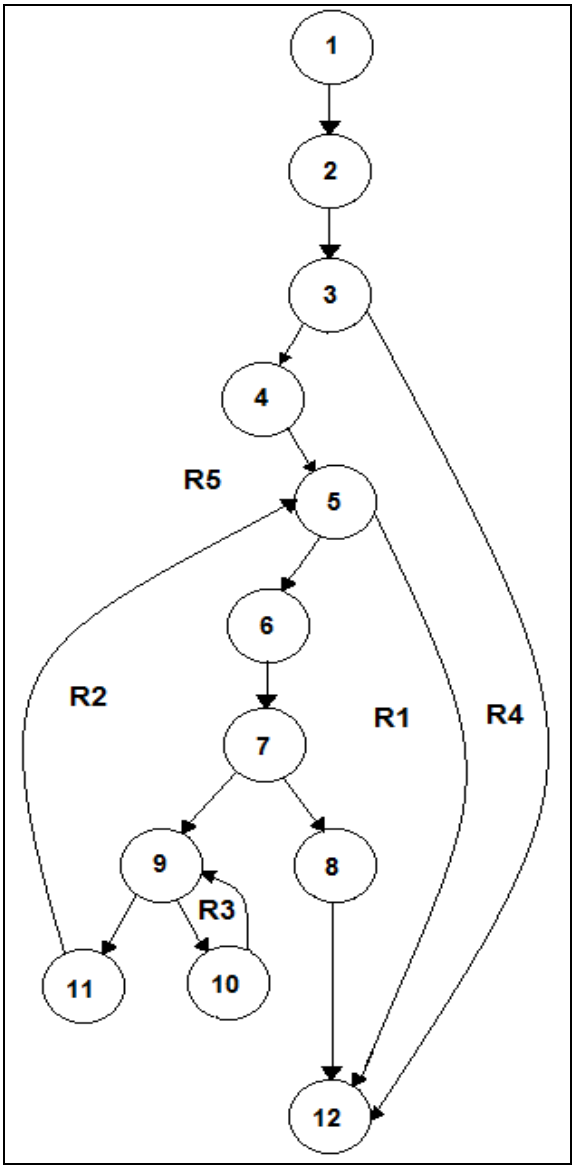


Figura 15: Grafo de caso de prueba Destruir documentos

Complejidad ciclomática:

V (G): Número de regiones del grafo.

V (G) = A – N + 2

V (G) = P + 1

A: Número de aristas del grafo.

N: Número de nodos.

P: Número de nodos predicados.

V (G) = 4

Caminos: 1-2-3-12, 1-2-3-4-5-12, 1-2-3-4-5-6-7-8-12, 1-2-3-4-5-6-7-9-11-5-12, 1-2-3-4-5-6-7-9-10-9-11-5-12.

Camino: 1-2-3-12.

Caso de prueba: Destruir documentos.

Entrada: Recibe una lista de lotes.

Resultado: Se verifica que la lista de lotes esté vacía y se lanza una excepción.

Condiciones: Haber pasado o no por parámetro una lista de lotes.

Camino: 1-2-3-4-5-12.

Caso de prueba: Destruir documentos.

Entrada: Recibe una lista de lotes.

Resultado: Se verifica que la lista de lotes no esté vacía, se registra la fecha y se crea una lista de documentos de lotes (documentos pertenecientes a un lote que representan documentos reales y poseen datos adicionales).

Condiciones: Haber pasado o no por parámetro una lista de lotes.

Camino: 1-2-3-4-5-6-7-8-12.

Caso de prueba: Destruir documentos.

Entrada: Recibe una lista de lotes.

Resultado: Se verifica que la lista de lotes no esté vacía, se registra la fecha y se crea una lista de documentos de lotes (documentos pertenecientes a un lote que representan documentos reales y poseen datos adicionales), se llena la lista de documentos de lotes con los pertenecientes al lote correspondiente en la iteración, se verifica que la lista esté vacía y se lanza una excepción.

Condiciones: Haber pasado o no por parámetro una lista de lotes.

Camino: 1-2-3-4-5-6-7-9-11-5-12.

Caso de prueba: Destruir documentos.

Entrada: Recibe una lista de lotes.

Resultado: Se verifica que la lista de lotes no esté vacía, se registra la fecha y se crea una lista de documentos de lotes (documentos pertenecientes a un lote que representan documentos reales y poseen datos adicionales), se llena la lista de documentos de lotes con los pertenecientes al lote correspondiente en la iteración, se verifica que la lista no esté vacía, y se elimina el lote correspondiente a la iteración.
Condiciones: Haber pasado o no por parámetro una lista de lotes.

Camino: 1-2-3-4-5-6-7-9-10-9-11-5-12.
Caso de prueba: Destruir documentos.
Entrada: Recibe una lista de lotes.
Resultado: Se verifica que la lista de lotes no esté vacía , se verifica que la lista de lotes no esté vacía, se registra la fecha y se crea una lista de documentos de lotes (documentos pertenecientes a un lote que representan documentos reales y poseen datos adicionales), se llena la lista de documentos de lotes con los pertenecientes al lote correspondiente en la iteración, se verifica que la lista no esté vacía, se actualizan los documentos con la fecha de destrucción de los mismos y se elimina el lote correspondiente a la iteración.
Condiciones: Haber pasado o no por parámetro una lista de lotes.

3.6. Pruebas unitarias y de integración

Aunque las pruebas de integración se realizan generalmente después de las pruebas unitarias, se hizo necesaria la unión de ambas en cada prueba, ya que para la comprobación de los métodos de cada capa implementada se utilizan métodos de clases ubicadas en capas inferiores. Estas pruebas fueron realizadas tanto a las funcionalidades de los componentes implementados, como a las que contienen las tareas definidas para la capa de procesos, comprobando el funcionamiento correcto de los mismos, así como la integración entre las capas. Para la realización de estas pruebas se utilizó la herramienta JUnit, explicada brevemente en el capítulo 1. Las **figuras 16 y 17** muestran pruebas unitarias y de integración realizadas a la capa lógica de negocio ubicada en la capa de componentes.

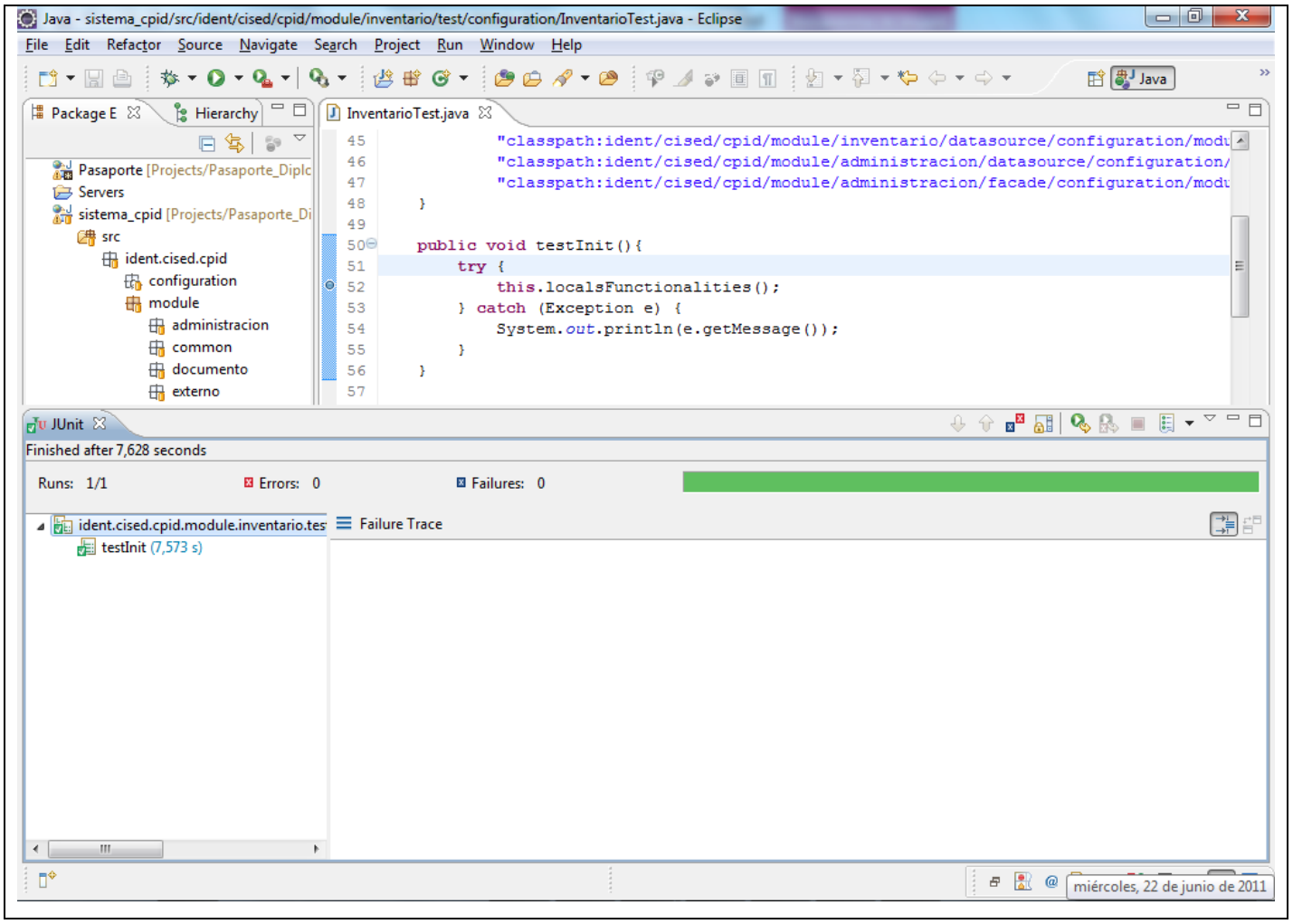


Figura 16: Prueba realizada exitosamente.

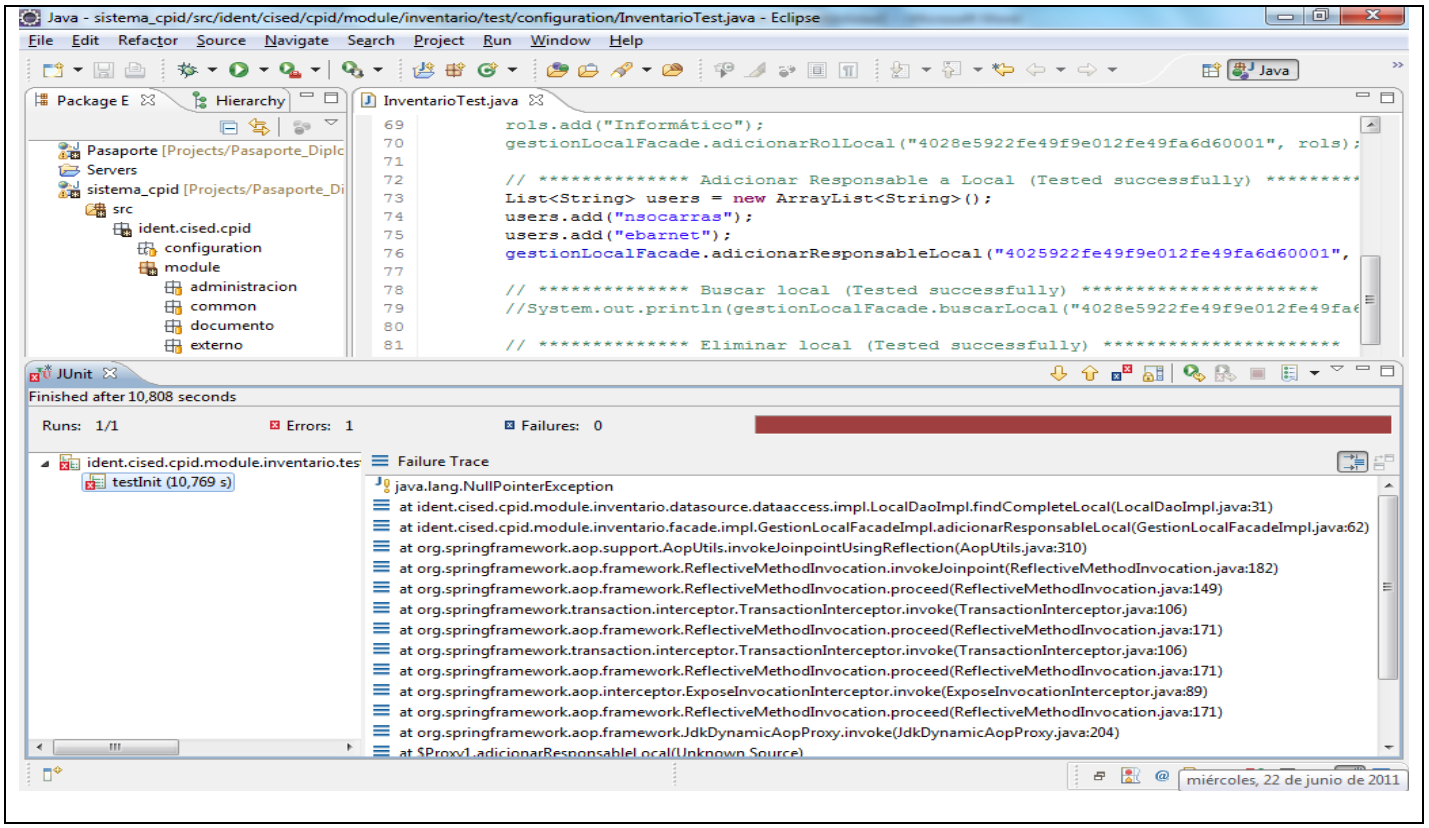


Figura 17: Prueba fallida realizada.

3.7. Conclusiones parciales

Se implementó la capa de componentes del módulo de control de inventario permitiendo esto, crear los procedimientos necesarios para acceder a la información que persiste en la base de datos, construir las funcionalidades que implementan la lógica del negocio y proveer una interfaz para utilizarlas.

Se implementó la capa de proceso del módulo de control de inventario posibilitando definición clara de las tareas que se ejecutan durante el control de inventario en el Sistema de Emisión de Pasaportes Diplomáticos, de Servicio y Acreditaciones.

Se aplicaron las pruebas de caja blanca, unitarias y de integración permitieron comprobar que la estructura lógica de los métodos es correcta, además de que posibilitaron la evaluación del comportamiento de las clases y los métodos implementados, así como la integración entre las capas implementadas.

Conclusiones generales

Se realizó una base teórica para entender el tanto el problema planteado como el ambiente en el cual se desenvuelve el mismo, así como un estado del arte sobre el tema tanto a nivel nacional como internacional y se determinó que era necesaria la implementación de un módulo para el control de inventario del subsistema de personalización.

Se utilizaron las herramientas definidas en el proyecto; Java como lenguaje de programación, Eclipse Galileo como entorno de desarrollo, Spring para trabajar el negocio, Hibernate para el acceso a datos, PostgreSQL como gestor de bases de datos y para guiar el proceso de desarrollo la metodología ágil FDD.

Se completó el análisis y diseño propuesto por el analista lo que dio como resultado un mejor entendimiento de las funcionalidades que se necesitan automatizar, así como los recursos que las mismas gestionan. Se generaron los artefactos faltantes en el análisis y diseño para argumentar y documentar de una manera más detallada la propuesta de solución y de forma tal que sea más factible la posterior implementación del módulo.

Se implementó la capa de componentes del módulo de control de inventario permitiendo esto, crear los procedimientos necesarios para acceder a la información que persiste en la base de datos, construir las funcionalidades que implementan la lógica del negocio y proveer una interfaz para utilizarlas.

Se implementó la capa de proceso del módulo de control de inventario posibilitando definición clara de las tareas que se ejecutan durante el control de inventario en el Sistema de Emisión de Pasaportes Diplomáticos, de Servicio y Acreditaciones.

Se aplicaron las pruebas de caja blanca, unitarias y de integración permitieron comprobar que la estructura lógica de los métodos es correcta, además de que posibilitaron la evaluación del comportamiento de las clases y los métodos implementados, así como la integración entre las capas implementadas.

Recomendaciones

Al finalizar la investigación y cumplir el objetivo general, en vista de mejorar el funcionamiento del sistema y con el fin de que se incrementen las funcionalidades brindadas para su mejor uso, se recomienda:

- ✓ Incorporar un modelo gráfico que indique como está estructurado físicamente un local, así como las estructuras que este contiene.
- ✓ Poner en funcionamiento el subsistema para que el módulo sea utilizado.

Bibliografía referenciada

1. Buenas Tareas. [En línea] 2010. [Citado el: 10 de diciembre de 2010.] http://www.buenastareas.com/ensayos/Inventario/191276.html.

2. Scribd. *Principios básicos para el Control de Inventario.* [En línea] abril de 2007. [Citado el: 12 de diciembre de 2010.] http://es.scribd.com/doc/2298564/Concepto-de-inventario.

3. Definición.de. [En línea] 2008-2011. [Citado el: 13 de diciemnbre de 2011.] http://definicion.de/documento/.

4. Definición ABC. [En línea] 2008-2011. [Citado el: 3 de enero de 2011.] http://www.definicionabc.com/general/pasaporte.php.

5. RedDominicana.com. [En línea] 2009. [Citado el: 5 de enero de 2011.] http://portal.reddominicana.com/foros/sabes-que-es-un-pasaporte-diplomatico.

6. Definición Tipos De Inventarios. [En línea] [Citado el: 8 de enero de 2011.] http://www.mitecnologico.com/Main/DefinicionTiposDeInventarios.

7. Capítulo 3: Spring, un framework de aplicación. [En línea] [Citado el: 16 de enero de 2011.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf.

8. Blog de WordPress.com. [En línea] 5 de julio de 2009. [Citado el: 18 de enero de 2011.] http://asaes.wordpress.com/2009/07/.

9. Rojo Capo. [En línea] 1 de 09 de 2010. [Citado el: 19 de enero de 2011.] http://rojocapo90.blogspot.com/2010/09/caracteristicas-del-lenguaje-java.html.

10. Scribd. [En línea] 2010. [Citado el: 19 de enero de 2011.] http://es.scribd.com/doc/57226129/Caracteristica-Del-Java.

11. The war of software. [En línea] noviembre de 2006. [Citado el: 21 de enero de 2011.] http://zoftwar.blogspot.com/2006/11/requisitos-no-funcionales-parte-1-con.html.

12. EcuRed. *Flujo de Trabajo Requerimiento.* [En línea] 5 de mayo de 2011. [Citado el: 1 de junio de 2011.] http://www.ecured.cu/index.php/Flujo_de_Trabajo_Requerimiento.

13. Arquitectura de Software. [En línea] 2008. [Citado el: 12 de febrero de 2011.] http://arquitecturasoftware.blogspot.com/.

Bibliografía consultada

S.Pressman, Roger. Ingeniería del software. Un enfoque práctico. 2001.

Jacobson, Booch y Rumbaugh. 2000. El Proceso Unificado de Desarrollo de software, UML y patrones.

Meléndrez, Edelsys Hernández. Metodología de la investigación. Como escribir una tesis. 2006.

Equipo de desarrollo de PostgreSQL. ABCdatos. [En línea] [Citado el: 14 de enero de 2011.] http://descargas.abcdatos.com/tutorial/descargarL6789.html.

Paradigm, Visual. 2010. Visual Paradigm. [En línea] 2010. http://www.visual-paradigm.com.

Inventario. [En línea] [Citado el: 2 de febrero de 2011.] http://es.mimi.hu/economia/inventario.html

Adictos al trabajo. JUnit 4. Pruebas de Software Java. [En línea] [Citado el: 10 de febrero de 2011.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=junit4

Pruebas de aceptación. [En línea] [Citado el: 2 de febrero de 2011.] http://www.lab.dit.upm.es/~lprg/material/apuntes/pruebas/aceptacion.htm

Glosario de términos

API: Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Applet: Componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo en un navegador web.

Axis2: Motor de servicios web basado en la arquitectura de Apache Axis.

Bytecode: es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto.

Componentes: Son todo aquel recurso desarrollado para un fin concreto y que puede formar solo, o junto

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

GRASP: Patrones Generales de Software para la Asignación de Responsabilidades.

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

HTML (*HyperText Markup Language*): Lenguaje de enmarcado.

HTTP (*Hyper Text Transfer Protocol*): Protocolo de transferencia de hipertexto usado en cada transacción de la Web.

JavaBean: Modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.

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

Plug-in: Un complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API. También se lo conoce como plug-in (del inglés "enchufable"), add-on (agregado), complemento, conector o extensión.

Servlet: Pequeño programa que corre en un servidor. Por lo general son aplicaciones Java que corren en un entorno de servidor web.

Socket: Interfaz de comunicación que ofrece un mecanismo de comunicación general entre dos procesos cualesquiera, que pertenezcan a un mismo sistema o a dos sistemas diferentes.

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

Web: Sistema para presentar información en Internet basado en hipertexto.

Anexos

Anexo 1: Operacionalización de las Variables.

Variable Conceptual	Dimensión	Indicadores	UM
Desarrollo del módulo control de inventario.	Factibilidad	Tiempo de desarrollo	Extenso Moderado Breve
		Costo	Costoso Moderado Breve
		Esfuerzo	Alto Moderado Despreciable

Tabla 3: Operacionalización de las variables, desarrollo del módulo de control de inventario.

Variable Conceptual	Dimensión	Indicadores	UM
Administrar los recursos materiales manejados por el subsistema de personalización.	Seguridad	Complejidad	Alta Media Baja
		Control	Alta Media Baja
		Organización del trabajo	Bueno Malo
		Importancia	Alta Media Baja
		Dependencia	Dependiente Independiente
		Tiempo de ejecución	Rápido Media Lento

Tabla 4: Operacionalización de las variables, administrar los recursos materiales manejados por el subsistema de personalización.

Anexo 2: El siguiente diagrama de clases del diseño solo muestra algunos componentes de la capa de proceso, debido a la cantidad de interfaces y clases que contiene esta capa, pues cada funcionalidad del módulo contiene al menos una interfaz y una clase que la implementa.

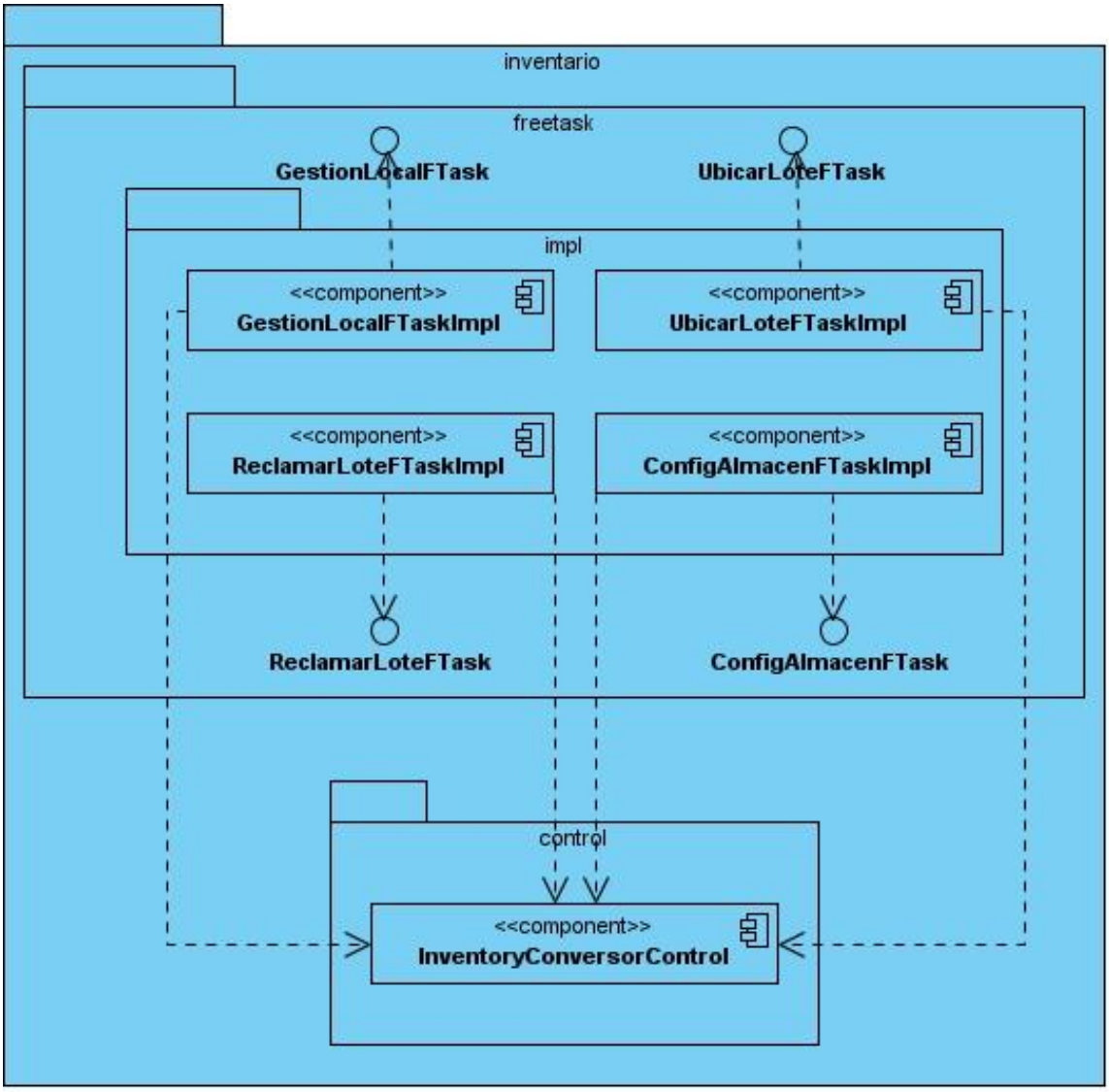


Figura 18: Diagrama de componentes para la capa de proceso

Anexo 3: El siguiente diagrama de clases de diseño muestra solo los componentes involucrados en la función “adicionar local”, debido a la cantidad de interfaces y clases que contienen todas las capas.

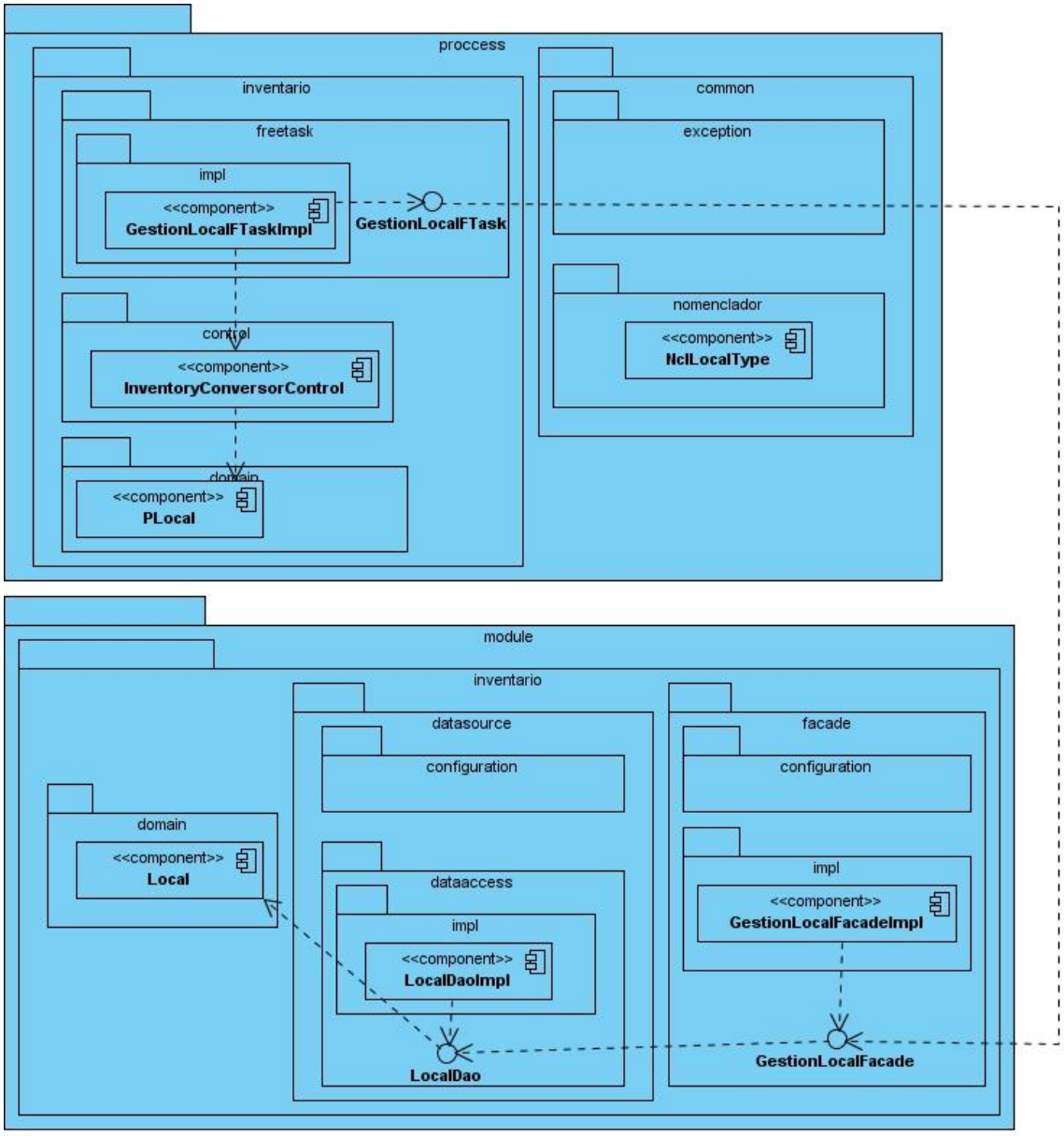


Figura 19: Diagrama de componentes general para funcionalidad con local

Anexo 4: Descripción de la clase del diseño "GestionLoteFacade".

Nombre: GestionLoteFacade	
Tipo de clase: Interfaz Está desarrollada para la gestión y manejo de todas las acciones básicas que se realizan sobre el objeto Lote y es la que interactúa a través de DAOs con la base de datos, por lo que es la que de alguna forma le da soporte a la capa de negocio correspondiente a sus funciones.	
Atributos:	**Tipo:**
-	-
Funciones:	
Nombre:	adicionarVirgen (Lote lote, String localId)
Descripción:	Recibe como parámetro un identificador del objeto padre (una estructura o local) y un Lote, este método se encarga de buscar el objeto que contenga ese identificador y registra dentro de este el lote recibido como parámetro.
Nombre:	asignarPreimpreso (String localId, Lote lot)
Descripción:	Recibe como parámetro un identificador del lugar destino (un local de tipo "Personalización") y un Lote, este método se encarga de buscar el objeto que contenga ese identificador y registra dentro de este el lote recibido como parámetro.
Nombre:	recibirPasaportesXSellar(List<DocumentoLote> docs, String ubicId)
Descripción:	Se encarga de crear un nuevo lote con los documentos recibido como parámetro en la ubicación (local o estructura) que contenga el identificador recibido como parámetro.
Nombre:	recibirDocumentos(List<Lote> lots, String ubicId)
Descripción:	Ubica los lotes recibidos como parámetro en la ubicación (local o estructura) que contenga como identificador la cadena recibida como parámetro.
Nombre:	destruirDocumentos(List<Lote> lots)
Descripción:	Se encarga de eliminar de la base de datos todos los lotes recibidos como parámetro.
Nombre:	listarLotes(String parentId)
Descripción:	Lista todos los lotes cuyo padre contenga el identificador recibido como parámetro.

Nombre:	listarLotes(String documentState, String documentType)
Descripción:	Lista todos los lotes cuyo padre contenga el identificador recibido como parámetro y sean del tipo especificado.
Nombre:	listarLotes(String localId, String documentState, String documentTypeName, boolean located)
Descripción:	Lista todos los lotes que se encuentren en el local raíz cuyo identificador coincida con el recibido como parámetro y tengan el estado y el tipo de documentos especificados.
Nombre:	listarDocumentos(String parentId, String documentState, String documentType)
Descripción:	Lista los documentos ubicados en los lotes dentro del local o estructura cuyo identificador coincida con el recibido como parámetro y sean del tipo y estén en el estado especificado.
Nombre:	listarDocumentosLote(String lotId)
Descripción:	Lista los documentos del lote con el identificador recibido como parámetro.
Nombre:	List<NclCausaReclamacion> listarCausasReclamacion()
Descripción:	Lista todas las causas de reclamación posibles.

Anexo 5: Descripción de la clase del diseño “GestionEstructuraFacade”.

Nombre: GestionEstructuraFacade	
Tipo de clase: Interfaz Está desarrollada para la gestión y manejo de todas las acciones básicas que se realizan sobre el objeto Lote y es la que interactúa a través de DAOs con la base de datos, por lo que es la que de alguna forma le da soporte a la capa de negocio correspondiente a sus funciones.	
Atributos:	**Tipo:**
-	-
Funciones:	
Nombre:	adicionarEstructura(String parentId, Estructura structure)
Descripción:	Recibe como parámetro un identificador del objeto padre (una estructura o local) y

	una estructura, este método se encarga de buscar el objeto que contenga ese identificador y registra dentro de este la estructura recibido como parámetro.
Nombre:	eliminarEstructura(String structureId)
Descripción:	Recibe como parámetro un identificador de una estructura y se encarga de eliminar la misma en caso de que esta pueda ser eliminada.
Nombre:	modificarEstructura(Estructura structure)
Descripción:	Se encarga de guardar en la base de datos la estructura recibida como parámetro.
Nombre:	buscarEstructura(String structureId)
Descripción:	Busca una estructura cuyo identificador coincida con la cadena recibida como parámetro.
Nombre:	listarEstructuras(String parentId)
Descripción:	Recibe como parámetro el identificador del objeto padre (una estructura o local), este método se encarga de buscar todas las estructuras cuyo padre posea el identificador recibido como parámetro.

Anexo 6: Descripción de la clase del diseño “GestionTipoEstructuraFacade”.

Nombre: GestionTipoEstructuraFacade	
Tipo de clase: Interfaz Está desarrollada para la gestión y manejo de todas las acciones básicas que se realizan sobre el objeto Lote y es la que interactúa a través de DAOs con la base de datos, por lo que es la que de alguna forma le da soporte a la capa de negocio correspondiente a sus funciones.	
Atributos:	**Tipo:**
-	-
Funciones:	
Nombre:	adicionarTipoEstructura(String structureTypeName)
Descripción:	Se crea un nuevo tipo de estructura cuyo nombre será el recibido como parámetro.
Nombre:	eliminarTipoEstructura(String structureName)
Descripción:	Se encarga de eliminar el tipo de estructura que posea el nombre recibido como parámetro.

Nombre:	modificarTipoEstructura(NclTipoEstructura structureType)
Descripción:	Se encarga de actualizar el tipo de estructura que posea el identificador del recibido como parámetro con los datos del mismo.
Nombre:	listarTiposDeEstructura()
Descripción:	Se encarga de listar todos los tipos de estructuras.
Nombre:	buscarTipoEstructura(String structureTypeName)
Descripción:	Se encarga de buscar un tipo de estructura que posea el nombre recibido como parámetro.