



JSJ Sport S.A.

Tienda deportiva

**Juan David Cubillos
Juan Piedrahita**



Índice general

I	Proyecto
1	Definición
1.1	Introducción
1.2	Problema
1.3	Objetivos
1.3.1	Objetivo General
1.3.2	Objetivos Específicos
1.4	Justificación
1.5	Alcance
1.5.1	Limitaciones
1.6	Presupuesto
2	Metodología
2.1	Introducción
2.2	Prototipo
2.3	Cronograma
II	Arquitectura y Diseño
3	Empresa
3.1	Introducción
3.2	Nombre

3.3	Misión	20
3.4	Visión	20
3.5	Objetivos	20
4	ADM	21
4.1	Introducción	21
4.2	Arquitectura base de una empresa	22
4.2.1	Puntos clave de ADM	22
4.2.2	Modelo de proceso ADM	23
4.3	ArchiMate	24
4.3.1	Capa de negocio	25
4.3.2	Capa de Aplicación	26
4.3.3	Capa de Tecnologías	27
4.3.4	Capa de Motivación	29
4.3.5	Capa de Implementación y Migración	30
4.3.6	Tabla de relaciones	30
4.4	Integración AMD y ArchiMate	32
5	Negocio	33
5.1	Introducción	33
5.2	Punto de Vista de Organización	34
5.2.1	Modelo	34
5.2.2	Caso de estudio	34
5.3	Punto de Vista de cooperación de Actor	35
5.3.1	Modelo	35
5.3.2	Caso de estudio	35
5.4	Punto de Vista de Función de Negocio	36
5.4.1	Modelo	36
5.4.2	Caso de estudio	36
5.5	Punto de Vista de Proceso de negocio	37
5.5.1	Modelo	37
5.5.2	Caso de estudio	37
5.6	Punto de Vista de Cooperación Proceso de negocio	38
5.6.1	Modelo	38
5.6.2	Caso de estudio	38
5.7	Punto de Vista de Producto	39
5.7.1	Modelo	39
5.7.2	Caso de estudio	39
6	Aplicación	41
6.1	Introducción	41
6.2	Punto de Vista de Comportamiento de Aplicación	42
6.2.1	Modelo	42
6.2.2	Caso de estudio	42

6.3	Punto de Vista de cooperación de Aplicación	43
6.3.1	Modelo	43
6.3.2	Caso de estudio	43
6.4	Punto de Vista de Estructura de aplicación	44
6.4.1	Modelo	44
6.4.2	Caso de estudio	44
6.5	Punto de Vista de Uso de Aplicación	45
6.5.1	Modelo	45
6.5.2	Caso de estudio	45
7	Tecnología	47
7.1	Introducción	47
7.2	Punto de Vista de Infraestructura	48
7.2.1	Modelo	48
7.2.2	Caso de estudio	48
7.3	Punto de Vista de Uso de Infraestructura	49
7.3.1	Modelo	49
7.3.2	Caso de estudio	49
7.4	Punto de Vista de Implementación y Despliegue	50
7.4.1	Modelo	50
7.4.2	Caso de estudio	50
7.5	Punto de Vista de Estructura de la Información	51
7.5.1	Modelo	51
7.5.2	Caso de estudio	51
7.6	Punto de Vista de Realización del Servicio	52
7.6.1	Modelo	52
7.6.2	Caso de estudio	52
7.7	Punto de Vista de Capas	53
7.7.1	Modelo	53
7.7.2	Caso de estudio	54
8	Motivación	55
8.1	Introducción	55
8.2	Punto de Vista de Stakeholder	56
8.2.1	Modelo	56
8.2.2	Caso de estudio	56
8.3	Punto de Vista de Realización de Objetivos	57
8.3.1	Modelo	57
8.3.2	Caso de Estudio	57
8.4	Punto de Vista de contribución de Objetivos	58
8.4.1	Modelo	58
8.4.2	Caso de Estudio	58
8.5	Punto de Vista de Principios	59
8.5.1	Modelo	59
8.5.2	Caso de Estudio	59

8.6	Punto de Vista de Realización de Requerimientos	60
8.6.1	Modelo	60
8.6.2	Caso de Estudio	60
8.7	Punto de Vista de Motivación	61
8.7.1	Modelo	61
8.7.2	Caso de estudio	61
9	Proyecto	63
9.1	Introducción	63
9.2	Punto de Vista de Proyecto	64
9.2.1	Modelo	64
9.2.2	Caso de estudio	64
9.3	Punto de Vista de Migración	65
9.3.1	Modelo	65
9.3.2	Caso de estudio	65
9.4	Punto de Vista de Implementación y Migración	66
9.4.1	Modelo	66
9.4.2	Caso de estudio	66
10	Diseño	67
10.1	Introducción	67
10.2	Requerimientos	68
10.2.1	Casos de Uso	70
10.3	Escenarios	71
10.3.1	Diagrama de secuencia	71
10.3.2	Diagrama de comunicación	73
10.4	Clases	75
10.5	Componentes	82
10.6	Nodos	83
10.7	Sistemas	84
10.8	Diagrama de Actividades	85
10.9	Estados	87
11	Patrones GoF	89
11.1	Introducción	89
11.2	Patrones Creacionales	90
11.2.1	Fabrica Abstracta	90
11.2.2	Método Fábrica	91
11.2.3	Singleton	92
11.3	Patrones Estructurales	95
11.3.1	Puente	95
11.3.2	Fachada	96

11.4 Patrones de Comportamiento	98
11.4.1 Comando	98
11.4.2 Iterador	99
11.4.3 Mediador	100
11.4.4 Momento	102
11.4.5 Observador	103
11.4.6 Estado	104
11.4.7 Estrategia	105

III

Conclusiones y Reflexiones

12 Conclusiones	109
13 Trabajos Futuros	111
13.1 Códigos	115
13.1.1 Códigos de las clases del Patrón Estado	115
13.1.2 Códigos de las clases del Patrón Singleton	119
13.1.3 Clase Cargador del paquete Utilidades	120
13.1.4 Clase Cableado del componente central	121
13.2 Desarrollo en Eclipse	123
13.2.1 Estructura Proyecto	123
13.2.2 Estructura del API	123
13.3 Prototipo pagina web	124



1. Definición

1.1 Introducción

En el mundo actual las pequeñas empresas se enfrentan día a día en el mercado, compiten por ganar prestigio y aumentar sus ganancias. Uno de los mayores problemas que se ven obligados a enfrentar estas pequeñas empresas es el de competir con grandes multinacionales, debido a que estas las dejan en una posición muy desfavorable a nivel de impacto y prestigio social debido a que cuentan con aliados que hacen que sus servicios sean más baratos e incluso mejores.

La mejor forma de lidiar con este problema es buscar siempre una innovación, una huella, un referente que haga que la empresa sea reconocida, es por esto que las empresas buscan abrir sus mercados y llegar al público de diferentes maneras. Muchas empresas aprovechan el auge de las redes sociales y el desarrollo del comercio web como una herramienta que le permite alcanzar nuevos mercados y de esta forma llegar a nuevos clientes potenciales.

1.2 Problema

Debido a que muchas de las pequeñas empresas no cuentan con los medios necesarios en lo que se refiere a conocimientos y personal para poder empezar a comercializar sus productos por medio de una plataforma web se ven obligados a tercerizar estos servicios de forma que puedan llevarse a cabo sus objetivos.

Muchas de las empresas que se dedican a tercerizar servicios web no tienen en cuenta todas las necesidades de los clientes y solo se preocupan por realizar una plataforma web en la que se puedan mostrar y vender productos, se deja de lado la necesidad de los clientes de tener un control completo de los procesos de compra y venta e incluso de asesoría a sus clientes, es por eso que se hace necesario una empresa que se dedique a apoyar a este tipo de empresas ayudándolas a impulsarse en el mercado basándose en las nuevas tecnologías sin dejar de lado cada una de las necesidades de los clientes.

1.3 Objetivos

1.3.1 Objetivo General

Brindar soluciones de tercerización de servicios para empresas que deseen expandir sus alcances en un entorno web con ayuda de las tecnologías emergentes.

1.3.2 Objetivos Específicos

- Cumplir cada una de las necesidades de los clientes de forma que se plasme en un entorno web la forma en que prestan sus servicios.
- Usar tecnologías de vanguardia que permitan a nuestros clientes estar un paso adelante en el mercado actual.
- Hacer del cliente una parte vital del equipo de trabajo durante el desarrollo del proyecto para que este tenga palabra y exprese sus necesidades.
- Brindar la máxima calidad posible a nuestros clientes por medio de productos altamente eficientes y con un excelente soporte.

1.4 Justificación

Debido a la falta de preocupación de las diferentes empresas de tercerización de servicios web por las verdaderas necesidades del cliente cuando este intenta expandir su mercado por medio de un servicio o de una plataforma web que le permita llegar a diferentes públicos y expandir sus alcances, se hace necesaria una empresa que apoye al cliente y le brinde la asesoría requerida de forma que este se sienta incluido durante todo el proceso.

El desarrollo de una empresa que cumpla con estas características es de gran importancia para cada uno de los clientes que contratan sus servicios y quedan satisfechos con los productos que obtienen, su calidad y el proceso que se llevo a cabo para obtenerlos puesto que el se vio involucrado en este proceso y el producto se diseño conforme a sus necesidades y con su apoyo constante; pero no solo es importante por esto sino que también es importante para ayudar al desarrollo general de la economía de los diferentes sectores y de la región al contribuir a que se abran nuevas formas de mercado y los productos y servicios lleguen a públicos a los que antes no se tenia alcance, es decir que esta empresa también sera una importante herramienta para el desarrollo.

1.5 Alcance

Una vez sea definido el objeto de estudio, se planteará el diseño de una plataforma web de tipo e-commerce que le permita mostrar su producto y comercializarlo, la plataforma web contara inicialmente con las siguientes características:

1. Información de contacto con la tienda.
2. Muestra de los artículos y la cantidad disponible en bodega.
3. Plataforma de pago en línea.
4. Recibo de compra exitosa virtual.

1.5.1 Limitaciones

El principal obstáculo de la realización de este proyecto es el tiempo de desarrollo para el mismo, por lo tanto para el primer prototipo se manejará un servicio de almacenamiento en base de datos local y no se creará un modulo que permita dar asesoría al cliente de la tienda en tiempo real.

1.6 Presupuesto

Cuadro 1.1: Presupuesto

RECURSOS MATERIALES				
Recurso	Descripción	Cantidad	Valor Unidad	Valor Total
Computadores Portátiles	Procesador: Intel Core i5 Memoria RAM:4gb DD:500 GB Pantalla 14”	1	\$1'500.000	\$1'500.000
	Procesador: Intel Core i7 Memoria RAM: 6gb DD: 1TB Pantalla 14”	1	\$2'000.000	\$2'000.000
Elementos de Red	Internet	4 meses	\$60.000	\$240.000
licencias	Java	2	\$0	\$0
	MySQL	2	\$0	\$0
	Eclipse	2	\$0	\$0
	TeXstudio	2	\$0	\$0
Centro de Desarrollo y oficina	Espacio fisico donde se llevará a cabo todo el desarrollo del proyecto	4 meses	\$1'000.000	\$4'000.000
Equipo Oficina	Escritorios	4	\$300.000	\$1'200.000
	Sillas	4	\$50.000	\$200.000
	Scanner e Impresora	1	\$150.000	\$150.000
	Papeleria y utiles de Oficina	1	\$400.000	\$400.000
Recursos Humanos				
Desarrolladores	Desarrollo del proyecto: análisis de requerimientos consecuentes con los objetivos, restricciones y alcances planteados y codificación y pruebas	2	\$12'000.000	\$24'000.000
Total del Proyecto				\$33'690.000



2. Metodología

2.1 Introducción

La finalidad de definir desde el principio una metodología y hacer uso de esta durante todo el desarrollo es la de hacer más eficaz el proceso de desarrollo del producto y lograr una alta calidad de forma que sea costeable tanto para el equipo de trabajo como para el cliente mismo, la metodología también define las reglas de trabajo para el grupo que llevará a cabo el desarrollo, las actividades y los procesos que este grupo debe realizar y la forma en que debe realizarlos.

Para este caso en específico se busca una metodología de trabajo que principalmente incluya al cliente como una de las partes fundamentales del desarrollo y permita al equipo estar en constante comunicación con este, la metodología debe permitir atender requerimientos emergentes y además la metodología escogida debe permitir mostrar al cliente los avances desarrollados en un período de tiempo de forma que este pueda hacer comentarios y exprese su satisfacción o informe en caso de que no esté conforme con algo.

2.2 Prototipo

Para el desarrollo de la plataforma web, se utilizará la metodología "Prototipo para desarrollo del software", la cual consta de una fase de requerimientos y de diseño con la creación de un subproducto, es esto lo que llamamos prototipo ya que es una fase temprana del mismo para ver su comportamiento, posteriormente se realiza una fase de implementación, una pruebas y una final de mantenimiento.

Además de esto se escoge esta metodología puesto que esta permite al cliente ser parte del proceso de desarrollo viendo el avance que este lleva y haciendo las observaciones pertinentes desde su punto de vista, de forma que se crea un dialogo entre el equipo de desarrollo y el cliente que permite establecer mejor las funcionalidades que se espera tenga el producto final y los requisitos que debe cumplir. Esto mejora la calidad del producto asegurando que este cumpla con las expectativas del cliente.

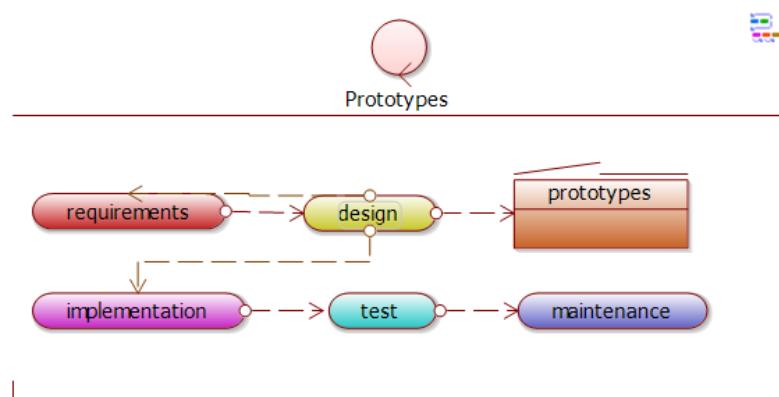


Figura 2.1: Proceso de desarrollo de software: Prototipo

2.3 Cronograma

Teniendo claro la metodología a utilizar, se procede a organizar el tiempo requerido para realizar el desarrollo del proyecto, teniendo en cuenta la cantidad de semanas destinadas para el desarrollo completo del prototipo y la documentación pertinente.

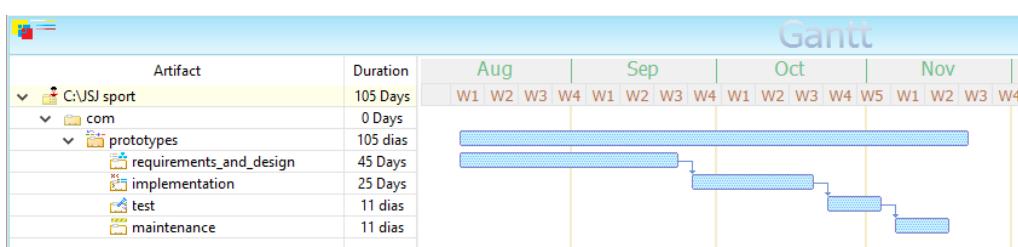


Figura 2.2: Proceso de desarrollo de software: Prototipo



3. Empresa

3.1 Introducción

En esta sección se presenta el objeto de estudio específico de nuestro proyecto, la empresa JSJ Sport S.A. fue escogida debido a que cumple con cada uno de los requisitos necesarios para postularse como una empresa que contrataría y consumiría los servicios de tercerización ofrecidos. JSJ Sport S.A. es una empresa de Bogotá que se dedica al comercio de todo tipo de productos deportivos, no solo para personas que practiquen un deporte como ocio sino que también para personas que se dediquen a una disciplina o sean deportistas de alto rendimiento. Lo que hace especial a JSJ Sport S.A. es que es una empresa Colombiana que manufactura sus productos en el interior del país y que tiene como principio el usar materias primas que sean estrictamente nacionales, esto con el fin de apoyar el producto nacional y a la vez confeccionar productos que sean de calidad incluso superior a los de marcas ya internacionales.

La empresa a pesar de contar con reconocimiento y prestigio local desea ampliar sus alcances de mercado de forma que llegue a un público mayor y que este pueda conocer la variedad y calidad de sus productos, esto no solo para el país de Colombia sino que también para muchos otros rincones del mundo.

3.2 Nombre

JSJ Sport S.A.

3.3 Misión

JSJ Sport S.A. brinda soluciones en el campo del deporte a nivel nacional e internacional a través de la innovación, servicio y calidad de los productos que ofrecemos, nuestros recursos están destinados a contribuir con el desarrollo de la salud física y mental de cada uno de nuestros clientes durante sus entrenamientos deportivos.

3.4 Visión

JSJ una empresa líder en el mercado de implementos y artículos deportivos que busca posicionamiento en Colombia, brindando productos de alta calidad para la práctica del deporte tanto a nivel profesional como recreativo, ofreciendo soluciones para las distintas necesidades deportivas de nuestros clientes, estamos comprometidos a apoyar y mejorar la salud mental y física de nuestro público por medio de nuestros productos.

3.5 Objetivos

- Comercializar productos deportivos a la medida de las necesidades de cada cliente.
- Contar con un catalogo amplio de productos deportivos de cualquier disciplina considerada como un deporte.
- Asesorar al cliente para que escoja el producto que más le beneficie.



4. ADM

4.1 Introducción

TOGAF ADM (Architecture Development Method) forma el centro de TOGAF. ADM es el resultado de contribuciones continuas de un gran número de arquitecturas, también describe un método para desarrollar y administrar el ciclo de vida de una arquitectura empresarial, ademas se apoya en muchos de los elementos de TOGAF y recursos de otras arquitecturas que están disponibles con el fin de conocer el negocio y las TI que de una organización.

El framework de TOGAF se complementa de Archimate puesto que este proporciona un conjunto independiente de conceptos, incluyendo una representación gráfica, que ayuda a crear un modelo coherente e integrado "por debajo de la linea de flotación" que se puede representar en forma de vistas TOGAF, es por eso que en el desarrollo de este capítulo se explicara el uso de ADM, sus puntos claves y las integración que se da con Archimate desde las diferentes capas que este tiene.

4.2 Arquitectura base de una empresa

ADM es útil para describir la Arquitectura Base de una empresa. Los requisitos empresariales que se tienen se pueden utilizar para identificar las definiciones y selecciones necesarias en la base de la arquitectura. Estos requisitos pueden ser un conjunto de modelos comunes reutilizables, políticas y definiciones de gobernabilidad, o incluso pueden ser algo más específico como selecciones tecnológicas sobresaliente. Al realizar la descripción de la Arquitectura Base se sigue principios similares a los de una arquitectura empresarial, con la diferencia de que los requisitos para toda una empresa se limitan a las preocupaciones generales y, por tanto, es menos completo que para una parte de la empresa específica.

4.2.1 Puntos clave de ADM

- ADM es iterativo, a lo largo de todo el proceso, entre las fases, y dentro de las fases . Para cada interacción de la ADM, una nueva decisión debe ser tomada en base a:
 - La amplitud de la cobertura de la empresa que se define.
 - El nivel de detalle que se define.
 - La extensión del período de tiempo destinado, incluyendo el número y la extensión de los períodos de tiempo intermedios.
 - Los activos de arquitectura para ser aprovechados, incluyendo:
 - Activos creados en versiones anteriores del ciclo de ADM dentro de la empresa.
 - Activos disponibles en otras partes de la industria (otros marcos, modelos de sistemas, modelos verticales de la industria, etc.).
- Las decisiones tomadas en cada fase se deben basar en una evaluación práctica de los recursos y la disponibilidad de competencias, y en el valor que realmente se puede esperar recibir de la empresa en el ámbito elegido del trabajo de la arquitectura.
- Como un método genérico, ADM está destinado a ser utilizado por empresas en una amplia variedad de diferentes zonas geográficas y aplicado en diferentes tipos sectores / industria vertical. Como tal, puede ser, pero no necesariamente tiene que ser, adaptado a las necesidades específicas.

4.2.2 Modelo de proceso ADM

Es importante recalcar que a lo largo del ciclo ADM, es necesario que se evaluen los resultados contra las expectativas originales, esto tanto para el ciclo ADM completo como para cada fase particular del proceso. [14]

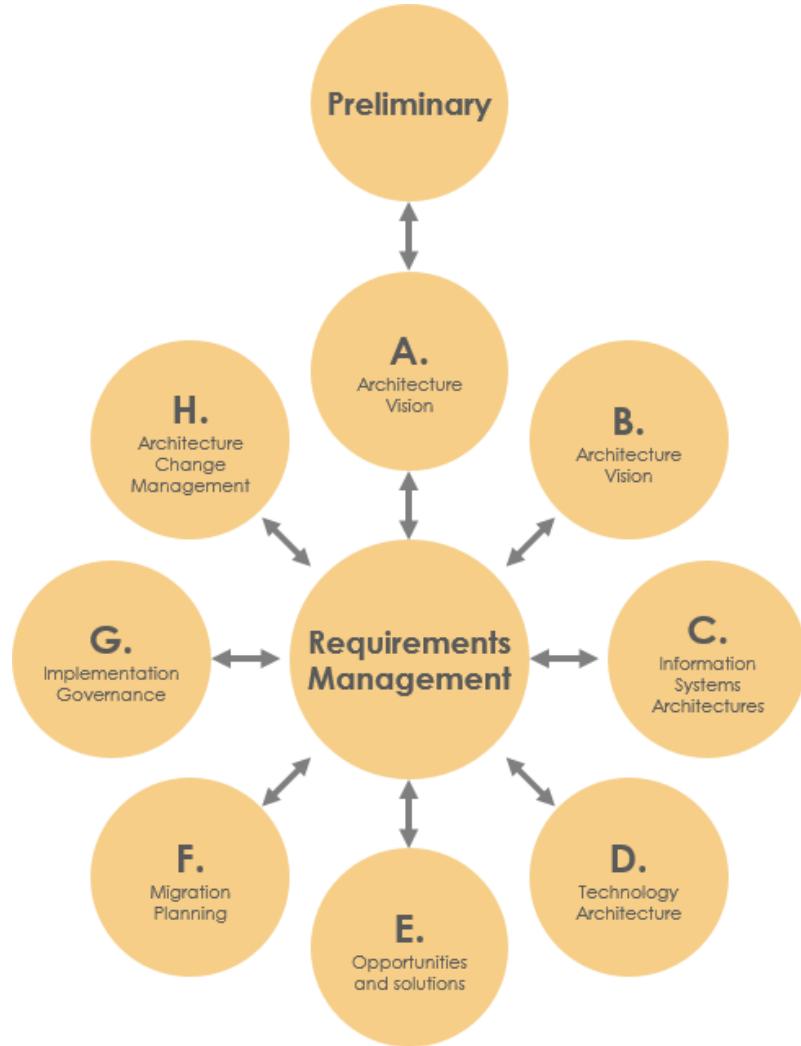


Figura 4.1: Estructura base del Modelo de Proceso de ADM

4.3 ArchiMate

ArchiMate es un lenguaje de modelado de arquitectura empresarial abierto e independiente que soporta la descripción, análisis y visualización de las relaciones entre los diferentes dominios de negocios de una forma no ambigua, es decir, la especificación de ArchiMate define un lenguaje común para describir la construcción y operación de procesos de negocios, estructuras organizacionales, flujos de información, sistemas de TI e infraestructura técnica. Esta visión ayuda a las partes interesadas a diseñar, evaluar y comunicar las consecuencias de las decisiones y cambios dentro y entre los dominios de negocio.

Con su versión más reciente ArchiMate 3.0 de 2016, se puede modelar la empresa a un nivel estratégico, como capacidad, recursos y resultados. También se incluye el apoyo para modelar el mundo físico de materiales y equipos.

ArchiMate se divide en capas como se puede observar en la siguiente figura:



Figura 4.2: Estructura por capas de ArchiMate [15]

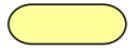
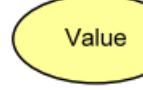
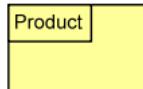
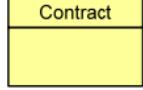
A continuación se van a mostrar los conceptos de las diferentes capas que se utilizan para el modelamiento en ArchiMate [12], dentro de las que se encuentran Capa de Negocio, Capa de Aplicación, Capa de Tecnologías, Capa de Motivación y Capa de Migración, además de esto también se mostrara la tabla de Relaciones.

4.3.1 Capa de negocio

Se refiere a los procesos de negocio, servicios, funciones y eventos ce cada una de las unidades de negocio. Esta capa ofrece productos y servicios a clientes externos, que se realizan en la organización mediante procesos de negocio realizados por actores y roles empresariales.

TABLA DE CONCEPTOS CAPA DE NEGOCIO

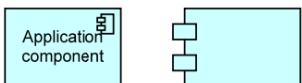
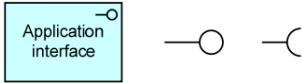
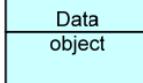
CONCEPTO	DESCRIPCION	NOTACION
Actor de Negocio	Una entidad de organización que es capaz de comportamiento artístico.	
Rol de Negocio	La responsabilidad de realizar el comportamiento específico, al cual un actor puede ser asignado.	
Colaboración de Negocio	Un conjunto de dos o más papeles de negocio que trabajan juntos para realizar el comportamiento colectivo.	
Interfaz de Negocio	Un punto de acceso donde un servicio de gestión es hecho disponible al entorno.	
Ubicación	Un punto conceptual o ampliado en espacio.	
Objeto de Negocio	Un elemento pasivo que tiene la importancia de una perspectiva de negocio.	
Proceso de Negocio	Un elemento de comportamiento de grupos basado en un ordenamiento de actividades. Es querido para producir un juego definido de productos o servicios de gestión.	
Función de Negocio	Un elemento de comportamiento de grupos basado en un juego escogido de criterios (recursos típicamente requeridos de negocio y/o competencias).	

CONCEPTO	DESCRIPCION	NOTACION
Interacción de Negocio	Un elemento de comportamiento que describe el comportamiento de una colaboración de negocio.	 
Evento de Negocio	Algo qué pasa (internamente o por fuera) e influye en el comportamiento.	 
Servicio de Negocio	Un servicio que realiza una necesidad de negocio de un cliente (interno o externo a la organización).	 
Representación	Una forma perceptible de la información llevada por un objeto de negocio.	
Significado	El conocimiento o la experiencia se presentan en un objeto de negocio o su representación, considerando un contexto particular.	
Valor	El valor de parente, utilidad, o importancia de un servicio de gestión o producto.	
Producto	Una colección coherente de servicios, acompañados por un contraer/poner de acuerdos, que ofrecen en total (a interno o externo) a clientes.	
Contrato	Una especificación formal o informal de acuerdo que especifica los derechos y obligaciones asociadas con un producto.	

Cuadro 4.1: Tabla de Conceptos de la Capa de Negocio

4.3.2 Capa de Aplicación

Se trata de aplicaciones de software que "soportan los componentes de la empresa con servicios de aplicación".

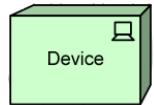
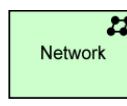
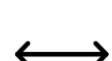
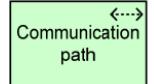
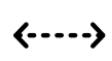
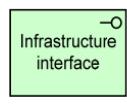
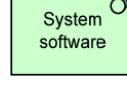
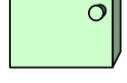
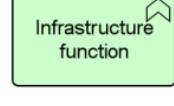
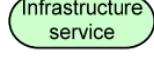
TABLA CONCEPTOS CAPA DE APLICACIÓN		
CONCEPTO	DESCRIPCION	NOTACION
Componente de Aplicación	Una parte modular, desplegable, y reemplazable de un sistema de software que encapsula su comportamiento y datos y expone estos por un juego de interfaces.	
Colaboración de Aplicación	Un conjunto de dos o más componentes de aplicación que trabajan juntos para realizar el comportamiento colectivo.	
Interfaz de Aplicación	Un punto de acceso donde un servicio de aplicación es hecho disponible a un usuario u otro componente de aplicación.	
Objeto de Datos	Un elemento pasivo conveniente para tratamiento automatizado.	
Función de Aplicación	Un elemento de comportamiento que los grupos automatizaron el comportamiento que puede ser realizado por un componente de aplicación.	
Interacción de Aplicación	Un elemento de comportamiento que describe el comportamiento de una colaboración de aplicación.	
Servicio de Aplicación	Un servicio que expone el comportamiento automatizado.	

Cuadro 4.2: Tabla de Conceptos de la Capa de Aplicación

4.3.3 Capa de Tecnologías

Esta capa "Trata con la infraestructura de hardware y comunicación para soportar la Capa de Aplicación, esta capa ofrece servicios de infraestructura necesarios para ejecutar aplicaciones, realizadas por computadora y hardware de comunicación y software de sistema".

TABLA CONCEPTOS CAPA DE TECNOLOGIAS

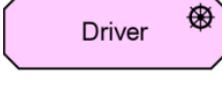
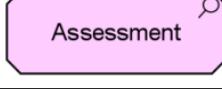
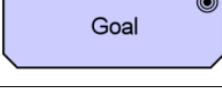
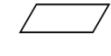
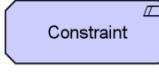
CONCEPTO	DESCRIPCION	NOTACION
Nodo	Un recurso computacional con lo cual los artefactos pueden ser almacenados o desplegados para la ejecución.	 
Dispositivo	Un recurso de hardware con lo cual los artefactos pueden ser almacenados o desplegados para la ejecución.	 
Red	Un medio de comunicación entre dos o más dispositivos.	 
Camino de Comunicación	Un eslabón entre dos o más nodos, por los cuales estos nodos pueden cambiar datos.	 
Interface de Infraestructura	Un punto de acceso donde los servicios de infraestructura ofrecidos por un nodo pueden ser tenidos acceso por otros nodos y componentes de aplicación.	  
Sistema de Software	Un entorno de software para los tipos específicos de componentes y objetos que son desplegados sobre ello en forma de artefactos.	 
Función de Infraestructura	Un elemento de comportamiento de grupos el comportamiento infraestructural puede ser realizado por un nodo.	 
Servicios de Infraestructura	Una unidad por fuera visible de funcionalidad, a condición de que por uno o varios nodos, expuestos por interfaces bien definidos, y significativo al entorno.	

CONCEPTO	DESCRIPCION	NOTACION
Artefacto	Un pedazo físico de los datos que es usado o producido en un proceso de desarrollo de software, o por el despliegue y la operación de un sistema.	 

Cuadro 4.3: Tabla de Conceptos de la Capa de Tecnologías

4.3.4 Capa de Motivación

Los conceptos de motivación se utilizan para modelar las motivaciones, o razones, que subyacen en el diseño o cambio de alguna arquitectura empresarial. Estas motivaciones afectan, orientan y limitan el diseño.

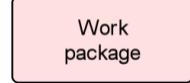
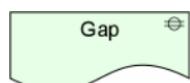
TABLA DE CONCEPTOS CAPA DE MOTIVACIÓN		
CONCEPTO	DESCRIPCION	NOTACION
Tenedor de Apuestas	El papel de un individuo, el equipo, o la organización (o clasifica de eso) que representa sus intereses a, o concierne en relación con, el resultado de la arquitectura.	
Conductor	Algo qué crea, motiva, y abastece de combustible el cambio de una organización.	
Evaluación	El resultado de algún análisis de algún conductor.	
Objetivo	Un estado de final que un tenedor de apuestas tiene la intención de alcanzar.	
Requerimientos	Una declaración de necesidad que debe ser realizada por un sistema.	 
Coacción	Una restricción en el camino en la cual un sistema es realizado.	 
Principio	Una propiedad normativa de todos los sistemas en un contexto dado, o el camino en cual ellos son realizados.	

Cuadro 4.4: Tabla de Conceptos de Capa Motivación

4.3.5 Capa de Implementación y Migración

Es similar a un proceso de negocio, en el sentido de que consiste en un conjunto de tareas relacionadas causalmente, dirigidas a producir un resultado bien definido.

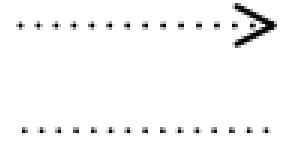
TABLA DE CONCEPTOS CAPA DE IMPLEMENTACIÓN Y MIGRACIÓN

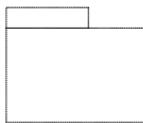
CONCEPTO	DESCRIPCION	NOTACION
Paquete de Trabajo	Una serie de acciones diseñadas para lograr un objetivo único dentro de un tiempo especificado.	
Entregable	Un resultado definido con precisión de un paquete de trabajo.	
Meseta	Un estado relativamente estable de la arquitectura que existe durante un período limitado de tiempo.	
Objetivo	Un resultado de un análisis de hueco entre dos mesetas.	

Cuadro 4.5: Tabla de Conceptos de Capa Migración

4.3.6 Tabla de relaciones

TABLA DE RELACIONES

CONCEPTO	DESCRIPCION	NOTACION
	RELACIONES ESTRUCTURALES	
Asociación	La asociación modela una relación entre los objetos que no es cubierta por el otro, la relación más específica.	
Acceso	La relación de acceso modela el acceso de conceptos conductuales a objetos de datos o el negocio.	
Usado Por	El usado por la relación modela el empleo de servicios por procesos, funciones, o interacciones y el acceso a interfaces por papeles, componentes, o colaboraciones.	

CONCEPTO	DESCRIPCION	NOTACION
Realización	La relación de realización une una entidad lógica con más entidad concreta que lo realiza.	
Asignación	La relación de asignación une las unidades de comportamiento con elementos activos (p.ej., papeles, componentes) que los realiza, o papeles con los actores que los realizan.	
Agregación	La relación de agregación indica que un objeto agrupa un número de otros objetos.	
Composición	La relación de composición indica que un objeto es compuesto de uno o varios otros objetos.	
RELACIONES DINAMICAS		
Flujo	La relación de flujo describe la transferencia de, por ejemplo, la información o el valor entre procesos, función, interacciones, y acontecimientos.	
Provocación	La relación de provocación describe las relaciones temporales o causales entre procesos, funciones, interacciones, y acontecimientos.	
OTRAS RELACIONES		
Agrupación	La relación que de agrupación indica que los objetos, del mismo tipo o tipos diferentes, pertenecen juntos basado en alguna característica común.	
Unión	Una unión es usada para unir las relaciones del mismo tipo.	
Especialización	La relación de especialización indica que un objeto es una especialización de otro objeto.	

Cuadro 4.6: Tabla de Relaciones

4.4 Intercación AMD y ArchiMate

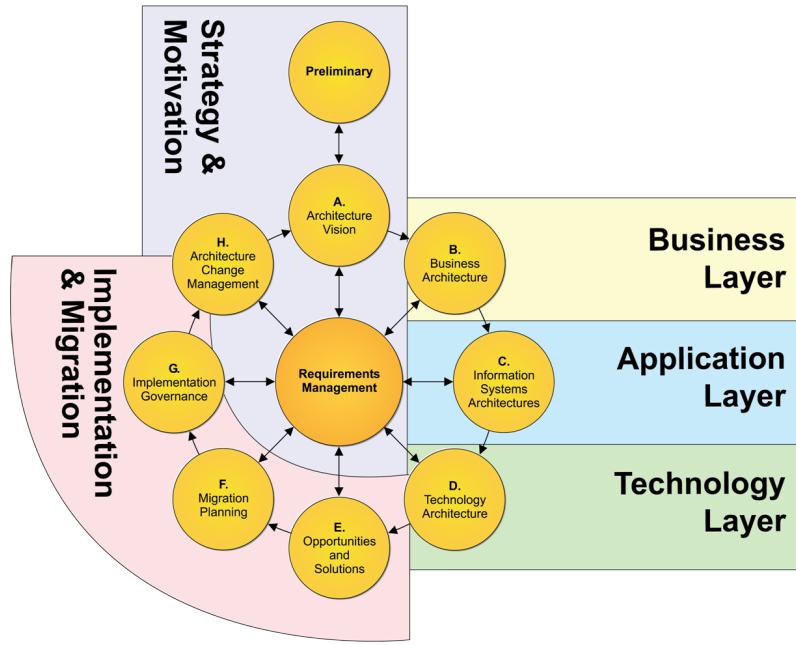


Figura 4.3: Relación entre AMD y ArchiMate [13].

Como se logra ver en la imagen la estructura del lenguaje central de ArchiMate se relaciona estrechamente con las arquitecturas principales de TOGAF ADM. Los elementos de estrategia, motivación, implementación y migración se correlacionan aproximadamente con el resto de ADM (aunque estos elementos también pueden usarse en las Fases B, C y D). Esta correspondencia indica una asignación bastante fácil entre las vistas TOGAF y los puntos de vista de ArchiMate.

Aunque algunos de los puntos de vista que se definen en el estándar TOGAF no se pueden asignar fácilmente a los puntos de vista de ArchiMate, el lenguaje ArchiMate y sus técnicas de análisis soportan los conceptos abordados en estos puntos de vista. Aunque no existe una correspondencia entre uno y otro, todavía hay una buena cantidad de correspondencia entre los puntos de vista de ArchiMate y los puntos de vista de TOGAF.

Los estándares de TOGAF y ArchiMate pueden usarse de forma fácil debido a que:

- Los dos estándares se complementan entre sí con respecto a la definición de un proceso de desarrollo de arquitectura y la definición de un lenguaje de modelado de Arquitectura Empresarial.
- Las dos normas se superponen en su uso de los puntos de vista, y el concepto de un repositorio común subyacente de artefactos y modelos arquitectónicos; Es decir, tienen una base común firme.
- El uso combinado de las dos normas puede apoyar una mejor comunicación con las partes interesadas.

5. Negocio

5.1 Introducción

Esta capa se refiere a los procesos de negocio, servicios, funciones y eventos de cada una de las unidades de negocio. Esta capa ofrece productos y servicios a clientes externos, que se realizan en la organización mediante procesos de negocio realizados por actores y roles empresariales.

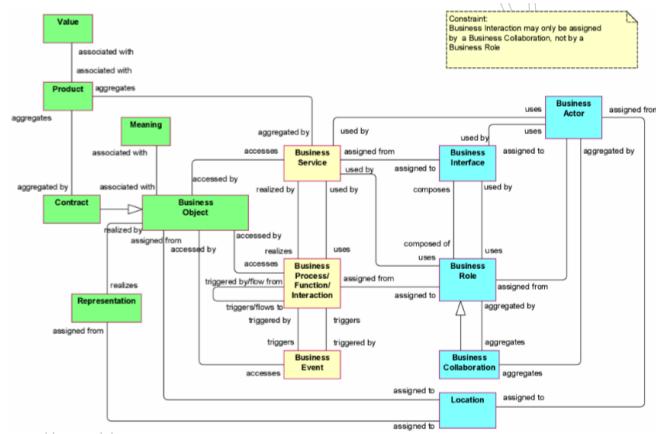


Figura 5.1: Metamodelo de la capa de negocio

La capa de negocio contiene la lógica principal de procesamiento de datos dentro de nuestra aplicación Web. Se comunica con la capa de presentación para obtener las entradas del usuario y presentar la información resultante, así como la capa de acceso a datos o directamente con servicios para realizar sus operaciones.

5.2 Punto de Vista de Organización

5.2.1 Modelo

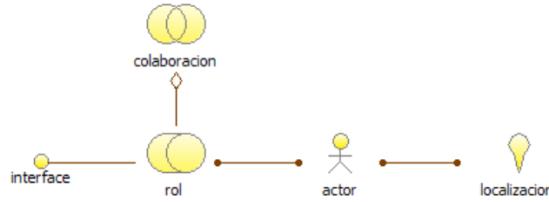


Figura 5.2: Metamodelo de Punto de Vista de Organización [8]

El punto de vista de la organización se centra en la organización (interna) de una empresa, un departamento, una red de empresas o de otra entidad organizativa. Es posible presentar modelos en este punto de vista como diagramas de bloques anidados, pero también de una manera más tradicional, como los organigramas. El punto de vista de la Organización es muy útil en la identificación de competencias, autoridad y responsabilidades en una organización.

5.2.2 Caso de estudio

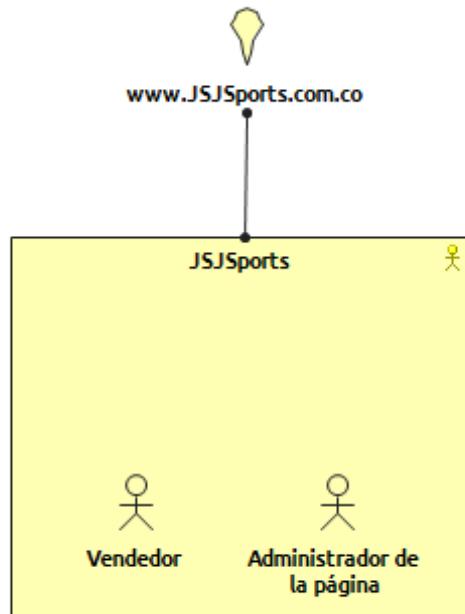


Figura 5.3: Punto de Vista de Organización

En la figura 5.3 se puede apreciar que la empresa JSJ sports la cual es un actor tiene un sitio web, localizado en una dirección electrónica, a su vez se observa que se encuentra conformada por un vendedor y un administrador.

5.3 Punto de Vista de cooperación de Actor

5.3.1 Modelo

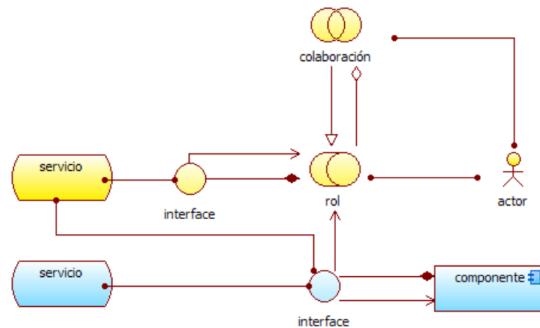


Figura 5.4: Metamodelo de Punto de Vista de Cooperación de Actor [3]

El punto de vista de la Cooperación Actor se centra en las relaciones de los actores entre sí y su entorno. Un ejemplo común de esto es el "diagrama de contexto", que pone a una organización en su entorno, que consiste en partes externas, como clientes, proveedores y otros socios comerciales. Es muy útil para determinar las dependencias externas y las colaboraciones y muestra la cadena de valor o la red en la que actúa el actor.

5.3.2 Caso de estudio

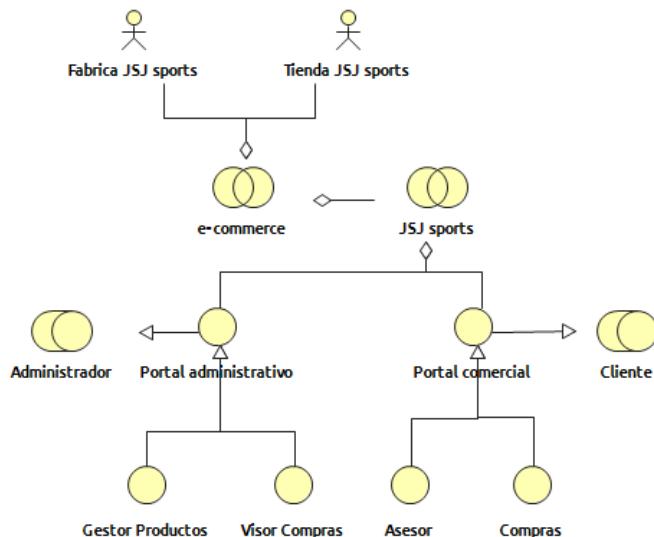


Figura 5.5: Punto de vista Cooperacion Actor

En la figura 5.5 se puede observar que la empresa JSJ sports se encuentra conformada por un e-commerce, que tiene como componentes la fabrica y la tienda de la empresa respectivamente, a su vez tendrá dos portales, un portal comercial que sera la interfaz de JSJSports para los clientes, el cual tiene una interfaz de asesor y una de compras y un portal administrativo que se compone de dos interfaces, un gestor de productos y un visor de compras y le permitirá al administrador realizar toda la gestión y configuración del sitio.

5.4 Punto de Vista de Función de Negocio

5.4.1 Modelo

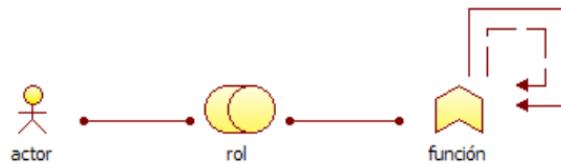


Figura 5.6: Metamodelo de Punto de Vista de Función de Negocio [7]

El punto de vista Función de negocio muestra las principales funciones de negocio de una organización y sus relaciones en términos de los flujos de información, valor o bienes entre ellos. Las funciones empresariales se utilizan para representar los aspectos más estables de una empresa en términos de las actividades primarias que realiza, independientemente de los cambios organizacionales o desarrollos tecnológicos. Por lo tanto, la arquitectura de la función comercial de las empresas que operan en el mismo mercado a menudo muestran similitudes cercanas.

5.4.2 Caso de estudio

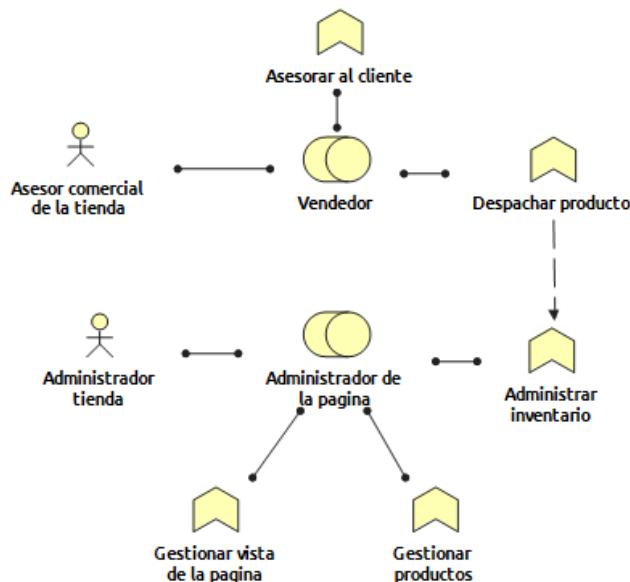


Figura 5.7: Punto de vista de función negocio

En la figura 5.7 se puede observar como la empresa JSJ sports tiene como principales roles un vendedor y un administrador de la página, el primero que se puede especificar como un asesor comercial de la tienda debido a que tiene como funciones asesorar al cliente durante los procesos de compra y despachar el producto una vez este ha sido ordenado, el segundo que se puede especificar como el administrador de la tienda, tiene como funciones gestionar la vista de la página, gestionar productos y administrar el inventario que es disparada por la función despachar productos del vendedor.

5.5 Punto de Vista de Proceso de negocio

5.5.1 Modelo

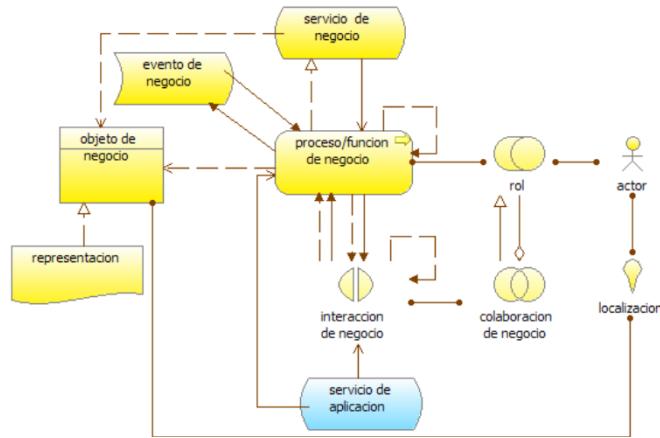


Figura 5.8: Metamodelo de Punto de Vista de Proceso de Negocio [9]

El punto de vista de Proceso de Negocio se utiliza para mostrar la estructura y composición de alto nivel de uno o más procesos empresariales.

5.5.2 Caso de estudio

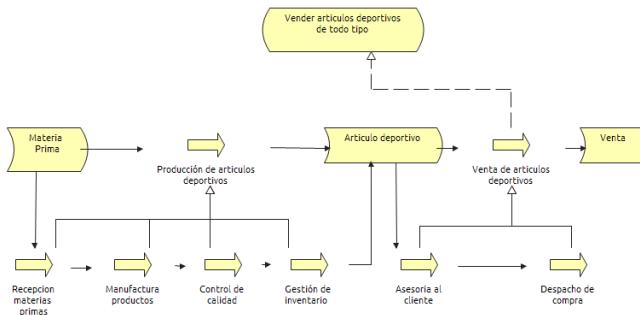


Figura 5.9: Modelo de Punto de Vista de Proceso de Negocio [5]

Como se puede ver en la figura 5.9 el servicio principal de la empresa es el vender artículos deportivos de todo tipo, este servicio depende completamente del proceso venta de artículos deportivos, el cual se divide en los sub-procesos de asesoría al cliente y despacho de la compra. A pesar de que el proceso principal es la venta, se sabe que este depende de otro gran proceso el cual es la producción de los artículos deportivos que se venden, esto es importante puesto que hace parte e los objetivos y misión de la empresa, este proceso a su vez se divide en 4 sub-procesos los cuales son: Recepción de materias primas, manufactura de productos, control de calidad y gestión de inventario.

5.6 Punto de Vista de Cooperación Proceso de negocio

5.6.1 Modelo

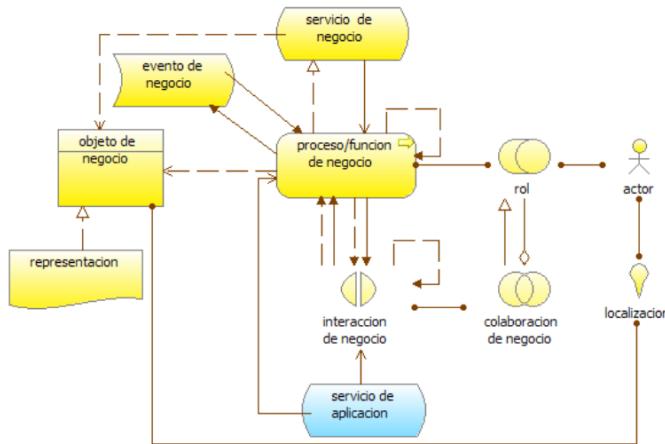


Figura 5.10: Metamodelo de Punto de Vista Cooperación de Proceso de Negocio [5]

El punto de vista de Cooperación de Proceso de Negocio se utiliza para mostrar las relaciones de uno o más procesos de negocio entre sí y / o con su entorno. Puede utilizarse tanto para crear un diseño de alto nivel de procesos empresariales dentro de su contexto como para proporcionar un gestor operativo responsable de uno o más de dichos procesos con información sobre sus dependencias.

5.6.2 Caso de estudio

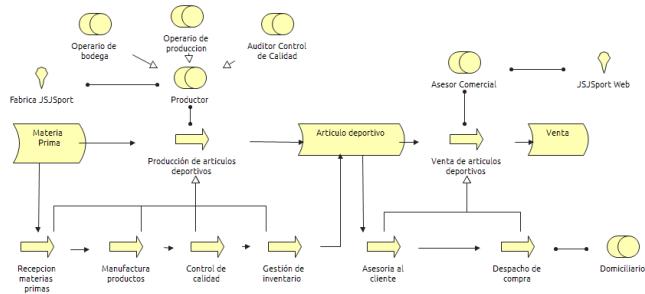


Figura 5.11: Modelo de Punto de Vista Cooperación Proceso de Negocio [5]

Al igual que en anterior punto de vista en este modelo se puede ver que la empresa tiene 2 grandes procesos, el primer proceso es el de la producción de artículos deportivos, este proceso es llevado a cabo por el productor en la Fabrica de JSJSport, el rol de productor esta conformado por varios roles, el primero es el operario de bodega quien es el encargado de recibir materias primas y organizar inventarios y productos, el segundo es el operario de producción donde se encuentran todos los encargados de la elaboración del producto y por último el encargado de realizar el proceso de control de calidad.

El segundo gran proceso es llevado a cabo por el asesor comercial quien asesora al cliente y realiza la venta en la tienda de JSJSport, ademas de esto en el sub-proceso de despacho de compra se incluye el rol de domiciliario quien es el encargado de entregar este producto al cliente.

5.7 Punto de Vista de Producto

5.7.1 Modelo

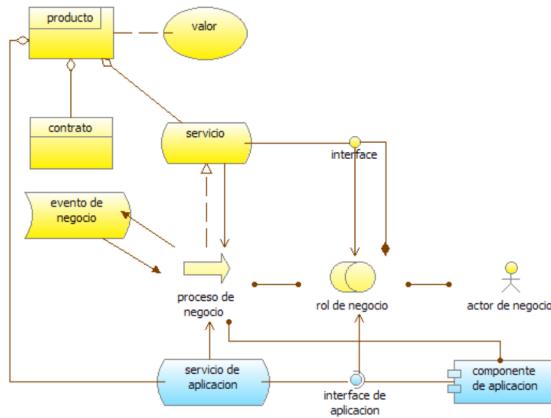


Figura 5.12: Metamodelo de Punto de Vista de Producto[10]

El punto de vista del Producto representa el valor que estos productos ofrecen a los clientes u otras partes externas involucradas y muestra la composición de uno o más productos en términos de los servicios constitutivos (comerciales o de aplicación) y los contratos asociados u otros acuerdos.

5.7.2 Caso de estudio

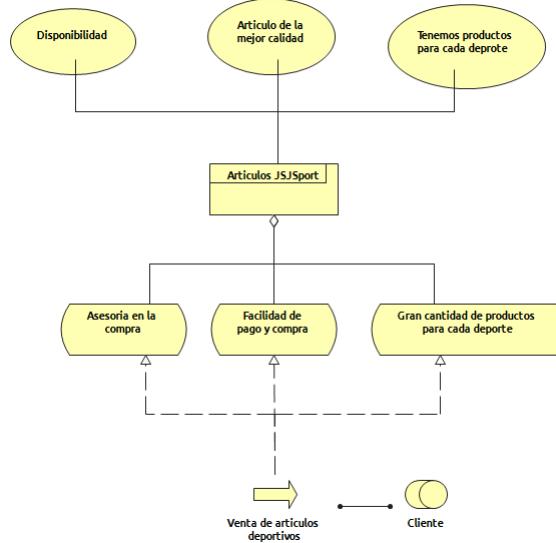


Figura 5.13: Modelo de Punto de Vista de Producto [5]

En la imagen 5.13 se puede observar que el principal servicio de JSJSport es la venta de artículos deportivos al cliente, el producto estrella de la empresa JSJ son los artículos deportivos JSJ, el producto cuenta con 3 diferentes servicios claves para la identidad de la empresa, los cuales son, asesoría al cliente, facilidad de pago y compra, y la gran cantidad de artículos ofrecidos para cada deporte con la cual se busca que en la tienda se tengan los artículos necesarios para cada disciplina deportiva. A su vez el producto tiene 3 valores, la disponibilidad del producto y de los asesores, la calidad de este producto y la garantía de que el cliente va a encontrar lo que busca.

6. Aplicacion

6.1 Introducción

La siguiente figura ofrece una visión general de los conceptos de capa de aplicación y sus relaciones, esta capa soporta los componentes de la empresa con servicios de aplicación. Muchos de los conceptos se han inspirado en el estándar UML 2.0 [7], [10], ya que este es el lenguaje dominante y el estándar que se usa hoy en día describir las aplicaciones de software.

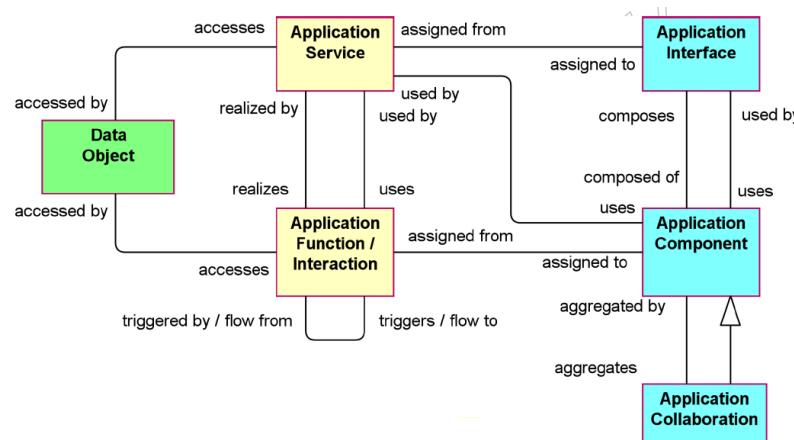


Figura 6.1: Capa de Aplicación

Esta figura no muestra todas las relaciones permitidas: cada concepto en el lenguaje puede tener relaciones de composición, agregación y especialización con conceptos del mismo tipo; Además, existen relaciones indirectas que pueden derivarse.

6.2 Punto de Vista de Comportamiento de Aplicación

6.2.1 Modelo

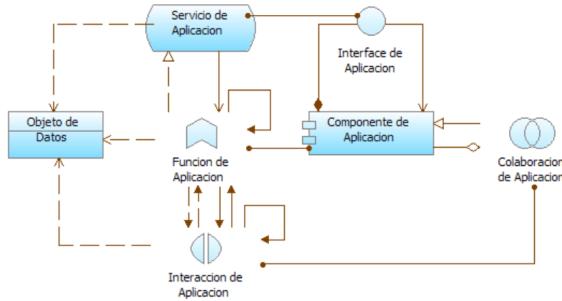


Figura 6.2: Metamodelo de Punto de Vista de Comportamiento de Aplicación [2]

El punto de vista del comportamiento de la aplicación describe el comportamiento interno de una aplicación, por ejemplo, cuando realiza uno o más servicios de aplicación. Este punto de vista es útil para diseñar el comportamiento principal de las aplicaciones, o para identificar la superposición funcional entre diferentes aplicaciones.

6.2.2 Caso de estudio

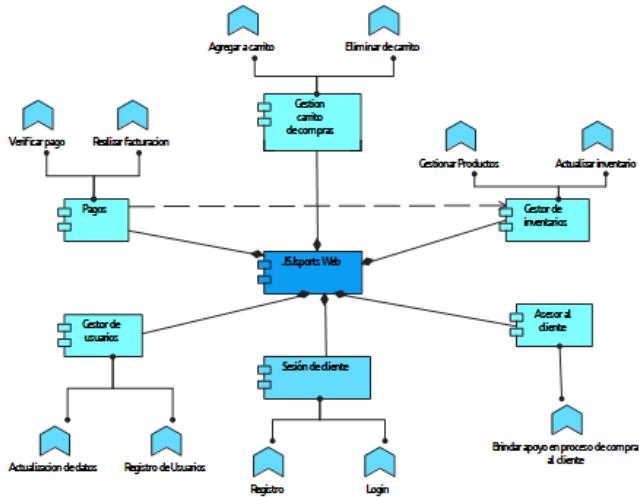


Figura 6.3: Punto de Vista Comportamiento de Aplicación

En la figura 6.3 se observa el componente principal de la empresa JSJSports, que esta conformada por varios componentes los que a su vez tienen diferentes funciones.

El componente pagos tiene dos funciones principales las cuales son verificar el pago y realizar la facturación, este componente a su vez llama al componente de gestor de inventario que se encarga de gestionar los productos y actualizar el inventario de los mismos, el componente gestor de carrito de compras se encarga de agregar y eliminar los diferentes productos seleccionados por el cliente del carrito. El componente de gestor de usuario se encarga de actualizar los datos y el registro de usuarios, el componente de sesión de cliente, se encarga del registro y el login que realiza el cliente para ingresar al sistema y finalmente el componente asesor de cliente que se encarga de establecer comunicación entre el asesor y el usuario para brindar apoyo en el proceso de compra.

6.3 Punto de Vista de cooperación de Aplicación

6.3.1 Modelo

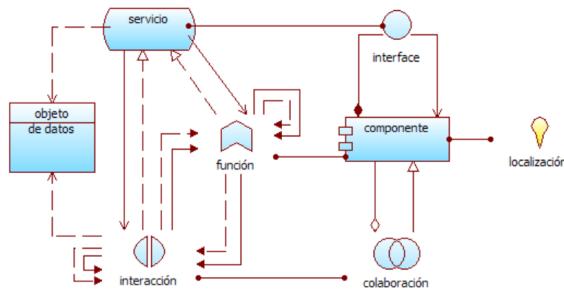


Figura 6.4: Metamodelo de Punto de Vista de Cooperación de Aplicación [4]

El punto de vista Cooperación de Aplicación describe las relaciones entre los componentes de las aplicaciones en términos de los flujos de información entre ellos o en términos de los servicios que se ofrecen y utilizan. Este punto de vista se suele utilizar para crear una visión general del entorno de aplicación de una organización. Este punto de vista también se utiliza para expresar la cooperación (interna) o la orquestación de servicios que juntos apoyan la ejecución de un proceso de negocio.

6.3.2 Caso de estudio

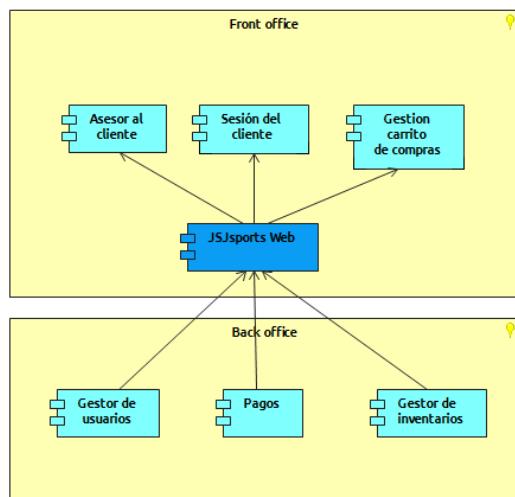


Figura 6.5: Punto de Vista Cooperación de Aplicación

Como se puede ver en la figura se divide la aplicación en dos ubicaciones una que es visible y tiene acceso para el usuario, el front office y la otra que es ya la correspondiente al back end de la aplicación que es el back office el cual no es de acceso directo para el usuario, allí se tiene la lógica de la aplicación y la persistencia de la misma.

En el back office se puede observar el gestor de pagos, el de inventarios y el de usuarios, en la parte del front office se tiene todo el componente de JSJSport web que actúa como puente entre el front office y el back office, además de este, también encontramos en el front office el componente de asesoria y el de sesión del cliente, y el componente que gestiona el carrito de compras del usuario que le permite realizar las compras.

6.4 Punto de Vista de Estructura de aplicación

6.4.1 Modelo

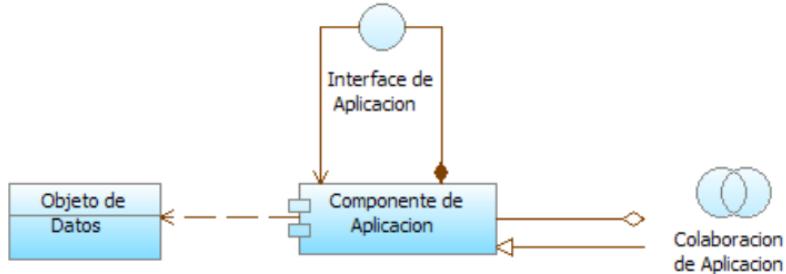


Figura 6.6: Metamodelo de Punto de Vista de Estructura de Aplicación [6]

El punto de vista de Estructura de la aplicación muestra la estructura de una o más aplicaciones o componentes. Este punto de vista es útil para diseñar o comprender la estructura principal de aplicaciones o componentes y los datos asociados, por ejemplo, para descomponer la estructura del sistema en construcción o para identificar componentes de aplicación heredados que son adecuados para la migración/integración.

6.4.2 Caso de estudio

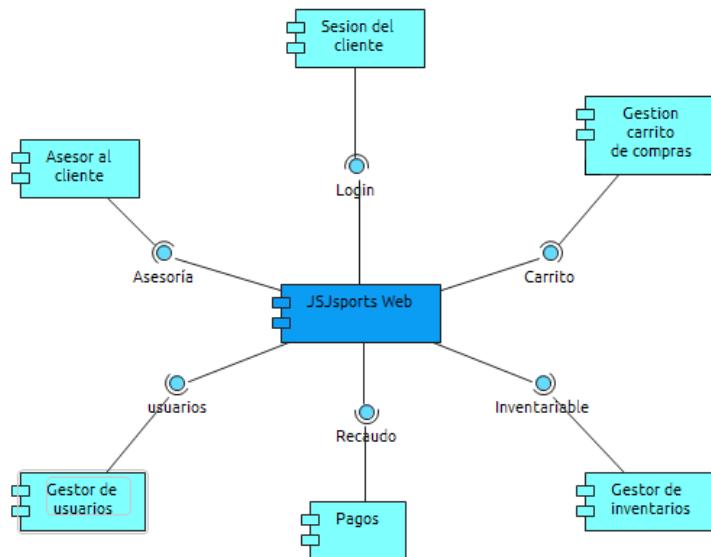


Figura 6.7: Punto de Vista Estructura de Aplicación

En este punto de vista podemos ver como los componentes se comunican con el componente central JSJSport Web por medio de interfaces, la interfaz asesoría permite brindar asesoría al usuario, la interfaz login le permite al usuario administrar su sesión, la interfaz carrito ayuda a manejar el carrito de compras. La interfaz recaudo permite gestionar los pagos, la interfaz de usuarios permite gestionar los usuarios, roles y permisos, y la interfaz inventariable permite hacer la gestión de los inventarios.

6.5 Punto de Vista de Uso de Aplicación

6.5.1 Modelo

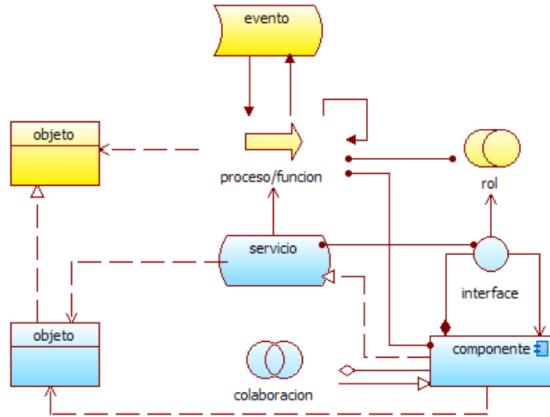


Figura 6.8: Metamodelo de Punto de Vista de Uso de Aplicación [11]

El punto de vista Uso de aplicación describe cómo se utilizan las aplicaciones para soportar uno o más procesos empresariales y cómo se utilizan en otras aplicaciones. Se puede utilizar en el diseño de una aplicación mediante la identificación de los servicios necesarios por los procesos de negocio y otras aplicaciones, o en el diseño de procesos de negocio mediante la descripción de los servicios que están disponibles.

6.5.2 Caso de estudio

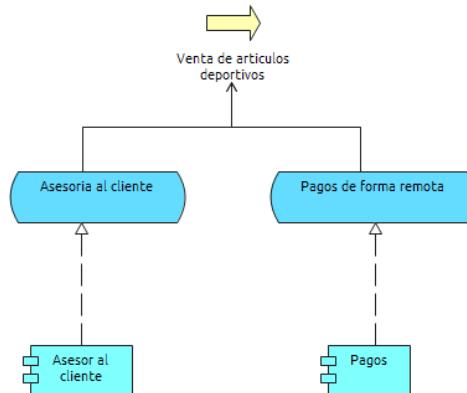


Figura 6.9: Punto de Vista Uso de Aplicación

El proceso empresarial que soportara el sistema será el de la venta de artículos deportivos, como se puede ver en la imagen de este proceso dependen 2 servicios, el primero es el de asesoría de cliente que se suple con ayuda del componente de asesor del cliente y que permite aconsejar, apoyar y acompañar al usuario durante el proceso de compra, el segundo servicio es el de hacer los pagos y las compras de forma remota sin necesidad de ir hasta la tienda física, este servicio se suple con el componente de pagos que ayuda a gestionar los pagos que se hacen en el sistema.

7. Tecnología

7.1 Introducción

El principal concepto estructural para la capa tecnológica es el nodo. Este concepto se utiliza para modelar entidades estructurales en esta capa. Es idéntico al concepto de nodo de UML 2.0. Se modela estrictamente el aspecto estructural de un sistema: su comportamiento es modelado por una relación explícita con los conceptos de comportamiento.

Una interfaz de infraestructura es la ubicación (lógica) donde los servicios de infraestructura ofrecidos por un nodo pueden ser accedidos por otros nodos o por componentes de la aplicación de la capa de aplicación.

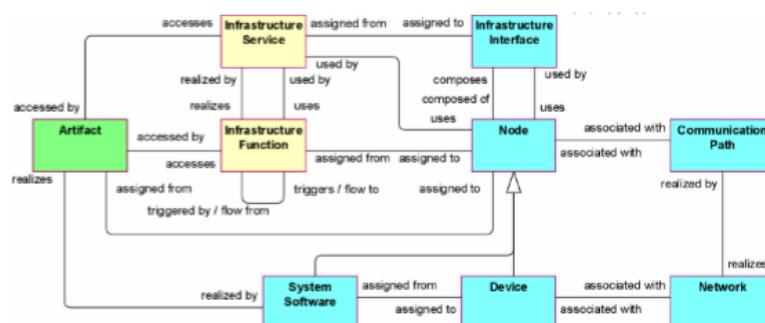


Figura 7.1: Metamodelo de la Capa de Tecnología

Las interrelaciones de los componentes de la capa tecnológica están formadas principalmente por la infraestructura de comunicación. El camino de comunicación modela la relación entre dos o más nodos, a través de los cuales estos nodos pueden intercambiar información. La realización física de un camino de comunicación es modelada con una red; es decir, un medio físico de comunicación entre dos o más dispositivos (u otras redes).

7.2 Punto de Vista de Infraestructura

7.2.1 Modelo

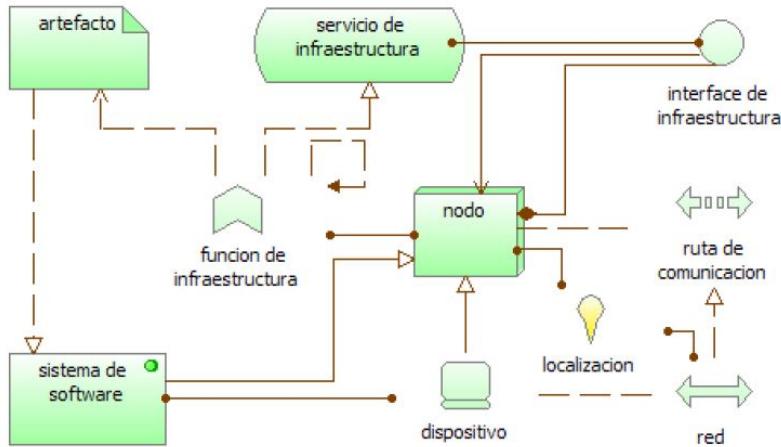


Figura 7.2: Metamodelo Punto de Vista Infraestructura

El punto de vista Infraestructura contiene los elementos de infraestructura de software y hardware que soportan la capa de aplicación, como dispositivos físicos, redes o software del sistema (por ejemplo, sistemas operativos, bases de datos y middleware).

7.2.2 Caso de estudio

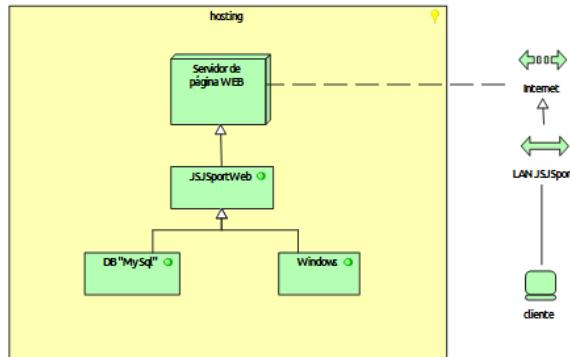


Figura 7.3: Modelo Punto de Vista Infraestructura

Como se puede ver en la Figura 7.3 este punto de vista refleja la infraestructura necesaria para nuestra aplicación, la página WEB se ubicara en un host y dependerá de un servidor WEB, además para la persistencia se hará uso de una base de datos de MySQL y además se hará uso de un sistema operativo Windows, el servidor WEB se comunica con internet, donde el cliente se comunica con su dispositivo a una LAN que se conecta a internet y por lo tanto a la página web.

7.3 Punto de Vista de Uso de Infraestructura

7.3.1 Modelo

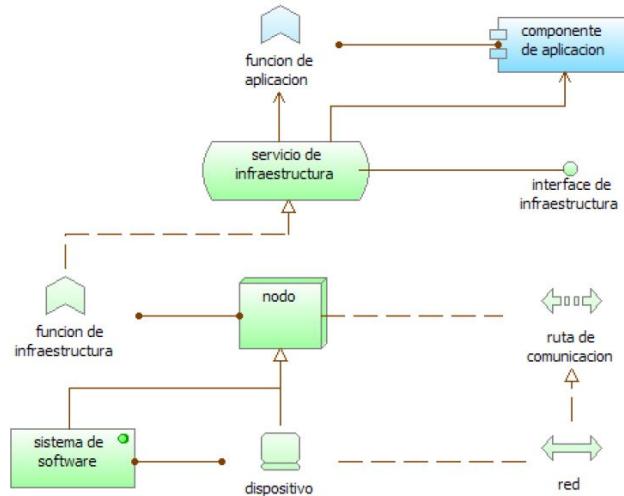


Figura 7.4: Metamodelo Punto de Vista Uso de Infraestructura

El punto de vista de uso de la infraestructura muestra cómo las aplicaciones son compatibles con la infraestructura de software y hardware: los servicios de infraestructura son entregados por los dispositivos; Software del sistema y las redes se proporcionan a las aplicaciones.

7.3.2 Caso de estudio

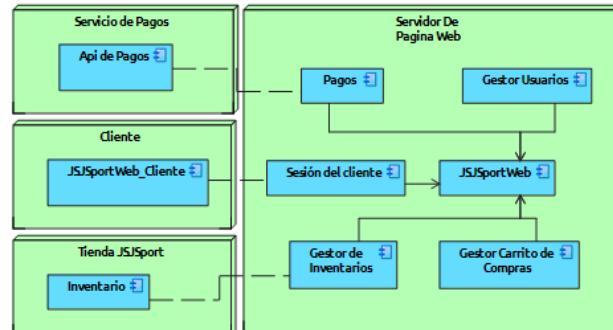


Figura 7.5: Modelo Punto de Vista Uso de Infraestructura

Como se puede ver en la figura 7.5 el uso de la infraestructura esta dividido en nodos, el nodo del servidor WEB aloja todos los componentes previamente definidos de la página web que corresponden a la página incluyendo la principal.

Los otros 3 componentes corresponden a los nodos externos a la aplicación, el nodo del servicio de pagos provee un api que proporciona un tercer y que se encarga de los pagos este se comunica con los pagos de la aplicación y lo acopla, el nodo del cliente es lo que el cliente ve desde su ordenador y se comunica con el componente que maneja la sesión del cliente, por último se tiene el componente de inventario que se aloja en la tienda donde se maneja el inventario físico.

7.4 Punto de Vista de Implementación y Despliegue

7.4.1 Modelo

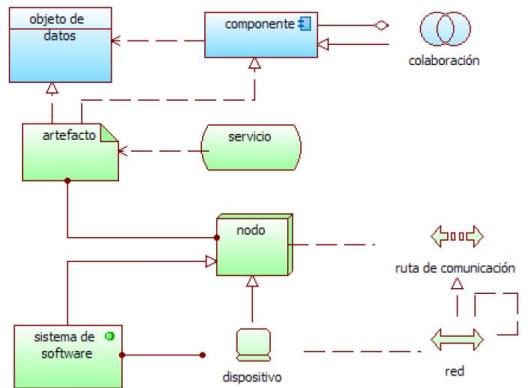


Figura 7.6: Metamodelo Punto de Vista Organización e Implementación

El punto de vista Organización e Implementación muestra cómo se realizan una o más aplicaciones en la infraestructura. Esto comprende la asignación de aplicaciones y componentes (lógicos) o artefactos (físicos), como Enterprise Java Beans, y la asignación de la información utilizada por estas aplicaciones y componentes en la infraestructura de almacenamiento subyacente. Por ejemplo, tablas de base de datos u otros archivos. Las vistas de implementación juegan un papel importante en el análisis de rendimiento y escalabilidad, ya que relacionan la infraestructura física con el mundo lógico de las aplicaciones. En análisis de seguridad y riesgo, las vistas de implementación se utilizan para identificar, por ejemplo, dependencias y riesgos críticos.

7.4.2 Caso de estudio

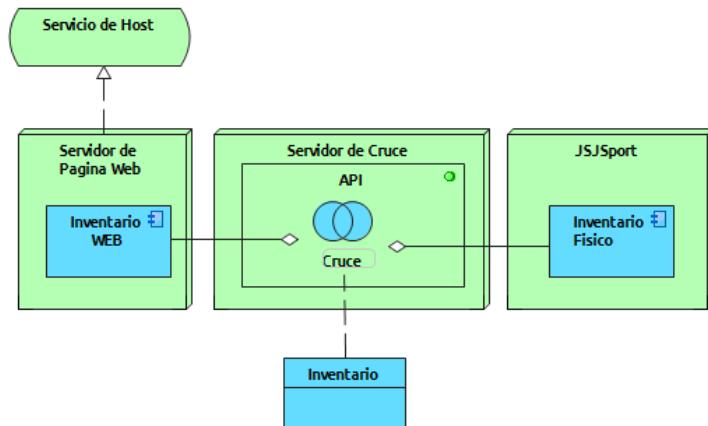


Figura 7.7: Modelo Punto de Vista Organización Implementación

En este punto de vista se puede observar la colaboración que existe entre el componente de inventarios de la página WEB que se aloja en el host y el componente de inventario físico que existe en la tienda de JSJSport, esta colaboración tiene lugar en un API que se encuentra en el servidor de cruce y que accede y hace uso del elemento inventario.

7.5 Punto de Vista de Estructura de la Información

7.5.1 Modelo

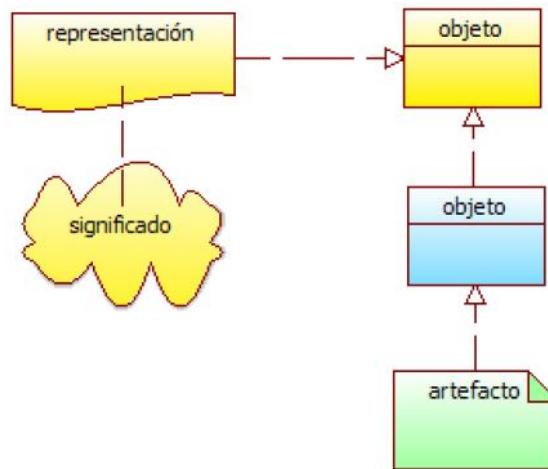


Figura 7.8: Metamodelo Punto de Vista Estructura de Información

El punto de vista de la estructura de información es comparable a los modelos de información tradicionales creados en el desarrollo de casi cualquier sistema de información. Muestra la estructura de la información utilizada en la empresa o en un proceso o aplicación de negocio específico, en términos de tipos de datos o estructuras de clases (orientadas a objetos).

7.5.2 Caso de estudio

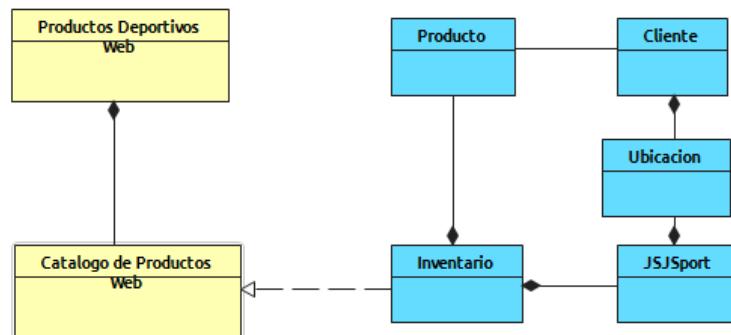


Figura 7.9: Modelo Punto de Vista Estructura Información

En la figura 7.9 se puede observar como el conjunto de productos que tiene JSJSports se ve reflejado en el catálogo de productos de la empresa, del lado de la aplicación se puede ver que el inventario se ve reflejado en el catálogo de productos, este inventario tiene muchos productos y a la vez se encuentra en JSJSport que tiene una ubicación.

Al igual que JSJSport, el cliente también tiene una ubicación y no solo esto sino que también accede a los productos que tiene el inventario, todo esto desde la aplicación.

7.6 Punto de Vista de Realización del Servicio

7.6.1 Modelo

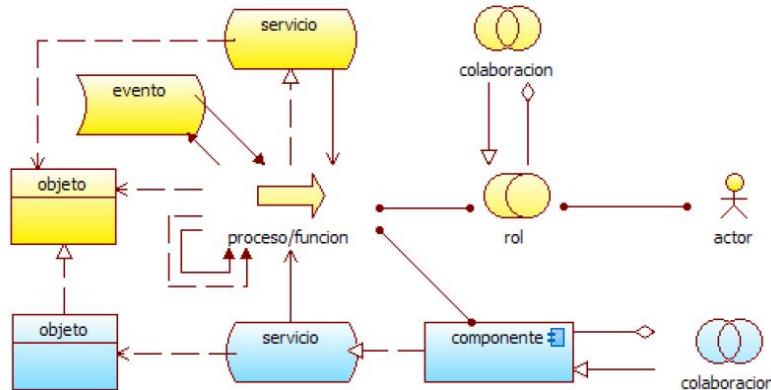


Figura 7.10: Metamodelo Punto de Vista Realización Del Servicio

El punto de vista Realización del servicio se utiliza para mostrar cómo uno o más servicios empresariales son realizados por los procesos subyacentes (ya veces por los componentes de la aplicación). Por lo tanto, forma el puente entre el punto de vista de productos empresariales y la vista de procesos empresariales.

7.6.2 Caso de estudio

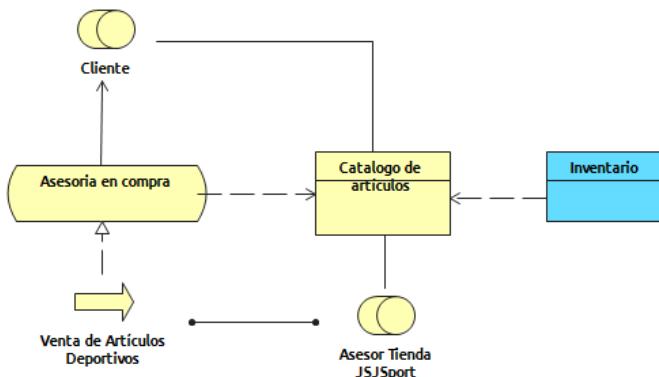


Figura 7.11: Modelo Punto de Vista Realización Servicio

En este punto de vista se puede observar como los servicios previamente definidos en las otras capas son ejecutados por los componentes que se definieron en la capa de aplicación, en este caso el servicio de asesoría en la compra de productos deportivos esta siendo brindado a un cliente, la función de venta de artículos deportivos busca satisfacer este servicio, el encargado de realizar esta función es un asesor de la tienda, todo esto se dirige al catálogo de productos puesto que este es el tema central con el que se busca satisfacer al cliente y es la razón por la cual se comunican el cliente y el asesor, este catálogo a su vez hace uso del inventario que se encuentra en la aplicación.

7.7 Punto de Vista de Capas

7.7.1 Modelo

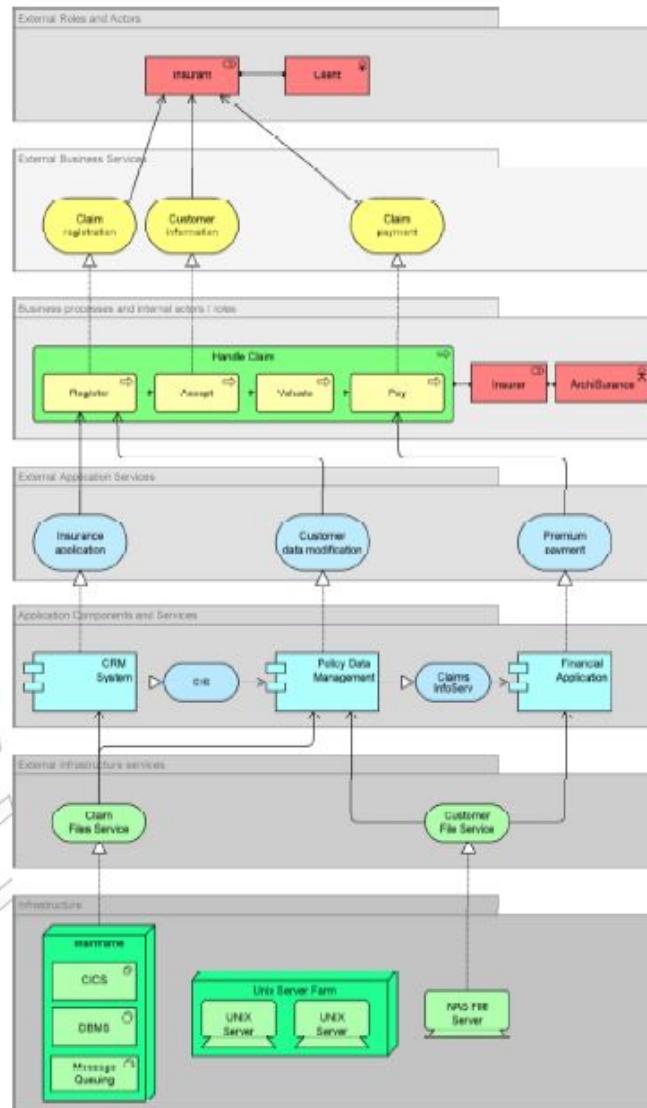


Figura 7.12: Metamodelo Punto de Vista Capas

El punto de vista Capas muestra varias capas y aspectos de una arquitectura empresarial en un diagrama. Hay dos categorías de capas, a saber, capas dedicadas y capas de servicio. Las capas son el resultado del uso de la relación de "agrupación" para una partición natural de todo el conjunto de objetos y relaciones que pertenecen a un modelo. La infraestructura, la aplicación, el proceso y las capas de actores / roles pertenecen a la primera categoría. El principio estructural detrás de un punto de vista completamente estratificado es que cada capa dedicada expone, mediante la relación de realización", una capa de servicios, que son "usados" por la siguiente capa dedicada.

7.7.2 Caso de estudio

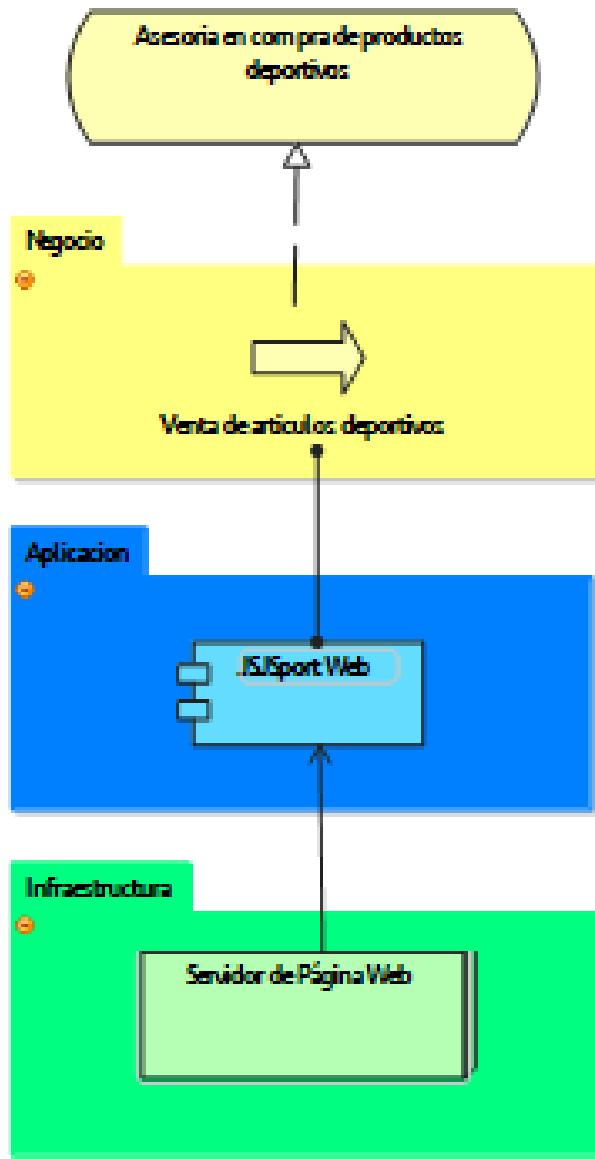


Figura 7.13: Modelo Punto de Vista de Capas

Sin lugar a dudas este punto de vista que se ve reflejado en la figura 7.13 es uno de los mas importantes sino es que es el más importante puesto que refleja el trabajo hecho en las anteriores capas y como se conectan, lo principal es el servicio de la asesoría en compras de productos deportivos que se busca brindar dentro de la capa de negocio vemos la función que se realizará, dentro de la aplicación el componente que va a realizar esta función y dentro de la infraestructura podemos ver lo que nos permitirá montar concretamente la solución para lograr así satisfacer el servicio principal.



8. Motivación

8.1 Introducción

Los conceptos de motivación se utilizan para modelar las motivaciones, o razones, que subyacen en el diseño o cambio de alguna arquitectura empresarial. Estas motivaciones influyen, orientan y limitan el diseño.

Es esencial comprender los factores, a menudo referidos como conductores, que influyen en los elementos motivacionales. Pueden originarse desde dentro o fuera de la empresa. Los conductores internos, también llamados preocupaciones, están asociados con las partes interesadas, que pueden ser algún ser humano individual o algún grupo de seres humanos, como un equipo de proyecto, empresa o sociedad. rentabilidad. Es común que las empresas realicen una evaluación de estos conductores; Por ejemplo, utilizando un análisis DAFO, con el fin de responder de la mejor manera.

Las motivaciones reales están representadas por objetivos, principios, requisitos y limitaciones. Los objetivos representan algún resultado deseado - o final - que un interesado quiere lograr; Por ejemplo, aumentando la satisfacción del cliente en un 10. Los principios y los requisitos representan las propiedades deseadas de las soluciones - o medios - para realizar las metas. Los principios son pautas normativas que guían el diseño de todas las soluciones posibles en un contexto dado. Por ejemplo, el principio "Los datos deben almacenarse sólo una vez" representa un medio para lograr el objetivo de consistencia de datos; se aplica a todos los posibles diseños de la arquitectura de la organización. Los requisitos representan declaraciones formales de necesidad, expresadas por los interesados, que deben ser satisfechas por la arquitectura o las soluciones. Por ejemplo, el requisito "Usar un único sistema de CRM" se ajusta al principio antes mencionado aplicándolo a la arquitectura de la organización actual en el contexto de la gestión de los datos de los clientes.

8.2 Punto de Vista de Stakeholder

8.2.1 Modelo

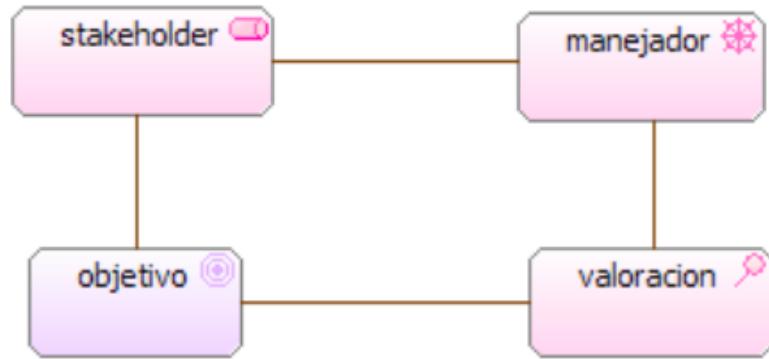


Figura 8.1: Metamodelo Punto de Vista de Partes Interesadas

El punto de vista de las partes interesadas permite al analista modelar las partes interesadas, los impulsores internos y externos del cambio y las evaluaciones (en términos de fortalezas, debilidades, oportunidades y amenazas) de estos controladores. También se pueden describir los vínculos con los objetivos iniciales (de nivel alto) que abordan estas preocupaciones y evaluaciones. Estos objetivos forman la base para el proceso de ingeniería de requisitos, incluyendo refinamiento de objetivos, contribución y análisis de conflictos, y la derivación de requisitos que realicen los objetivos

8.2.2 Caso de estudio

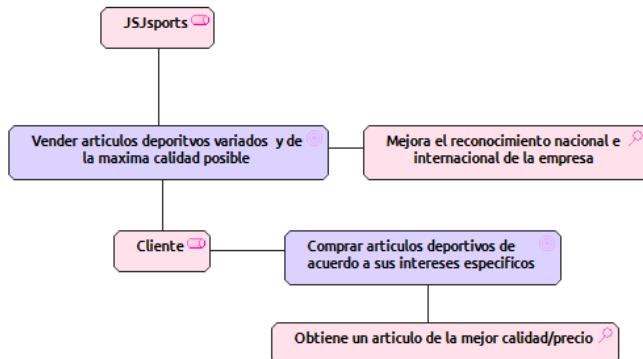


Figura 8.2: Modelo Punto de Vista de Partes Interesadas

En este punto de vista de la figura 8.2 se puede observar como las dos partes interesadas son la tienda JSJSports y se relacionan por medio del objetivo de vender artículos deportivos, puesto que a JSJSport le interesa vender productos y a el cliente comprarlos, este objetivo se mide por medio del reconocimiento que adquiere JSJSports nacional e internacional mente. Ademas del objetivo de la venta el cliente tiene el objetivo de conseguir productos de calidad conforme a sus deseos específicos, este objetivo se mide si se logra que el cliente consiga el producto que desea en la tienda y de la mejor calidad.

8.3 Punto de Vista de Realización de Objetivos

8.3.1 Modelo

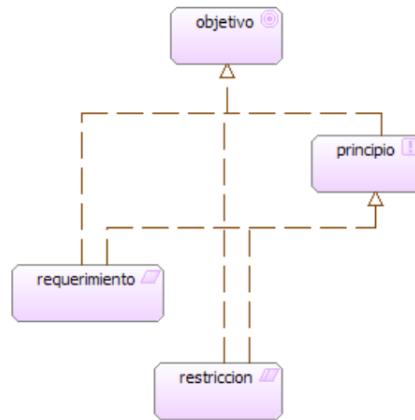


Figura 8.3: Metamodelo Punto de Vista de Realización de Objetivos

El punto de vista de la realización de metas permite a un diseñador modelar el refinamiento de metas (de alto nivel) en metas más concretas y el refinamiento de objetivos concretos en requisitos o restricciones que describen las propiedades que se necesitan para realizar las metas. El refinamiento de objetivos en subobjetivos se modela utilizando la relación de agregación. El refinamiento de metas en requisitos se modela utilizando la relación de realización. Además, los principios pueden ser modelados que guían el refinamiento de objetivos en requisitos

8.3.2 Caso de Estudio

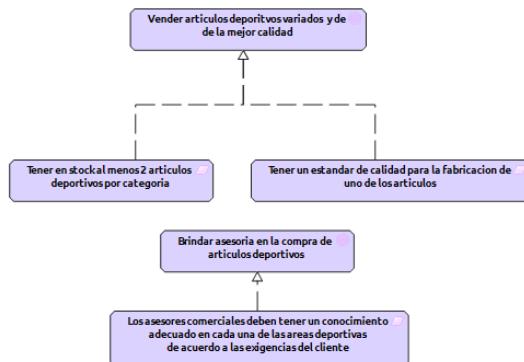


Figura 8.4: Modelo Punto de Vista de Realización de Objetivos

En la figura 8.4 se pueden ver los dos objetivos principales de la empresa JSJSports, el primero de estos es el de vender artículos deportivos de la mejor calidad y de todo tipo, este tiene dos requerimientos, estos son el de tener en stock por lo menos 2 artículos diferentes para cada disciplina deportiva y el de tener un estandar que defina la calidad de estos productos.

El segundo objetivo es el de brindar asesoría al cliente en el proceso de compra al cliente, el cumplimiento de este objetivo requiere que los asesores tengan un conocimiento sobre cada una de las disciplinas y los artículos que se pueden recomendar al cliente según sus requerimientos.

8.4 Punto de Vista de contribución de Objetivos

8.4.1 Modelo

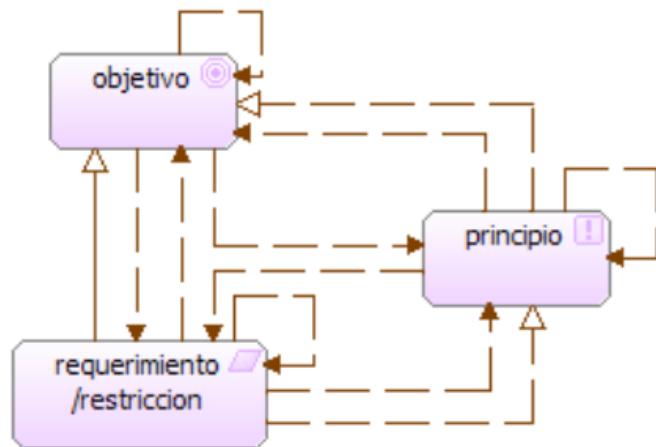


Figura 8.5: Metamodelo Punto de Vista de Contribucion

El punto de vista de la contribución permite a un diseñador o analista modelar las relaciones de influencia entre objetivos y requisitos. Las vistas resultantes pueden usarse para analizar el impacto que las metas tienen entre sí o para detectar conflictos entre los objetivos de las partes interesadas. Típicamente, este punto de vista puede ser utilizado después de que los objetivos se hayan refinado hasta cierto punto en subobjetivos y, posiblemente, en requisitos.

8.4.2 Caso de Estudio

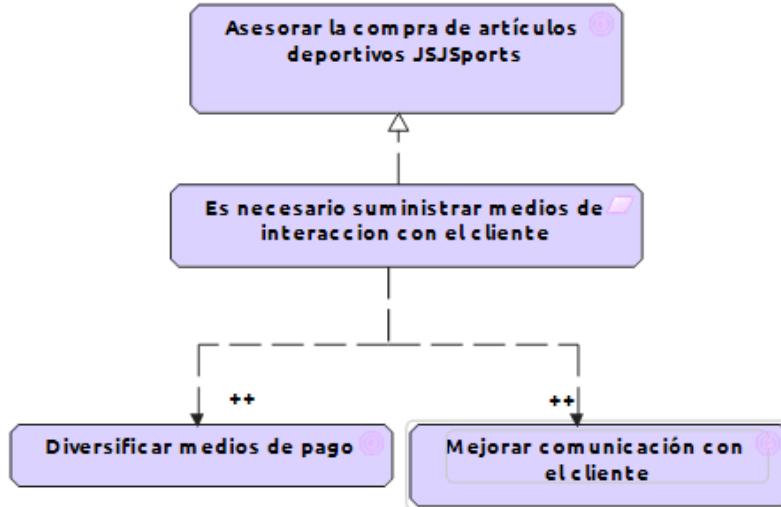


Figura 8.6: Modelo Punto de Vista de Contribucion

En el modelo de contribución en la figura 8.6 se puede observar como para garantizar el cumplimiento del objetivo de asesorar la compra de artículos deportivos es necesario garantizar medios suficientes de interacción con el cliente, esto se logra Diversificando los medios de pago y mejorando la comunicación con los clientes.

8.5 Punto de Vista de Principios

8.5.1 Modelo

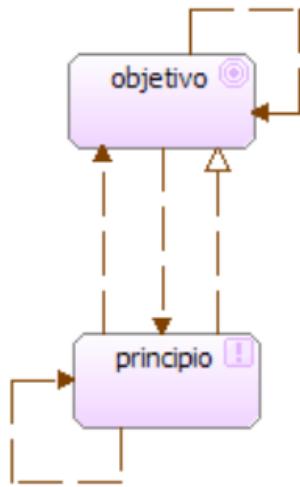


Figura 8.7: Metamodelo Punto de Vista de Principios

El punto de vista de los principios permite al analista o diseñador modelar los principios que son relevantes para el problema de diseño en cuestión, incluyendo los objetivos que motivan estos principios. Además, las relaciones entre los principios y sus objetivos pueden ser modeladas.

8.5.2 Caso de Estudio

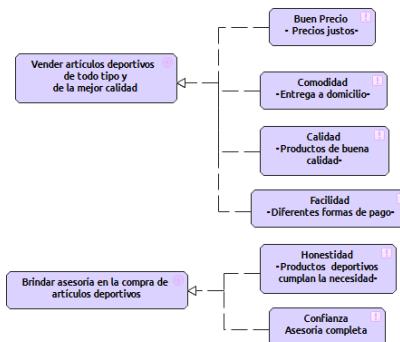


Figura 8.8: Modelo Punto de Vista de Principios

En la figura 8.8 se puede observar los dos objetivos principales del problema y los objetivos que motivan estos principios, en este caso al objetivo de vender artículos deportivos vemos que lo soportan los principios de un buen precio que sea justo para el cliente y la tienda, la comodidad de recibir el producto en su ubicación, el principio de la mejor calidad en cada uno de los productos que se ofrecen y como último la facilidad que se le brinda al cliente de pagar de diferentes formas. El objetivo de brindar asesoría al cliente se basa en los principios de honestidad y confianza que puede tener el cliente de que su proceso de compra sera totalmente claro y lo que se le recomienda es lo que le conviene.

8.6 Punto de Vista de Realización de Requerimientos

8.6.1 Modelo

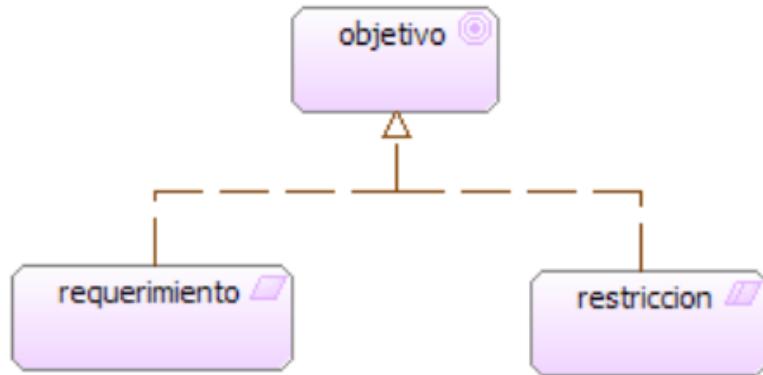


Figura 8.9: Metamodelo Punto de Vista de Realización de Requerimientos

El punto de vista de la realización de los requisitos permite al diseñador modelar la realización de los requisitos por parte de los elementos básicos, como los actores empresariales, los servicios empresariales, los procesos empresariales, los servicios de aplicación, los componentes de la aplicación, etc. Además, este punto de vista puede usarse para refinar requisitos en requisitos más detallados. La relación de agregación se utiliza para este propósito.

8.6.2 Caso de Estudio

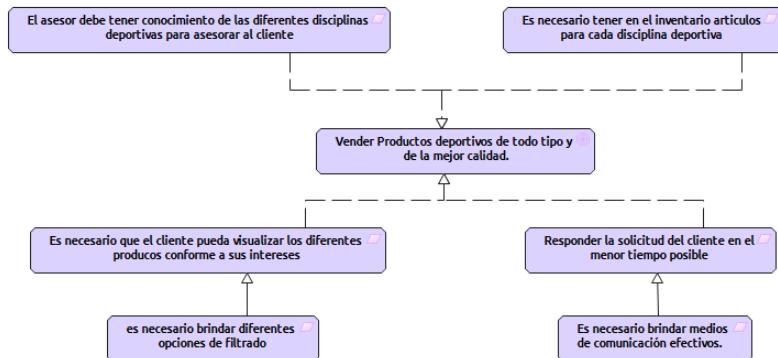


Figura 8.10: Modelo Punto de Vista de Realización de Requerimientos

En este punto de vista se puede observar cada uno de los requerimientos que soportan el objetivo de vender artículos deportivos de todo tipo para cada disciplina deportiva y de la mejor calidad, en la figura 8.10 se pueden observar 4 requerimientos principales, el primero se refiere a que el asesor debe contar con los conocimientos necesarios para poder brindar una correcta asesoría al cliente, el segundo busca garantizar que el cliente encuentre lo que busca en la tienda virtual, el tercero se refiere a la forma en que el cliente puede buscar los productos y la forma en que este selecciona lo que le interesa para este es necesario que garantizar diferentes tipos de filtros y el último de los requerimientos se refiere a la velocidad con que un asesor responde la solicitud de un cliente, de este requerimiento se extiende el requerimiento de contar con un buen medio de comunicación.

8.7 Punto de Vista de Motivación

8.7.1 Modelo

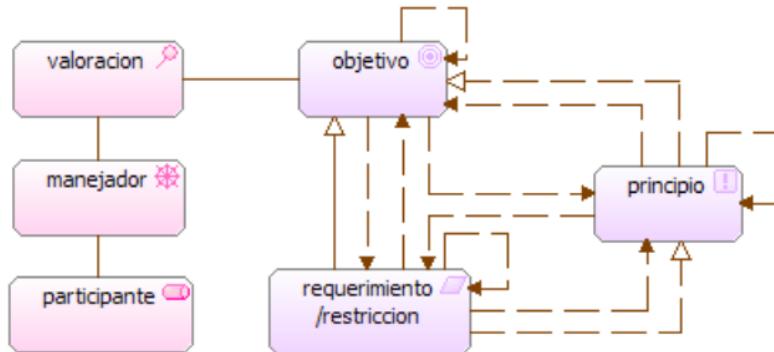


Figura 8.11: Metamodelo Punto de Vista de Motivación

El punto de vista de la motivación permite al diseñador o analista modelar el aspecto de la motivación, sin centrarse en ciertos elementos dentro de este aspecto. Por ejemplo, este punto de vista puede utilizarse para presentar un panorama completo o parcial del aspecto de la motivación relacionando a las partes interesadas, sus objetivos principales, los principios que se aplican y los principales requisitos de servicios, procesos, aplicaciones y objetos.

8.7.2 Caso de estudio

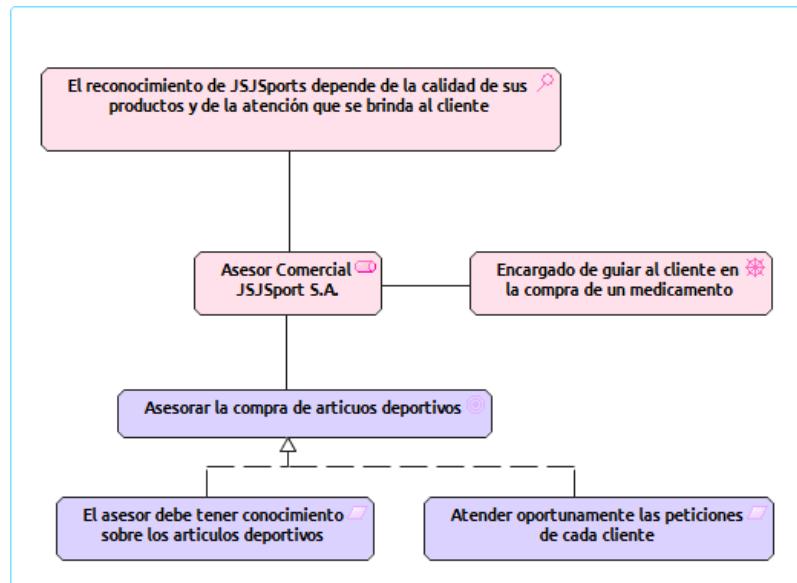


Figura 8.12: Modelo Punto de Vista de Motivación

En este punto de vista se puede observar un panorama de todo el aspecto de motivación del proyecto en este caso se ve al stakeholder Asesor Comercial, con su valoración y su manejador respectivo, además de esto se ve el objetivo principal con el que este cumple y los requerimientos que lo soportan en este caso, para el asesor, los requerimientos son conocer los artículos deportivos y las disciplinas para las que estos se ofertan y el de atender oportunamente las peticiones de cada cliente.

9. Proyecto

9.1 Introducción

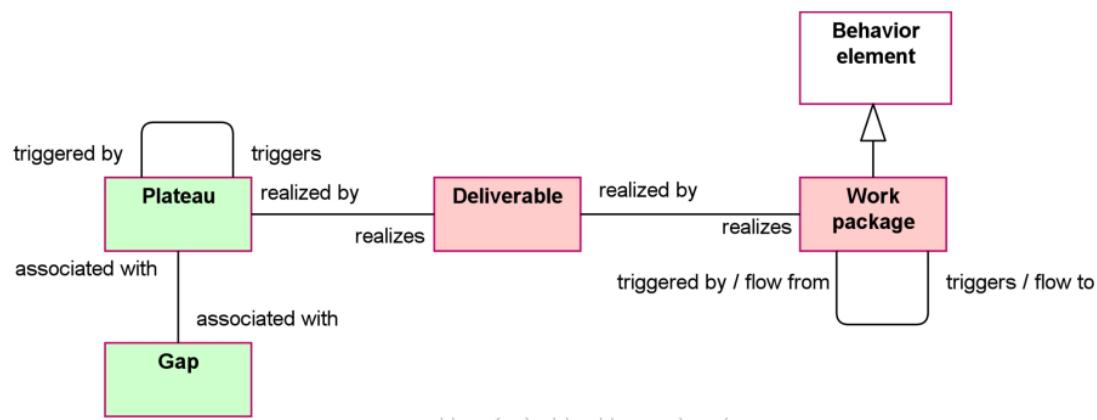


Figura 9.1: Capa Proyecto

Conceptualmente, la capa de proyecto es similar a un proceso de negocio, en el sentido de que consiste en un conjunto de tareas relacionadas causalmente, dirigidas a producir un resultado bien definido. Sin embargo, la capa de proyecto es un proceso único. Sin embargo la capa de proyecto se puede describir de una manera muy similar a la descripción de un proceso.

9.2 Punto de Vista de Proyecto

9.2.1 Modelo

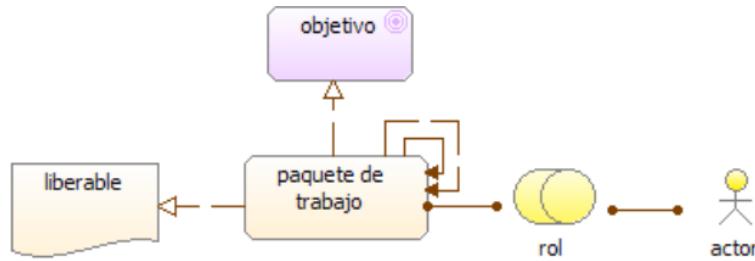


Figura 9.2: Metamodelo Punto de Vista Proyecto

Un punto de vista del proyecto se utiliza principalmente para modelar la gestión del cambio de arquitectura. La "arquitectura" del proceso de migración, desde una situación antigua (arquitectura de la empresa estatal actual) hasta una nueva situación deseada (arquitectura empresarial estatal objetivo) tiene consecuencias significativas en la estrategia de crecimiento a medio y largo plazo y en el proceso de toma de decisiones.

9.2.2 Caso de estudio

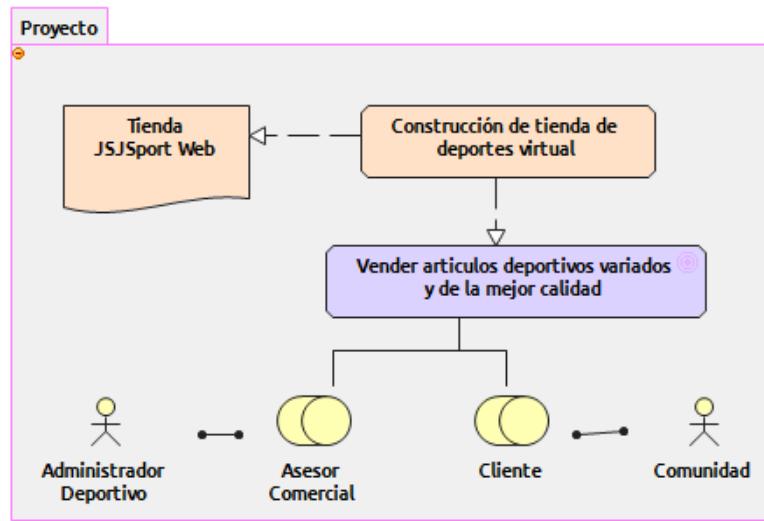


Figura 9.3: Modelo Punto de Vista Proyecto

En la figura 9.3 se puede observar que el paquete de trabajo es la construcción de la tienda de deportes virtual para JSJSports Web lo cual es el entregable de todo el proyecto, es decir este es el producto o resultado final al que se espera llegar, con el paquete de trabajo se busca cumplir el objetivo e vender artículos deportivos de la mejor calidad, este proceso a su vez involucra dos roles, el primero es el de asesor comercial que es interpretado por un administrador deportivo que tenga los conocimientos necesarios, el último rol es el de cliente, este rol es muy importante puesto que representa a toda la comunidad que accede a la tienda virtual.

9.3 Punto de Vista de Migración

9.3.1 Modelo



Figura 9.4: Metamodelo Punto de Vista Migración

El punto de vista de la migración implica modelos y conceptos que pueden usarse para especificar la transición de una arquitectura existente a una arquitectura deseada.

9.3.2 Caso de estudio

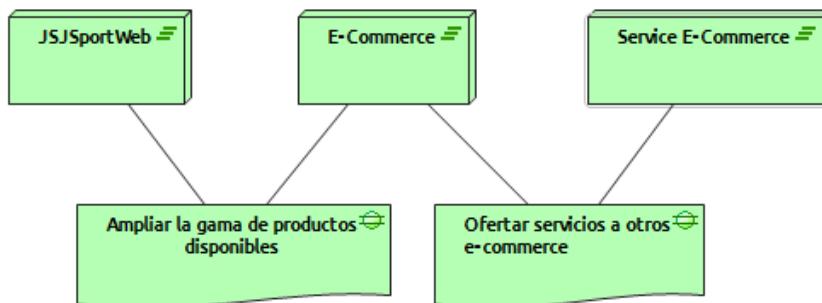


Figura 9.5: Modelo Punto de Vista Migración

En este punto de vista podemos ver la arquitectura que se ha venido modelando hasta el momento que se ve representada en el hito de JSJSports web, como equipo de trabajo hemos visto y planteado como trabajo futuro 2 arquitecturas diferentes una consecuencia de la otra, la primera de estas es convertir a JSJSports web en una plataforma e-commerce general que no solo se aplique a artículos deportivos sino que en esta se pueda ofrecer cualquier tipo de artículo y se cuente con un vendedor que brinde soporte al cliente durante el proceso de compra, para esto es necesario aumentar el número de módulos y de productos disponibles, es decir diversificar la página.

Para el caso de el último de los hitos, la arquitectura deseada busca hacer que JSJSports Web proporcione un servicio para diferentes páginas, es decir que no sea necesario que el cliente se encuentre en el dominio de la página sino que este pueda consultar productos por medio de otras o de terceros.

9.4 Punto de Vista de Implementación y Migración

9.4.1 Modelo

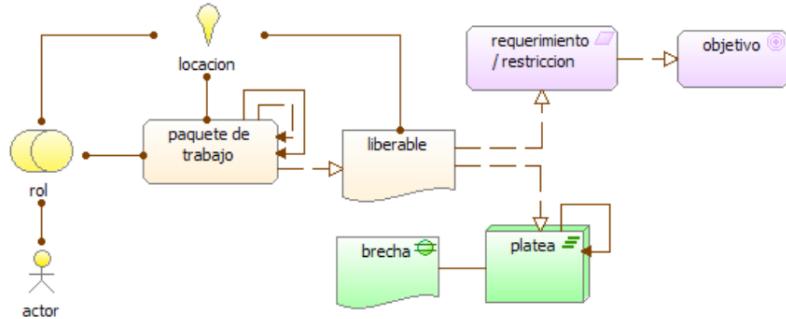


Figura 9.6: Metamodelo Punto de Vista Migración e Implementación

El punto de vista de migración e implementación se utiliza para relacionar programas y proyectos con las partes de la arquitectura que implementan. Esta visión permite modelar el alcance de los programas, proyectos, actividades del proyecto en términos de las mesetas que se realizan o los elementos de arquitectura individuales que se ven afectados. Además, la forma en que se afectan los elementos puede indicarse anotando las relaciones.

9.4.2 Caso de estudio

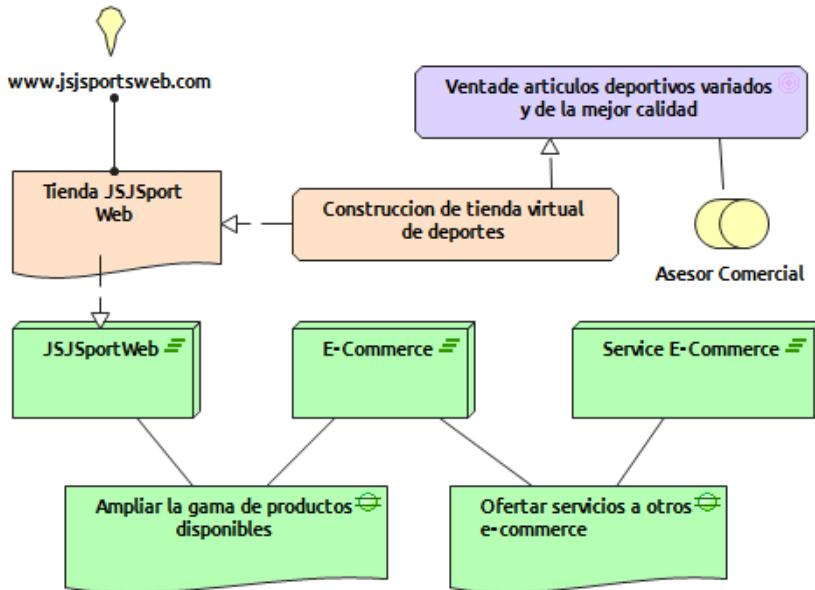


Figura 9.7: Modelo Punto de Vista Migración e Implementación

En la figura 9.7 se puede observar el punto de vista de Migración e implementación que une los puntos de vista de las figuras 9.3 y 9.5, aquí se unen estos puntos de vista por medio del entregable del proyecto y el la primera arquitectura hito puesto que ambos referencian a la tienda de JSJSports que se modela, además se especifica que todo esto se hará bajo el dominio de www.JSJSportsWeb.com.



10. Diseño

10.1 Introducción

El propósito de la gestión de requerimientos es asegurar que el proyecto cumple con las expectativas de sus clientes y en general de todos sus interesados, tanto externos como internos, siendo este el proceso que garantiza el vínculo entre lo que esperan los clientes y usuarios, y lo que los equipos de proyecto tienen que desarrollar. Si bien muchos de sus principios pueden ser adaptados a todo tipo de proyectos, es en los proyectos de desarrollo de software donde adquieren todo su sentido, garantizando el proceso y sirviendo de referencia para asegurar y controlar los cambios que en el proyecto puedan surgir (trazabilidad).

Es fundamental para los desarrolladores tener una idea clara sobre el funcionamiento del programa o sistema que se va realizar para ello se basan en herramientas que les permiten realizar el análisis de requerimientos para complacer las necesidades del cliente, a continuación veremos los resultados del proceso de análisis de dichos requerimientos para nuestro problema en particular.

10.2 Requerimientos

Los principales requerimientos se presentaran a continuación en las siguientes tablas

Cuadro 10.1: REQ EXPANDIR NEGOCIO

ID del requisito	REQ EN	
Versión	1	
Autores	Sebastian Piedrahita, Juan Cubillos	
Descripción	Las empresas PYME desean ampliar las ventas de sus negocios mediante la venta de sus productos a través de plataformas web.	
Precondicion		
Secuencia normal	Paso	Acción
	1	Crear una plataforma web para las empresas PYME en la que puedan publicitar sus productos y llegar a cualquier tipo de población deseada.
Postcondicion	REQ REL, REQ CC y REQ PP	
Excepciones	Paso	Acción
	1	Otros países
Importancia	Primordial	
Urgencia	Primordial	
Comentarios		

Cuadro 10.2: REQ PUBLICITAR PRODUCTO

ID del requisito	REQ PP	
Versión	1	
Autores	Sebastian Piedrahita, Juan Cubillos	
Descripción	Mediante el uso de una plataforma web, las empresas PYME podrán publicitar sus productos mostrando sus características y el inventario del mismo.	
Precondicion	REQ EN	
Secuencia normal	Paso	Acción
	1	Establecer id del producto
	2	Subir imagen del producto a publicar
	3	Indicar las características del producto ya seleccionado
	4	Indicar el inventario del producto
	5	Publicar Producto
Postcondicion		
Excepciones	Paso	Acción
		El producto agregado ya existe
	1	Modificar publicación del producto
	2	Cambiar características
	3	Modificar inventario
	4	Publicar el producto con los nuevos cambios
Importancia	Primordial	
Urgencia	Primordial	
Comentarios		

Cuadro 10.3: REQ COMUNICACIÓN CON CLIENTES

ID del requisito	REQ EN	
Versión	1	
Autores	Sebastian Piedrahita, Juan Cubillos	
Descripción	Las empresas PYME desean no perder la comunicación y asesoría que se le da al cliente en el proceso de compra.	
Precondicion	REQ EN	
Secuencia normal	Paso	Acción
	1	Crear una sección de discusión dentro de la plataforma que permita la comunicación entre asesores y clientes.
Postcondicion		
Excepciones	Paso	Acción
		El cliente no esta registrado
Importancia	Primordial	
Urgencia	Primordial	
Comentarios		

Cuadro 10.4: REQ RECAUDO EN LINEA

ID del requisito	REQ CC	
Versión	1	
Autores	Sebastián Piedrahita, Juan Cubillos	
Descripción	Una vez publicado el producto se podrá proceder a la compra del mismo, teniendo en cuenta la verificación del medio de pago y se procederá a pedir los datos del usuario para el posterior envío	
Precondicion	REQ EN	
Secuencia normal	Paso	Acción
	1	Seleccionar un producto
	2	Indicar el método de pago para la compra del producto
	3	Verificar el medio de pago ingresado por el usuario (tarjeta débito o crédito)
	4	Ingresar datos del usuario para el envío del producto
	5	Confirmar compra del producto y generar recibo.
	6	Recibir notificación de pago
	7	Despacho del producto
Postcondicion		
Excepciones	Paso	Acción
		El método de pago es incorrecto
	1	Cambiar método de pago
		El producto no se encuentra disponible
	1	Cancelar compra
	2	Seleccionar otro producto
Importancia	Primordial	
Urgencia	Primordial	
Comentarios		

10.2.1 Casos de Uso

Un diagrama de casos de uso nos sirve para observar el “QUE” del sistema, mostrando procesos que deberán ser resueltos según las necesidades (requerimientos) del cliente, en este caso el cliente requiere que el sistema PYME pueda expandir su negocio así ampliando el mercado del mismo, en el siguiente diagrama se plantea una solución al cliente enfocado en dicho sistema teniendo en cuenta tanto sus requerimientos como sus actores y qué papel desempeña en el sistema.

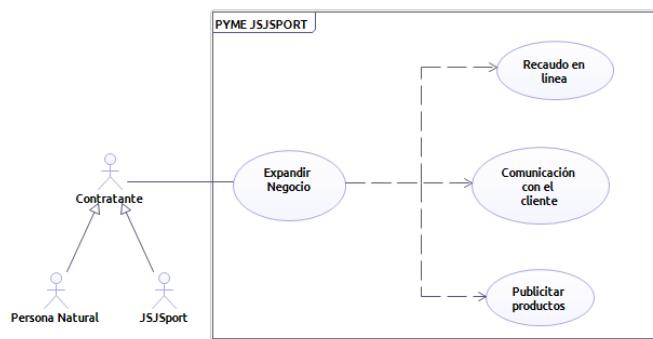


Figura 10.1: Diagrama de casos de Uso

El proceso de expandir el negocio, como se puede observar en la figura 10.1, incluye tres procesos importantes, el primero de ellos es que el sistema pueda publicitar productos, en otras palabras, brindar al cliente una plataforma web donde pueda exponer sus productos hacia al público deseado, teniendo en cuenta los aspectos como su inventario, apoyándose con una base de datos. El proceso de comunicación con el cliente consiste en todo lo relacionado al proceso de asesoría que se le brinda al cliente en la página, durante todo el proceso de compra que ese realice podrá consultar el cualquier momento a algún asesor cualquier duda o inquietud que le surja con respecto a los productos o a los procesos de pago.

El proceso de recaudo en línea, consiste en todo lo relacionado con la compra y venta del producto que se ofrecen en la plataforma web anteriormente generada, y verifica que el medio de pago ingresado por el usuario sea válido, una vez se compruebe el paso anterior se procederá a la confirmación de la compra, generando un comprobante de pago para la empresa y para el usuario para finalmente proceder a pedirle al usuario que ingrese los datos del domicilio.

10.3 Escenarios

En este sección observaremos el modelamiento desde los cimientos de nuestra plataforma, de la sección anterior concluimos el "QUE" del sistema o dicho en otras palabras que es lo que el cliente desea, a partir de la toma y organización de requerimientos procedemos a hacer un análisis del sistema que vamos a diseñar, a continuación observaremos diferentes herramientas que nos proveen unas bases para empezar a realizar nuestro sistema, conoceremos información actual y propondremos a grandes rasgos un modelo de posible solución a nuestro problema.

10.3.1 Diagrama de secuencia

Un diagrama de secuencia es una herramienta que proporciona al usuario y al desarrollador el "COMO" del sistema, esto se refiere a como se van a abordar los requerimientos obtenidos en nuestro primer capítulo, estos requerimientos, ahora llamados procesos deben ser abordados por los actores que interactuarán directamente con el sistema, y de esta manera que debe el sistema responder para llevar a cabo el proceso de manera óptima.

En nuestro caso los procesos que abordaremos para dar respuesta a esta pregunta son los que mostraremos a continuación.

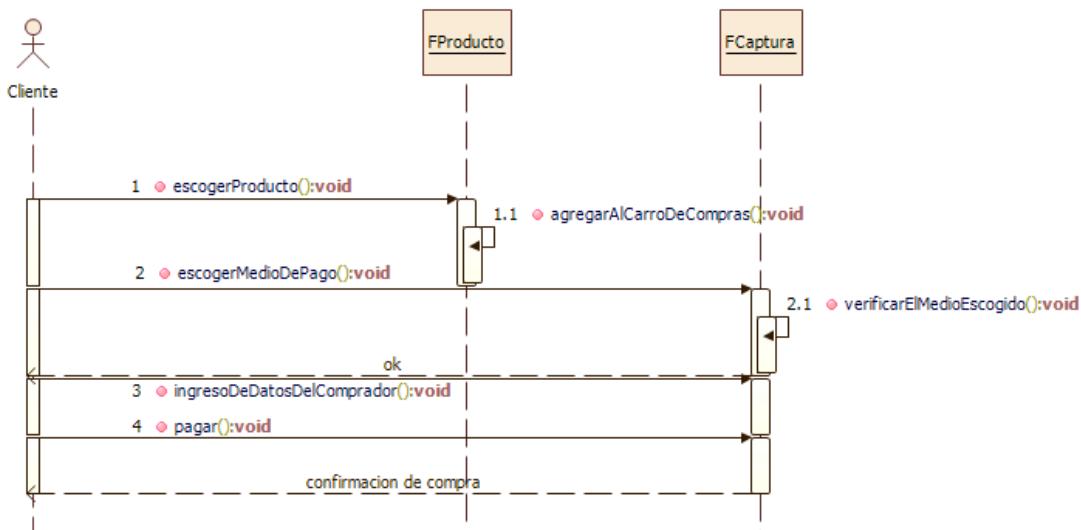


Figura 10.2: Diagrama de secuencia Recaudo en Línea

En el diagrama de secuencia anterior, se describe la secuencia entre el cliente, un formulario de captura y un formulario de producto. Inicialmente el cliente escogerá un producto, lo que genera un llamado a formulario producto, quien a su vez, hace un llamado recursivo para agregar al carro de compras, terminado este proceso el cliente escogerá un medio de pago, haciendo un llamado al formulario de captura, quien antes de retornar un mensaje satisfactorio, verificará el medio de pago escogido y retornará un ".ok", tras recibir el mensaje de satisfacción del medio de pago, el cliente ingresará sus datos en el formulario de captura y posteriormente seleccionará pagar, para lo cual recibirá una confirmación de compra.

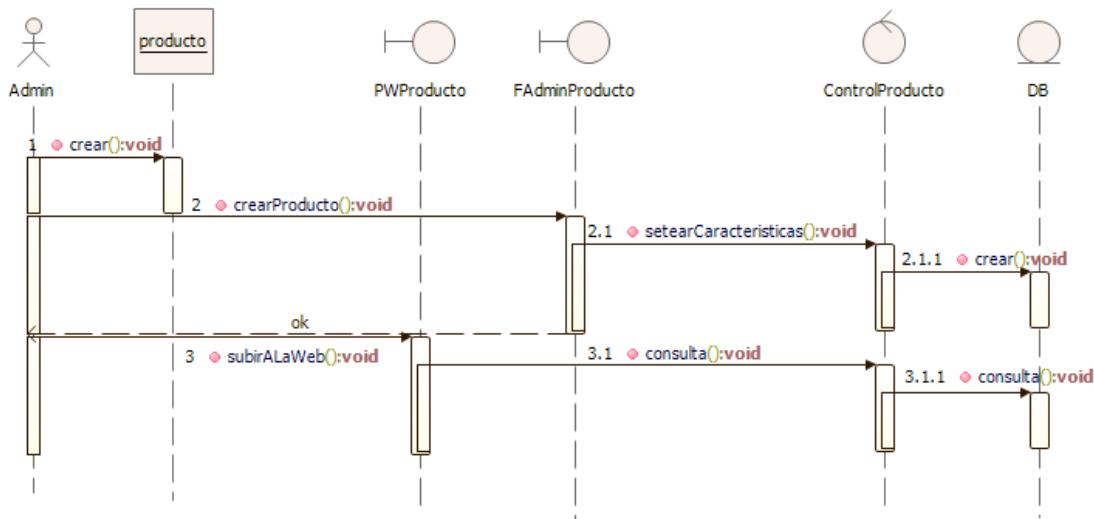


Figura 10.3: Diagrama de secuencia Publicitar Producto

En el diagrama de secuencia anterior, se describe el proceso para crear y publicar un producto en la plataforma web del cliente. Inicialmente el administrador (cliente) deberá crear el producto, posteriormente seleccionara crear el producto mediante un formulario de administrador, en el cual colocara (seteara) las características del mismo, las cuales pasaran por una clase de control de producto y este a su vez creara el producto en la base de datos, con lo cual el formulario de administración del producto, retornara un mensaje de satisfacción. Por último el administrador, seleccionara subir a la web, haciendo un llamado a la interfaz PW producto, quien hará una consulta del producto, por medio de la clase control producto, para hacer una consulta en la base de datos.

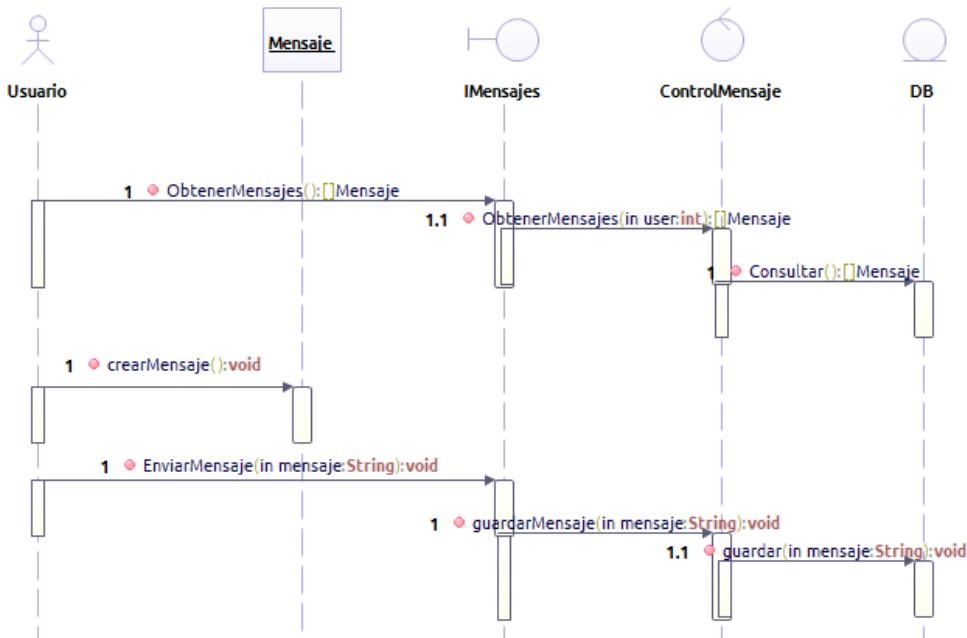


Figura 10.4: Diagrama de secuencia Comunicación con el cliente

En la figura 10.4 se puede observar el proceso que se realizara cuando un usuario independientemente de su rol (cliente/asesor) desee crear o establecer una comunicación con su contraparte, asesor->cliente y cliente->asesor. Inicialmente el usuario deberá ingresar al módulo de comunicación y allí se hará una consulta de todos los mensajes enviados y recibidos previamente y se mostrarán en estilo de conversación, posteriormente el usuario creara un mensaje redactando el contenido de este y lo enviará, este pasara por una interfaz de mensajes que accederá a la clase de control de mensajes y lo guardará en la base.

10.3.2 Diagrama de comunicación

Un diagrama de comunicación le sirve al desarrollador para modelar las interacciones entre los objetos que hacen parte de un sistema en términos de mensajes en secuencia, esto se refiere a todas las acciones que pueden realizar los objetos del sistema sobre otros objetos del mismo sistema, después de realizado un diagrama de secuencia se facilita la creación de los diagramas de comunicación.

A continuación mostraremos diagramas de comunicación de los procesos realizados en los anteriores diagramas de secuencia

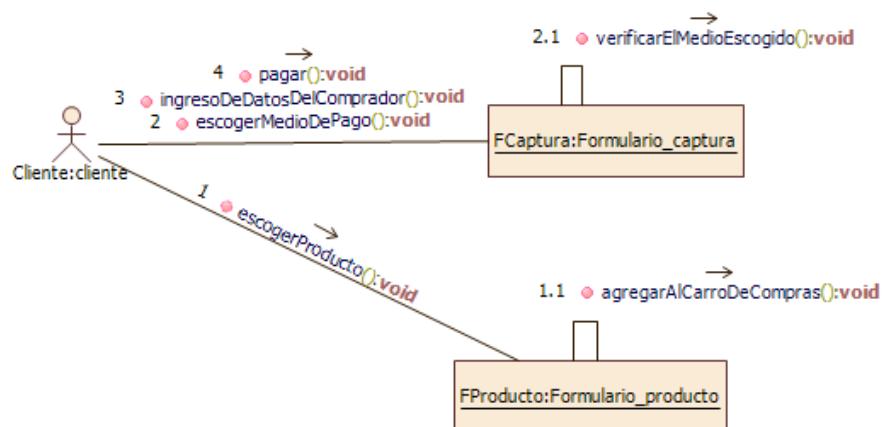


Figura 10.5: Diagrama de comunicación Recaudo en Línea

El diagrama de comunicación anterior, describe mediante que mensajes se comunica el cliente con las clases formulario de captura y formulario de producto. Con el formulario de captura se comunica mediante los mensajes, pagar, ingreso de datos del comprador, y escoger medio de pago, y con el formulario de producto se comunica mediante el mensaje escoger producto, adicionalmente el formulario de captura se comunica consigo mismo, mediante verificar medio de pago escogido, y el formulario de producto se comunica consigo mismo, mediante el mensaje agregar al carrito de compras.

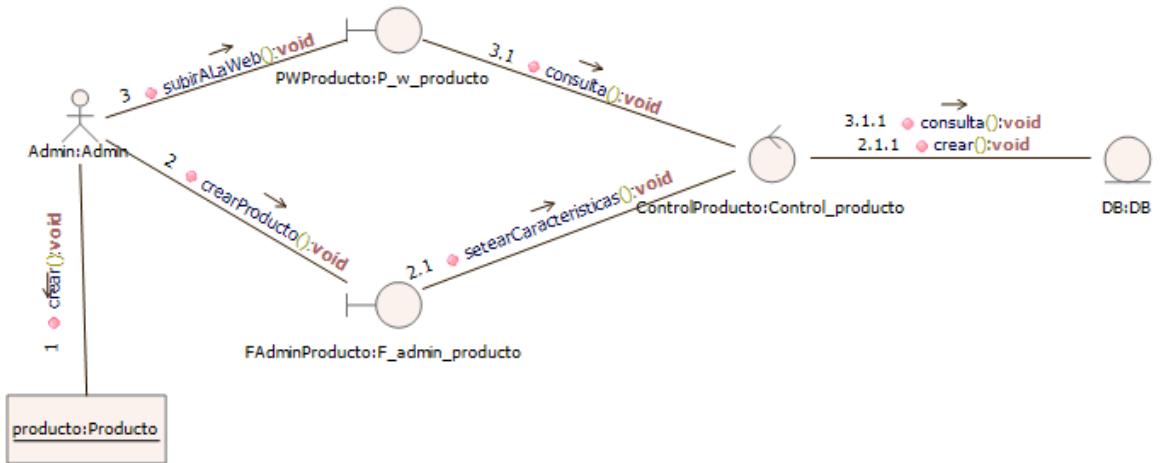


Figura 10.6: Diagrama de comunicación Promocionar Productos

El diagrama de comunicación anterior, muestra la comunicación entre el cliente (administrador) y la interfaz PW producto (mediante el mensaje subir a la web) y la interfaz Formulario de administración producto, mediante el mensaje crear producto, quienes se comunican con la clase control producto, mediante los mensajes de consulta y setear características respectivamente, por otro lado, el administrador se comunica con el objeto producto, mediante el mensaje de creación. Por ultimo la clase control producto, se comunica con la base de datos, con los mensajes de consulta y creación de productos.

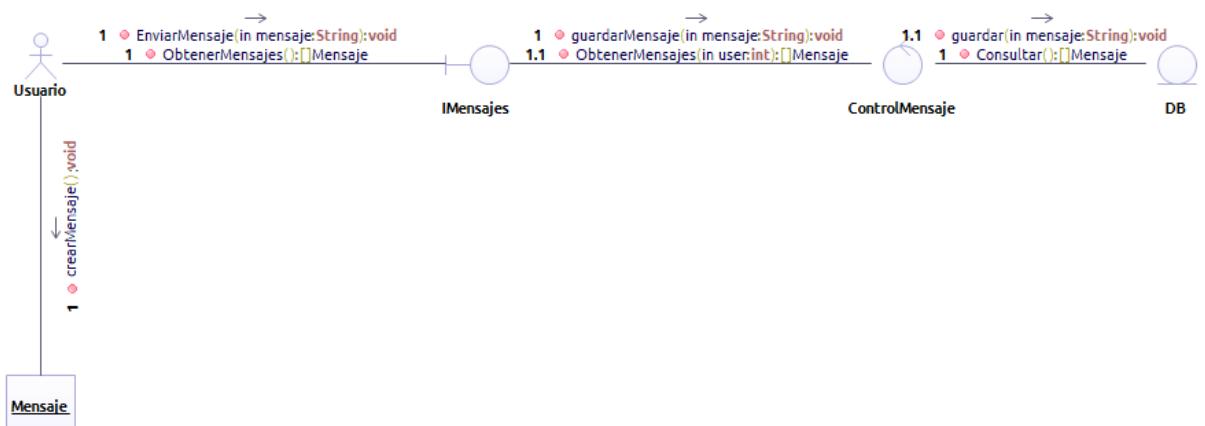


Figura 10.7: Diagrama de comunicación para Gestión de Mensajes

Como se puede ver en la figura 10.7 la comunicación entre el usuario y la interfaz de mensajes se da cuando el cliente desea consultar sus mensajes o enviar un mensaje que escribió, la interfaz de mensajes a su vez se conecta con el control del mensaje el cual permite hacer las diferentes comunicaciones a la base de datos conforme al proceso que se necesite.

10.4 Clases

Un diagrama de clases es una forma de visualización de código de programación llevado a un nivel más sencillo para que el desarrollador tenga un mayor nivel de comprensión, por medio de estos diagramas se evidencia que existen soluciones a problemas recurrentes, estos son llamados patrones, el uso de patrones en el desarrollo de software no solo optimiza el funcionamiento del sistema si no permite la escalabilidad y la agregación de nuevas funciones.

Para la solución propuesta al problema de la PYME JSJSports se presentan los siguientes diagramas de clases, usando patrones de diseño para el desarrollo de un sistema óptimo y escalable.

Diagrama de clases Fabrica Abstracta

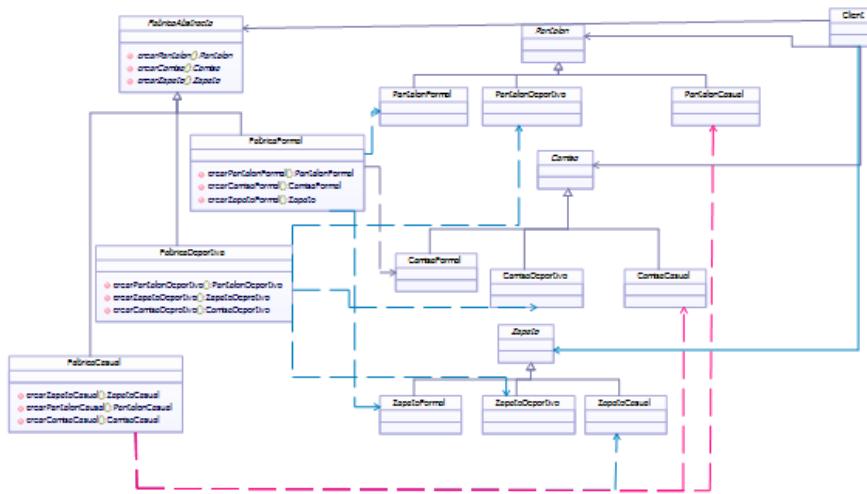


Figura 10.8: Diagrama de clases Fabrica Abstracta

En el diagrama podemos ver que, para nuestro caso tendremos diferentes líneas de productos que podrán ser registrados en la base de datos. Para esto usamos el patrón fabrica abstracta que nos permite crear familias de productos de acuerdo a la línea establecida, además este patrón permite el escalamiento del aplicativo en forma horizontal, con lo cual, podremos a futuro adicionar otras nuevas líneas o colecciones de productos, sin necesidad de modificar las líneas ya creadas.

Diagrama de clases Método Fábrica

Para el caso de estudio se puede observar que se implementó el método fabrica para facilitar la creación del tipo de usuarios que maneja el sistema, en este caso cada usuario tiene asociadas unas opciones, por ejemplo el usuario no registrado no puede realizar una compra, el usuario registrado si, entre otros, para crear un usuario para cada caso cuando se necesite y saber las opciones que este tiene asociadas se implementó este patrón.

Diagrama de clases Singleton

Para nuestro caso de estudio tenemos una clase ConexionDB que es análoga a la clase singleton, esta tiene un método llamado getConexion que se encarga de retornar la instancia ya creada de la clase o en el caso de que aun no exista ninguna instancia crear una nueva.

La idea de aplicar este patrón en la conexión, es para que una vez se haya establecido una conexión

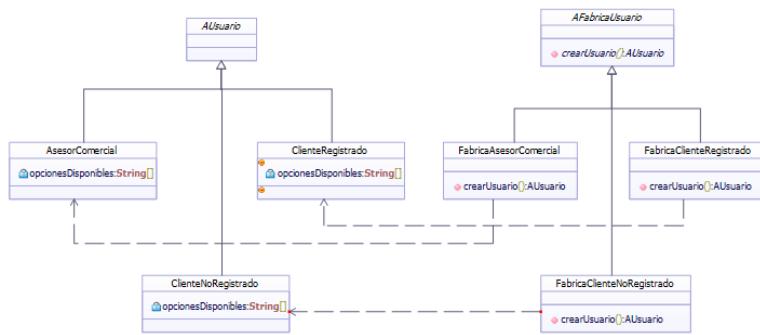


Figura 10.9: Diagrama de clases Patrón Método Fábrica

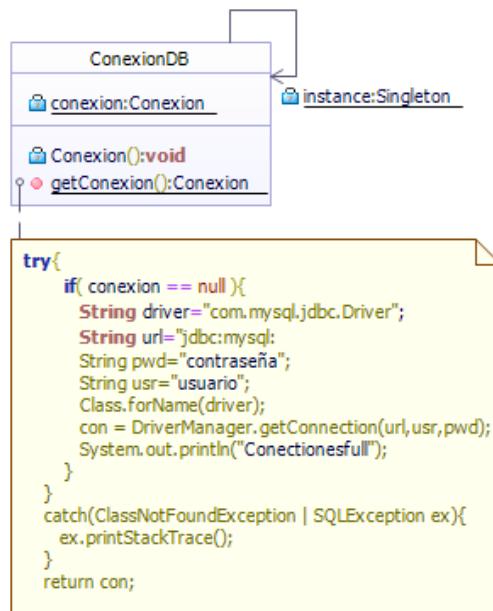


Figura 10.10: Diagrama de clases Patrón Singleton

a la base de datos, se trabaje sobre la misma sesión y no se creen muchas conexiones dependiendo de lo que se vaya a realizar

Diagrama de clases Patrones Comando y Puente

El patrón comando utilizado en este caso permite separar y simplificar el uso de los botones y la función de cada uno en distintas opciones presentadas al usuario, para este caso particular están definidos los casos en los que el Admin crea, modifica, elimina o agrega un nuevo producto a la base de datos. El patrón bridge, es un puente entre la abstracción de una base de datos y sus funciones con la lógica de estas funciones de acuerdo con el tipo de base de datos, puesto que la manera en que se borra, crea, agrega o modifica es diferente en cada base de datos (MySQL, Postgres, etc), de manera que en la base concreta se define el método en que cada función de la abstracción, realiza su operación. Esto permite que más adelante podamos agregar otras bases de datos a nuestro aplicativo sin la necesidad de modificar el código ya creado, y haciendo escalamiento de nuestro programa de manera horizontal.

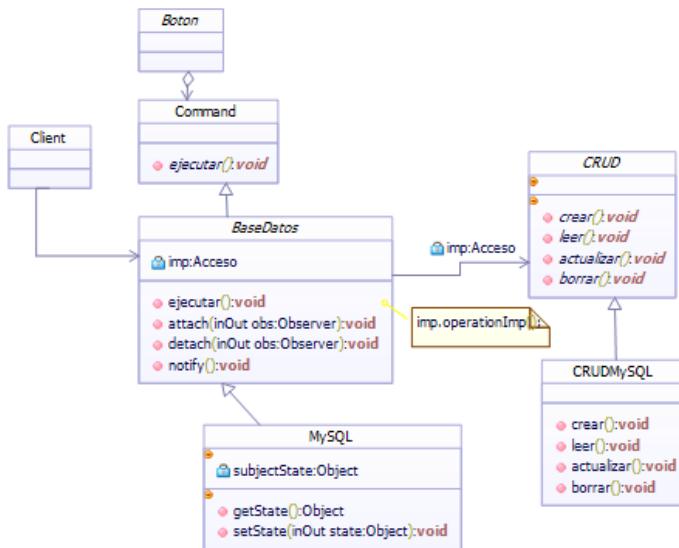


Figura 10.11: Diagrama de clases Comando y Puente

Diagrama de clases Patrón Fachada

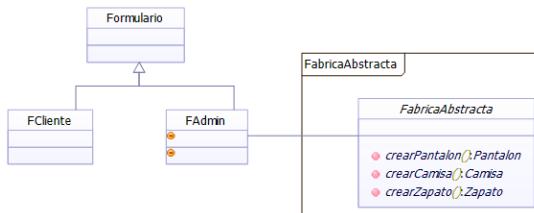


Figura 10.12: Diagrama de clases Patrón Fachada

Este patrón nos permite ocultar al cliente y al admin lo que sucede detrás del formulario que utilizan, es decir oculta la lógica del programa, la cual no es interesante para ellos, podemos incluir varios subsistemas con este patrón y dejarlos solo visibles para los que estén realmente interesados en cada uno.

Diagrama de clases Patrón Iterador

En este caso se puede observar en la figura 10.13 que se hace uso del patrón iterador para iterar por cada uno de los elementos del carrito de compras. En este caso nuestro cliente es cada uno de los productos que pertenecen al carrito, el carrito abstracto tiene una lista de productos que son los escogidos por el cliente, cuando el cliente manipula el carrito de compras se le permite agregar o eliminar productos de este, una vez este ha decidido que desea comprar todos los elementos que tiene en el carrito es necesario para generar la factura y seguir con el proceso de pago saber el total del valor de los productos que el cliente escogió, para esto se hace uso del IteradorCarrito el cual recorre el arreglo de productos del carrito preguntando si hay más elementos y obteniendo el valor del producto actual multiplicando el valor de la unidad por la cantidad escogida por el cliente; una

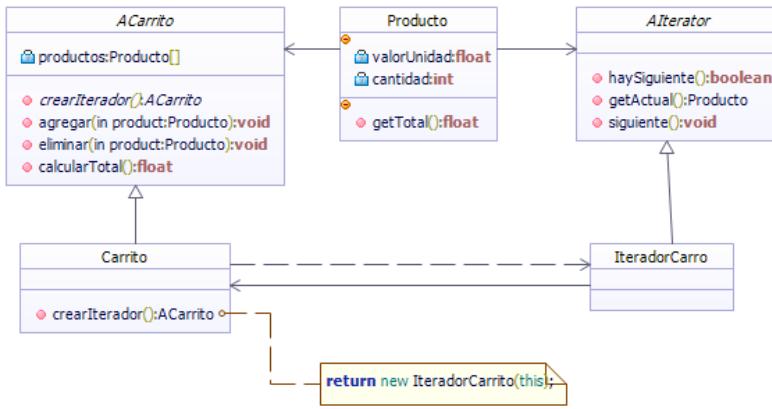


Figura 10.13: Diagrama de clases Patrón Iterador

vez el iterador define que no hay más elementos siguientes se puede mostrar el valor del total al cliente y continuar con el proceso de pago.

Diagrama de clases Patrón Mediador

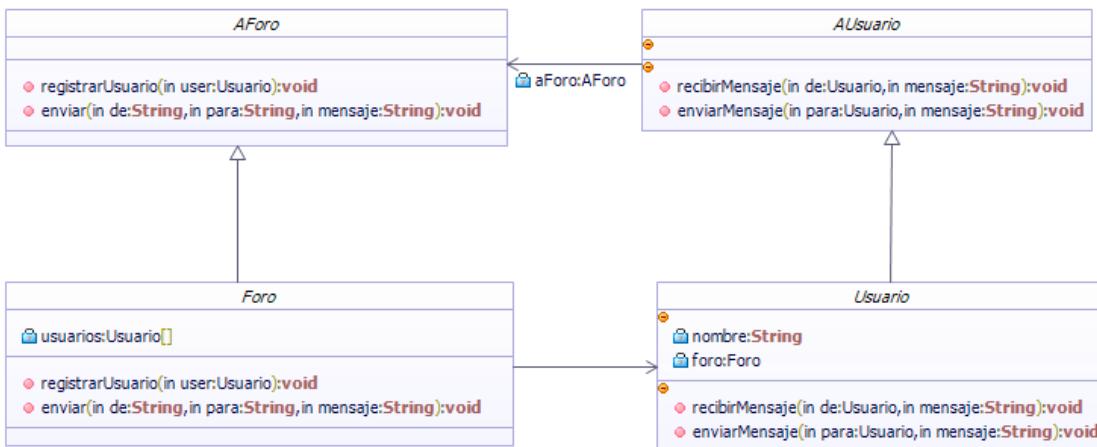


Figura 10.14: Diagrama de clases Patrón Mediador

Para este caso en específico se hace uso de patrón observador en su ejemplo más común no por simplicidad sino porque haciendo uso de este se satisface perfectamente uno de los principales requerimientos del sistema el cual es brindar un mecanismo para que a pesar de ser una tienda virtual los usuarios puedan seguir comunicándose con los asesores.

En este caso se plantea una interfaz abstracta para cada usuario que tiene dos métodos, enviar y recibir mensaje, esta se implementa en cada usuario que a su vez tiene un nombre que lo identifica en el foro y le permitirá distinguirse de los demás y ademas conoce el foro. La interfaz abstracta `AForo` tiene dos métodos que permiten a los usuarios suscribirse al foro y el método enviar que es

donde se realizara todo el proceso de registro y envio del mensaje, esta interfaz la implementa la Clase Foro que además tiene un arreglo de usuarios que son los usuarios que pertenecen al chat.

Diagrama de clases Patrón Memento

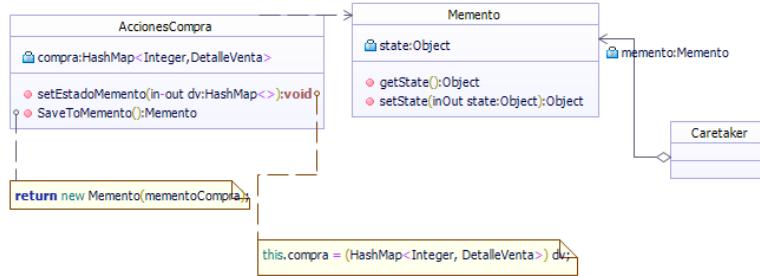


Figura 10.15: Diagrama de clases Patrón Memento

Para nuestro caso de estudio el patrón memento tiene las siguientes clases, una clase **AccionesCompra**, que se encarga de manejar las acciones relacionadas al carrito de compra, como agregar productos, eliminarlos, confirmar compra entre otras, cada vez que se visualizan los productos agregados al carrito, se genera un detalle de la venta, que tiene los productos y la cantidad que se va a comprar, es decir el estado del carrito, la clase **memento** tiene un estado que es el estado interno de la clase **AccionesCompra** en momento dado, y la clase **CareTaker** es la encarga de almacenar cada uno de los estados internos de la clase **AccionesCompra** y retornarlos cuando sea necesario. El objetivo del uso de este patron en estas clases, es que una vez se esten visulizando los productos agregados al carrito de compras, y se eliminien por error se pueda deshacer esta accion y retornar al estado anterior del carrito, asi mismo tambien se podra rehacer la accion devolviendo el carrito a su ultimo estado.

Diagrama de clases Patrón Observador

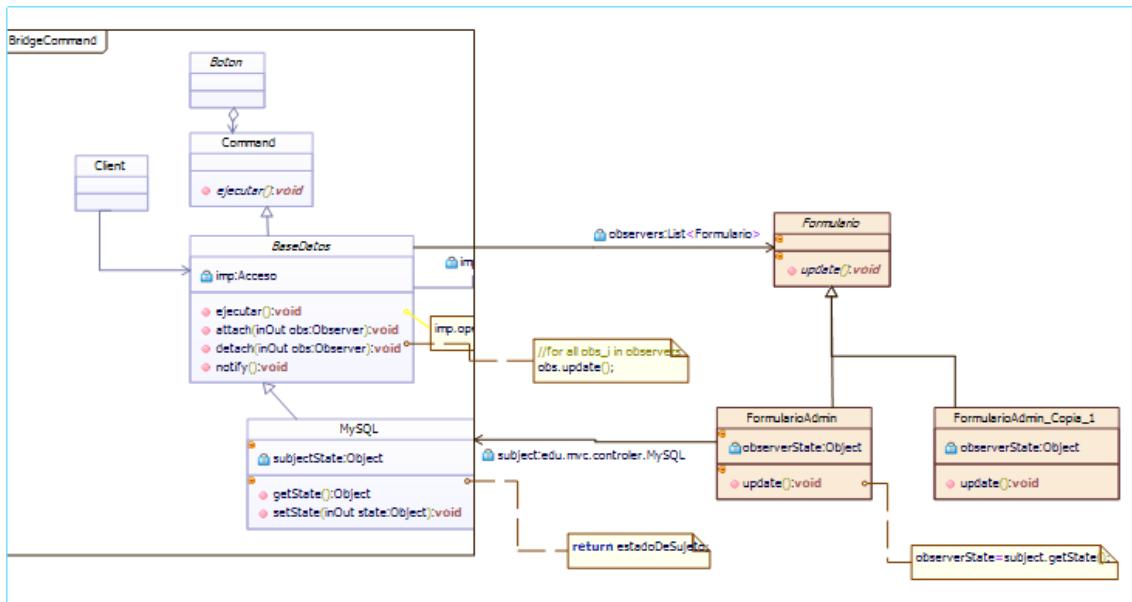


Figura 10.16: Diagrama de clases Patrón Observador

El patrón observador es utilizado en nuestro caso debido a la necesidad que tenemos de estar actualizando constantemente los datos de los productos en los formularios del Admin y el Cliente, ya sea actualizando el estado del producto en el stack, o alguna de sus características cada vez que entramos a modificarlas, este patrón nos permite facilitar esta actualización directamente en nuestra base de datos.

Diagrama de clases Patrón Estado

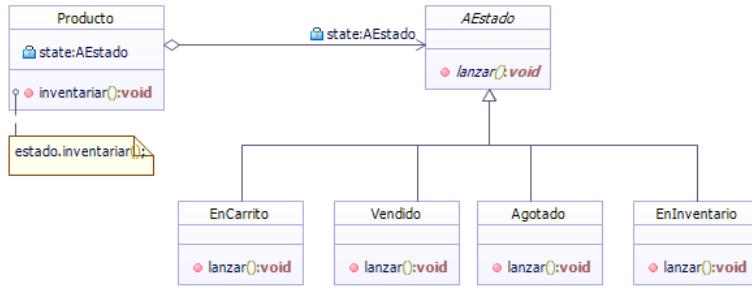


Figura 10.17: Diagrama de clases Patrón Estado

En el anterior Diagrama de estados podemos ver la transición que tienen los productos desde el momento en que se cargan a la página, hasta que se realiza exitosamente una compra, para esto nos basamos en 4 estados posibles, en inventario, en carrito, vendido, agotado, el producto empieza en un estado base de en inventario con una cantidad disponible, cuando se agrega el producto a carrito se cambia de estado a en carrito, en donde tiene dos posibles disparadores, que son desagregar de carrito y confirmar compra, si se desagrega del carrito, regresara a su estado base en inventario, si se confirma la compra cambiara de estado ha vendido, en donde se actualizara el inventario, si la cantidad comprada es igual a la disponibles en inventario, el producto quedara en un estado de agotado, si aún hay unidades disponibles volverá a un estado de inventario, a partir de ahí podemos crear una clase producto la cual podrá contener cualquiera de los 4 estados

Diagrama de clases Patrón Estrategia

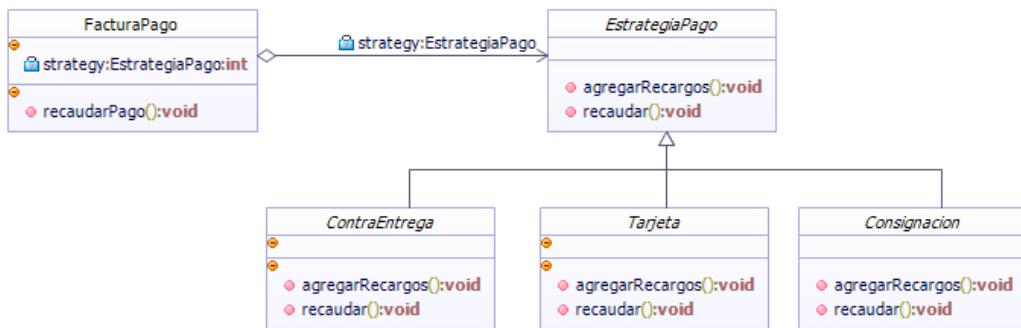


Figura 10.18: Diagrama de clases Patrón Estrategia

Como se puede observar en la figura 10.18 en este caso se aplica la estrategia para solicitar al cliente el recaudo del pago, debido a que se tienen diferentes formas o posibilidades de pago y

cada pago se hace de forma diferente, en este caso la factura solicita recaudar el pago y conforme al método escogido por el cliente se ejecutará la estrategia correspondiente.

Debido a que cada medio de pago requiere agregar recargos diferentes, como es el caso de la forma de pago de consignación y de tarjeta que requieren agregar los impuestos correspondientes, o el caso del pago a contra entrega en donde es necesario agregar el recargo del mensajero que se encargará de entregar el producto al cliente y cobrar el valor correspondiente al total.

Al igual que los recargos que se agregan son diferentes, también son diferentes las formas de recaudar el pago para cada modalidad, en el caso de la contra entrega el usuario es redirigido a confirmar los datos de envío, el método de pago con tarjeta requiere comunicarse con la plataforma de pagos y la consignación redirigir al banco correspondiente.

El uso de este patrón nos permite que a futuro sea posible agregar nuevas formas de pago sin tener que modificar el código y la implementación ya existente.

10.5 Componentes

Un componente es un objeto dentro de un sistema que cumple una funcionalidad específica, en un diagrama de componentes se pueden observar varios componentes relacionados entre sí, los componentes cumplen la característica que son cajas negras y además pueden implementar interfaces entre ellos.

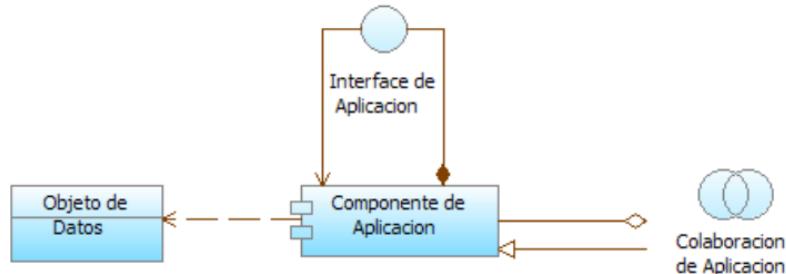


Figura 10.19: Metamodelo un Diagrama de Componentes

Caso de estudio

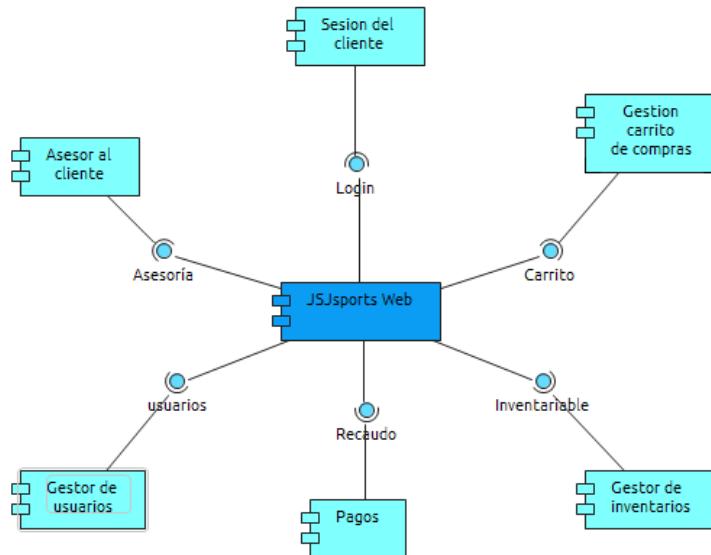


Figura 10.20: Diagrama de Componentes

En este diagrama de componentes podemos observar cada uno de los componentes del sistema y las interfaces que funcionan como pegamento para estos componentes con el con el componente central JSJSport Web, la interfaz asesoría permite brindar asesoría al usuario y es la que se encarga de realizar todo los procesos de comunicación, la interfaz login le permite al usuario administrar su sesión e ingresar o salir del sistema, la interfaz carrito ayuda a manejar el carrito de compras y los productos que desea el usuario. La interfaz recaudo permite gestionar los pagos de las compras realizadas, la interfaz de usuarios permite gestionar los usuarios y sus roles y permisos, por último la interfaz inventariable permite hacer la gestión de los inventarios y productos que tiene la PYME.

10.6 Nodos

Un diagrama de nodos nos permite como desarrolladores conocer los recursos físicos necesarios para que el sistema sea implementado, hay dos tipos de nodos, nodos dispositivo y procesador con los cuales tanto el desarrollador como el cliente puede interactuar, esto representa la comunicación y las relaciones entre el hardware para la modelación del sistema.

A continuación observaremos el diagrama de nodos para nuestra plataforma web:

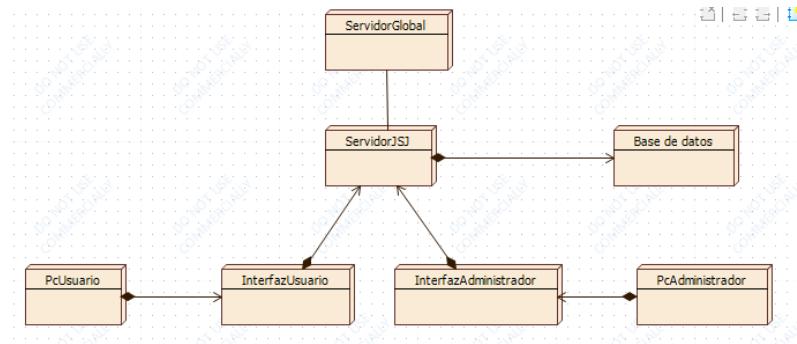


Figura 10.21: Diagrama de nodos

Como podemos observar en el caso de nuestro proyecto se observan 7 nodos de los cuales el nodo del servidor jsj, la base de datos, y las páginas de las interfaces de usuario y administrador son primordiales En el nodo del servidor jsj se tienen todos los componentes que permiten el funcionamiento de la página de ventas de artículos deportivos Entre estos componentes encontramos acciones que serán ejecutadas dependiendo del tipo de usuario que ingrese Para el caso de la interfaz administrativa, contara con acciones como agregar, modificar o eliminar productos, estableciendo una conexión con los componentes del servidor que permiten realizar dichas acciones En el caso de la interfaz de usuario se dispondrá de acciones como la compra de los productos, y todo el proceso que conllevará estableciendo una comunicación con el servidor jsj que llamará a dichos componentes El servidor jsj también necesita una comunicación con una base de datos que se encargará de almacenar todo tipo de información relacionada con la página, como los productos, los usuarios, información de los pedidos, etc Los otros nodos que podemos encontrar como pc de usuario y de admin son recursos físicos necesarios para la comunicación con la página web.

En el punto de vista de uso de infraestructura en la figura 7.5 se puede observar de forma más clara lo que se encuentra en el nodo del servidor JSJ, y las interacciones con las interfaces del administrador y del cliente, dentro del servidor de JSJ encontramos cada uno de los componentes definidos para el sistema integrados en función de satisfacer cada una de las necesidades de los usuarios.

10.7 Sistemas

También llamado diagrama de paquetes se observa la organización jerárquica y la relación que tienen los paquetes de los cuales se compone el sistema, a partir de este podemos observar el rendimiento del sistema por medio de la evaluación de métricas.

En el siguiente diagrama se evidencia la organización de cada una de las partes del proyecto y su ubicación en los diferentes paquetes que implementa el mismo, en este caso dentro de la aplicación web de JSJSports en el paquete de componentes encontramos todos los componentes definidos anteriormente dentro del paquete com. Además de los componentes encontramos el api y las utilidades que son los que nos permitirán hacer el pegamiento con los componentes y cargar cada uno de los mismos componentes necesarios para el funcionamiento.

En la parte inferior de la figura 10.22 se puede observar el paquete com, en donde básicamente se encuentra la presentación que tiene todos los aspectos de la vista, el repositorio que se encarga de la persistencia y el paquete más importante el cableado que se encarga de realizar todo el cableado de la aplicación y estructurarla.

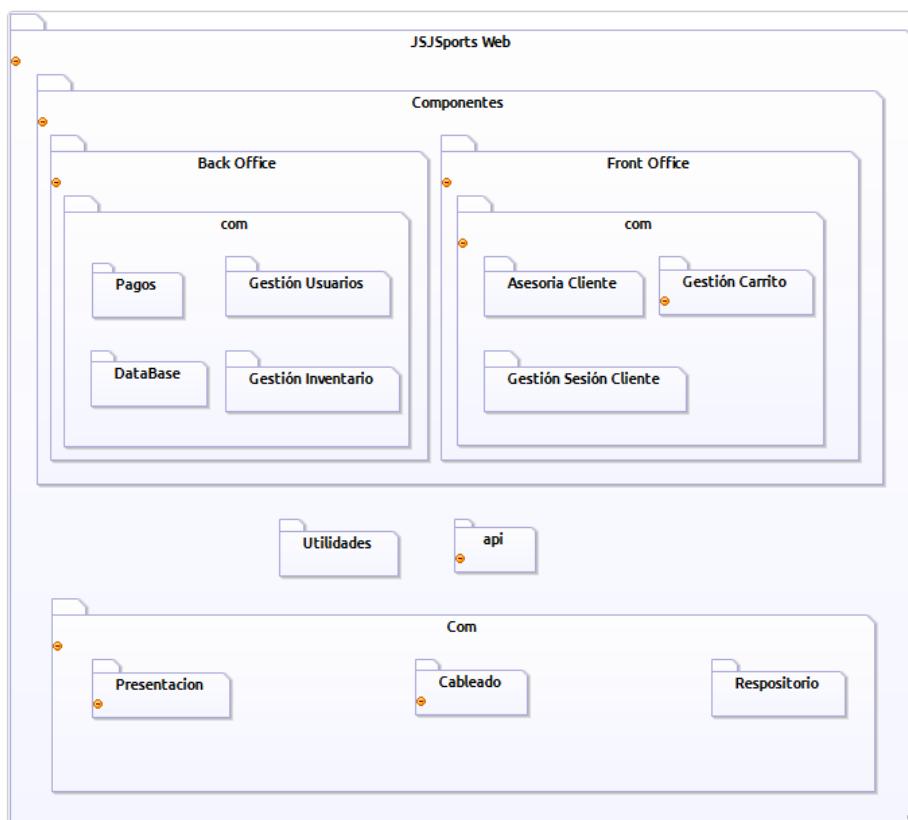


Figura 10.22: Diagrama de Sistemas

10.8 Diagrama de Actividades

Un diagrama de actividades permite al desarrollador observar cómo se está realizando el proceso y como se comunican los diferentes actores, dentro de dichas actividades puede haber decisiones pero siempre el proceso debe acabar.

Muestra un proceso de negocio o un proceso de software como un lujo de trabajo a través de una serie de acciones. Las personas, los componentes de software o los equipos pueden realizar estas actividades. Puede usar un diagrama de actividades para describir procesos de varios tipos, como los ejemplos siguientes:

- Un proceso de negocio o un flujo de trabajo entre los usuarios y el sistema.
- Los pasos que se realicen en un caso de uso.
- Un protocolo de software, es decir, las secuencias de interacciones entre componentes permitidos.
- Un algoritmo de Software

Caso de Estudio

En el siguiente diagrama de actividades se muestra como es el proceso de la compra de un producto, en la que intervienen 3 actores principales:

1. Comprador: Se encarga de escoger el producto, que desea comprar, seleccionando un método de pago y recibiendo la confirmación de compra del mismo.
2. Plataforma: Se encarga de presentar el producto al comprador, y es el enlace entre el comprador y el inventario, para la realización de cada acción.
3. Inventario: Se encarga de verificar la disponibilidad de cada producto, y actualizar el inventario del mismo.

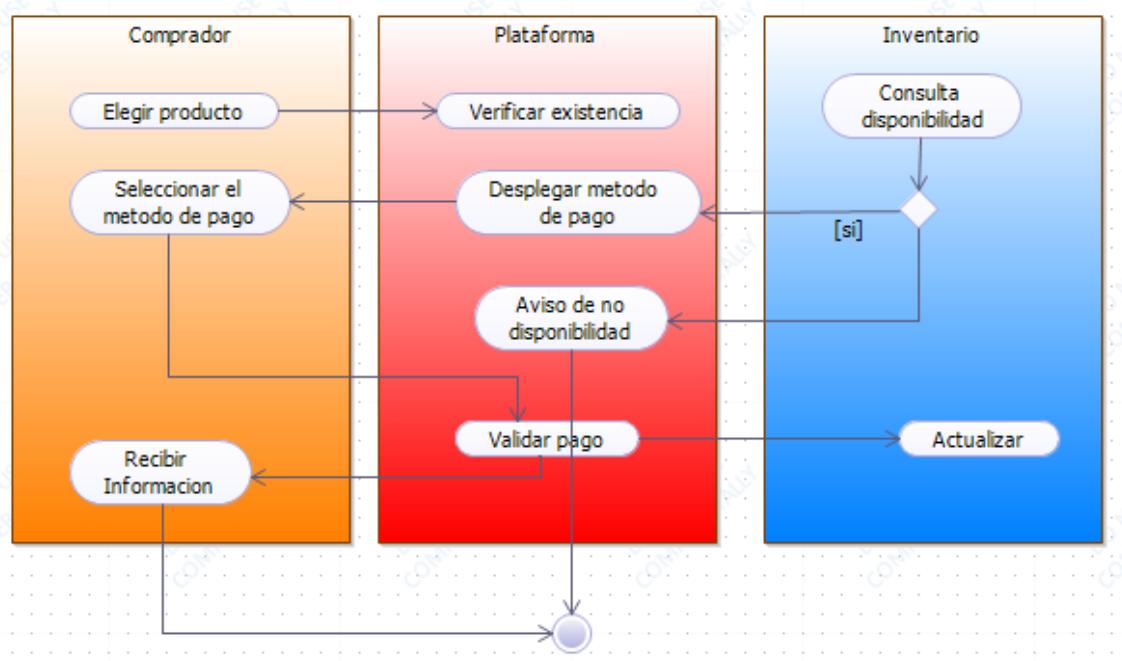


Figura 10.23: Diagrama de actividades para la compra de un producto

El proceso de compra de un producto empieza por el comprador que es el que escoge el producto deseado, enviando una petición a la plataforma, para verificar su existencia, la cual hace la consulta en el inventario para ver su disponibilidad, en este paso hay dos posibles opciones, en el caso de que no sea disponible, envía un aviso a la plataforma para que informe al comprador y se acaba la

acción, en el caso de que sea disponible, la plataforma muestra los métodos de pago disponibles, una vez que el comprador escoja un método de pago, se validara en la plataforma y se procederá a enviar la facturación al comprador y a la actualización del inventario del producto.

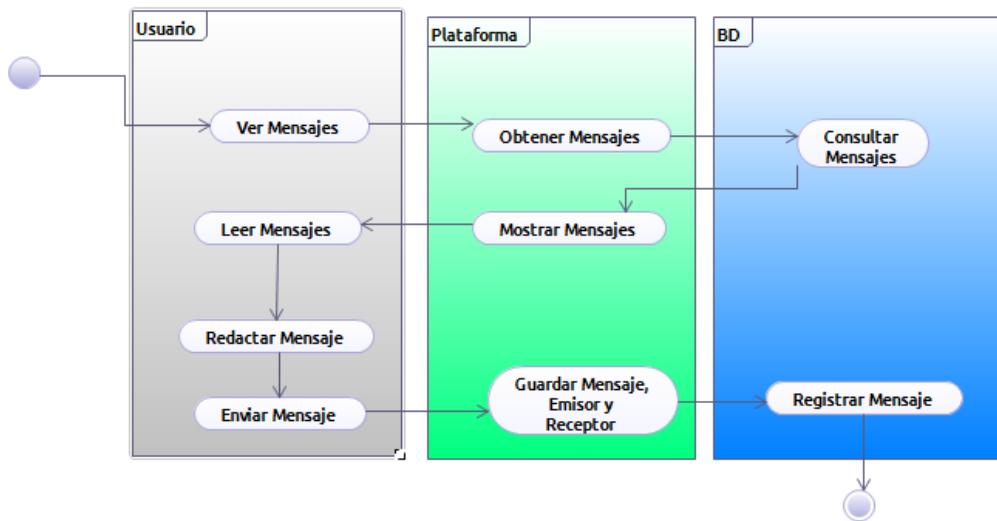


Figura 10.24: Diagrama de actividades para proceso de comunicación

En la figura 10.24 se puede observar las actividades implicadas en el proceso de comunicación de dos usuarios del sistema (asesor->cliente o cliente->asesor), para ver los mensajes es necesario que este se solicite ver los mensajes y la plataforma solicitará a la bd los mensajes del usuario y los mostrara, una vez el usuario lea los mensajes podrá redactar un mensaje y enviarlo, allí la plataforma solicitará a la base de datos registrar el respectivo mensaje.

10.9 Estados

Los estados son aquellas fases en las que el sistema puede estar, un diagrama de estados permite observar como son las transiciones entre estas fases, y cuáles son los lanzadores que con una condición activan estas transiciones, los estados son únicos.

A continuación veremos el diagrama de estados para el sistema de compra en línea planteado:



Figura 10.25: Diagrama de estado producto

En el anterior Diagrama de estados podemos ver la transición que tienen los productos desde el momento en que se cargan a la página, hasta que se realiza exitosamente una compra, para esto nos basamos en 4 estados posibles, en inventario, en carrito, vendido, agotado, el producto empieza en un estado base de en inventario con una cantidad disponible, cuando se agrega el producto a carrito se cambia de estado a en carrito, en donde tiene dos posibles disparadores, que son desagregar de carrito y confirmar compra, si se desagrega del carrito, regresara a su estado base en inventario, si se confirma la compra cambiara de estado ha vendido, en donde se actualizara el inventario, si la cantidad comprada es igual a la disponibles en inventario, el producto quedara en un estado de agotado, si aún hay unidades disponibles volverá a un estado de inventario, a partir de ahí podemos crear una clase producto la cual podrá contener cualquiera de los 4 estados como se muestra a continuación:

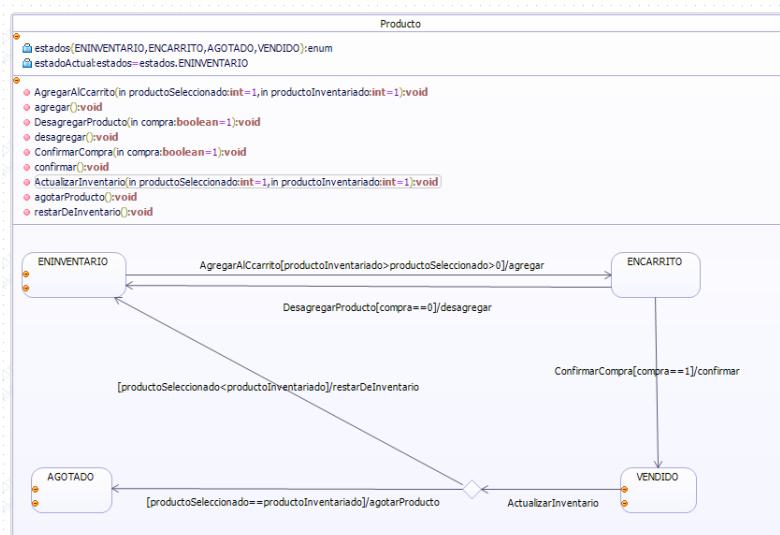


Figura 10.26: Clase producto estado

En este diagrama de clase se muestran los métodos necesarios para realizar las transiciones de un estado.



11. Patrones GoF

11.1 Introducción

Estos 23 patrones de diseño son los patrones más conocidos y usados en la actualidad en el campo del Diseño Orientado a Objetos.

En el año 1994, que apareció el libro “Design Patterns: Elements of Reusable Object Oriented Software” escrito por los ahora famosos Gang of Four (GoF, que en español es la pandilla de los cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Los integrantes de la pandilla de los cuatro recopilaron y documentaron 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. Desde luego que ellos no son los inventores ni los únicos involucrados, pero ese fue luego de la publicación de ese libro que empezó a difundirse con más fuerza la idea de patrones de diseño.

Los patrones de diseño el grupo de GoF clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento. [1]

“Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.”, es decir, los patrones de diseño brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Cuando se habla de un patrón de diseño es necesario tener presente los siguientes elementos que lo conforman:

- Nombre
- Problema: Define cuando se debe aplicar el patrón.
- Solución: Descripción abstracta de la respuesta que soluciona el problema.
- Consecuencias: Beneficios y costos de hacer el uso del patrón.

Los patrones se pueden clasificar en tres categorías como se muestra a continuación:

- Patrones Creacionales: Instanciación de objetos.
- Patrones Estructurales: Separan la interfaz de la implementación.
- Patrones de Comportamiento: Caracterizan las formas en las que interactúan y reparten responsabilidades las distintas clases u objetos.

11.2 Patrones Creacionales

Definición

Los patrones de diseño creacionales proporcionan ayuda a la hora de crear objetos apoyando el proceso de la toma de decisiones, incluso cuando esta toma de decisiones sea de forma dinámica. Además que tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instantiación y ocultar los detalles de cómo los objetos son creados o inicializados.

Tipos de Patrones

Los patrones que se encuentran en la lista de Patrones Creacionales en el enfoque GOF son:

1. Abstract Factory (Fábrica Abstracta)
2. Factory Method (Método Fabrica)
3. Prototype (Prototipo)
4. Builder (Constructor)
5. Singleton (Instancia Única)

11.2.1 Fabrica Abstracta

Modelo

El patrón **Abstract Factory** o **Fábrica Abstracta** resuelve el problema de crear familias de objetos. Permite trabajar con objetos de diferentes familias de manera que no se mezclen entre sí. El patrón Abstract Factory, por tanto, se recomienda cuando se determina la inclusión de nuevas familias de productos en un futuro, pero resultaría contraproducente si que necesita añadir nuevos productos o modificar los existentes, ya que tendría repercusión en todas las familias creadas.

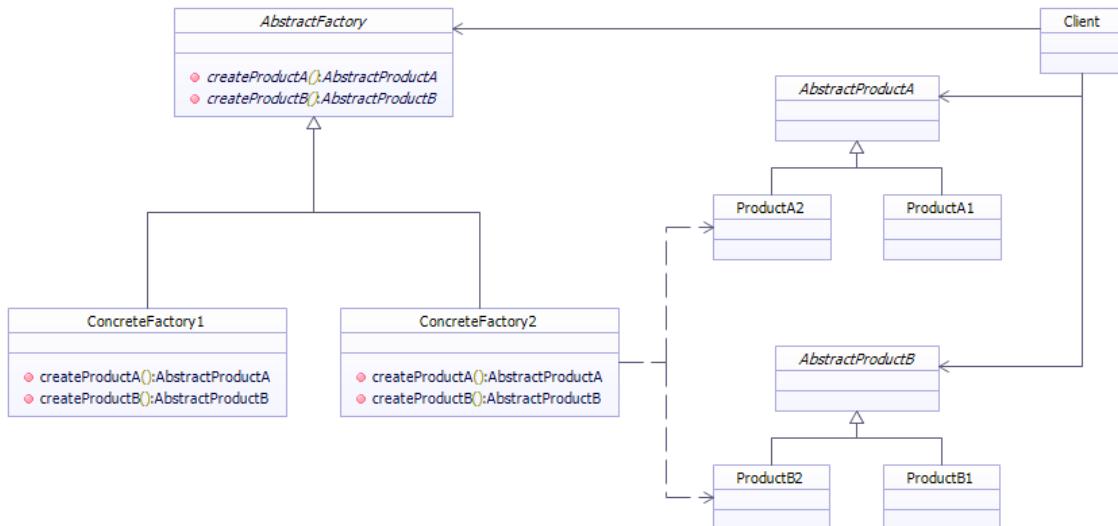


Figura 11.1: Metamodelo Patrón Fábrica Abstracta

Según esto, podemos decir que los componentes típicos del patrón Fábrica Abstracta son los siguientes:

- **Cliente:** Entidad que llamará a la fábrica adecuada para crear uno de los objetos que provee dicha factoría, es decir, intentará obtener una instancia de alguno de los productos que entre en juego.
- **Fábrica Abstracta:** Definición de la interfaz que usarán las diferentes factorías, como mínimo, debe ofrecer un método para la obtención de cada objeto que se pueda crear

- **Fábricas Concretas:** Aquí se representarán las diferentes familias de los productos. Provee la instancia concreta del objeto que se encarga de crear.
- **Producto Abstracto:** Definirá las interfaces para la familia de productos genéricos. El cliente trabajará directamente sobre esta interfaz, que será implementada por los diferentes productos concretos.
- **Producto Concreto:** Se encargará de la implementación específica de los diferentes productos.

Caso de estudio

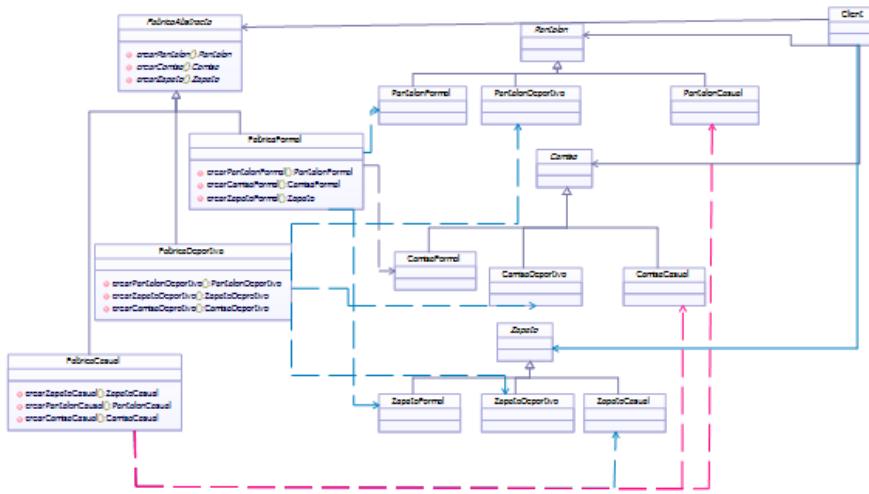


Figura 11.2: Diagrama caso de estudio Patrón Fabrica Abstracta

En el diagrama podemos ver que, para nuestro caso tendremos diferentes líneas de productos que podrán ser registrados en la base de datos. Para esto usamos el patrón fabrica abstracta que nos permite crear familias de productos de acuerdo a la línea establecida, además este patrón permite el escalamiento del aplicativo en forma horizontal, con lo cual, podremos a futuro adicionar otras nuevas líneas o colecciones de productos, sin necesidad de modificar las líneas ya creadas.

11.2.2 Método Fábrica

Modelo

El patrón **Método Fábrica** libera al desarrollador sobre la forma correcta de crear objetos. Define la interfaz de creación de un cierto tipo de objeto, permitiendo que las subclases decidan qué clase concreta necesitan instancias. Muchas veces ocurre que una clase no puede anticipar el tipo de objetos que debe crear, ya que la jerarquía de clases que tiene requiere que deba delegar la responsabilidad a una subclase.

Los elementos de este patrón son:

- **Creador:** Declara el método de fabricación (creación), que devuelve un objeto de tipo Producto.
- **Creador Concreto:** Redefine el método de fabricación para devolver un producto.
- **Producto Concreto:** Es el resultado final. El creador se apoya en sus subclases para definir el método de fabricación que devuelve el objeto apropiado.

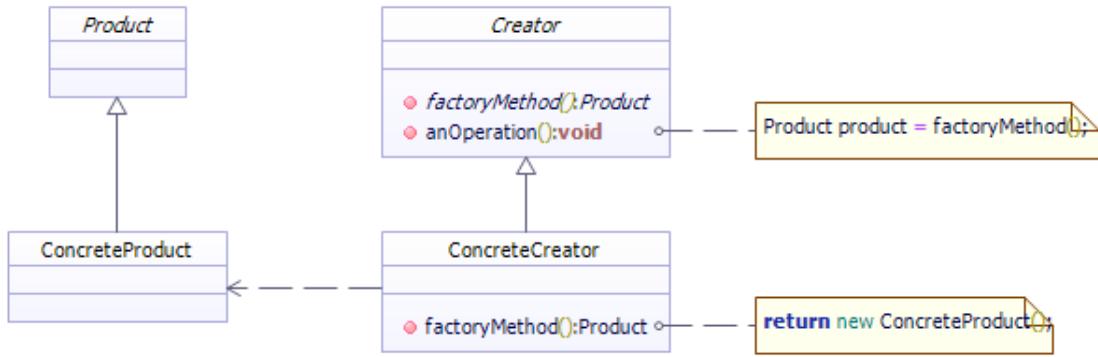


Figura 11.3: Metamodelo Patrón Método Fabrica

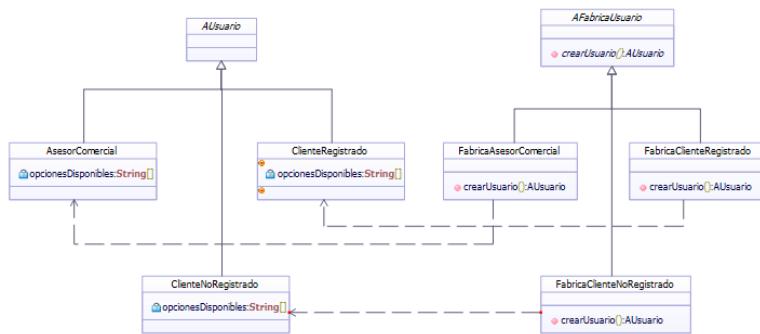


Figura 11.4: Diagrama caso de estudio Patrón Método Fábrica

Caso de estudio

Para el caso de estudio se puede observar que se implementó el método fabrica para facilitar la creación del tipo de usuarios que maneja el sistema, en este caso cada usuario tiene asociadas unas opciones, por ejemplo el usuario no registrado no puede realizar una compra, el usuario registrado sí, entre otros, para crear un usuario para cada caso cuando se necesite y saber las opciones que este tiene asociadas se implementó este patrón.

11.2.3 Singleton

Modelo

La idea del patrón Singleton es proveer un mecanismo para limitar el número de instancias de una clase. Por lo tanto el mismo objeto es siempre compartido por distintas partes del código. Puede ser visto como una solución más elegante para una variable global porque los datos son abstraídos por detrás de la interfaz que publica la clase singleton. Dicho de otra manera, esta patrón busca garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Debe usarse cuando:

1. Debe haber exactamente una instancia de una clase y deba ser accesible a los clientes desde un punto de acceso conocido.
2. Se requiere de un acceso estandarizado y conocido públicamente.

Sus usos más comunes son clases que representan objetos únicos. Por ejemplo, si hay un servidor que necesita ser representado mediante un objeto, este debería ser único, es decir, debería existir una sola instancia y el resto de las clases deberían comunicarse con el mismo servidor.

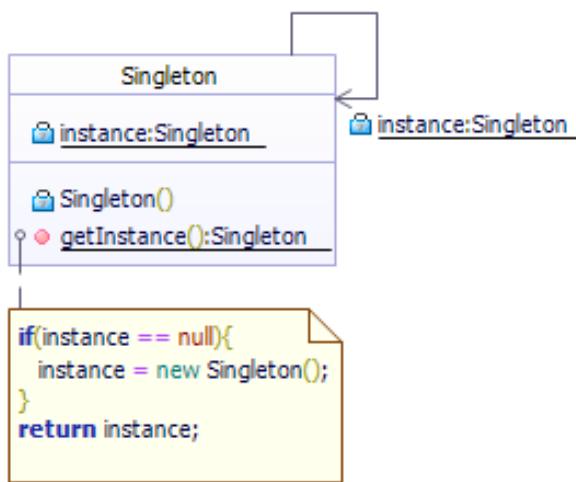


Figura 11.5: Metamodelo patron Singleton

Un Calendario, por ejemplo, también es único para todos.

Este patrón tiene una clase Singleton que define una instancia para los clientes que la acceden. Esta instancia es accedida desde un método llamado `getInstancia` que devuelve la instancia actual de la clase o crea una nueva en el caso de ser nula.

Caso de estudio

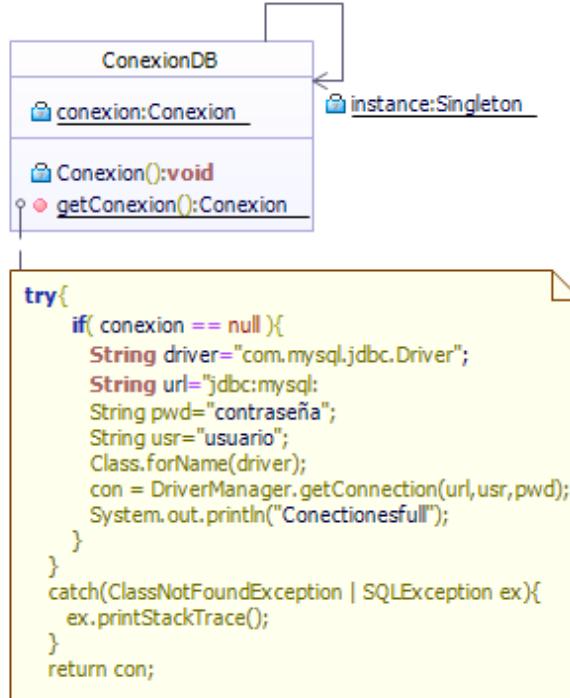


Figura 11.6: Diagrama caso de estudio Patrón Singleton

Para nuestro caso de estudio tenemos una clase ConexionDB que es analoga a la clase singleton, esta tiene un metodo llamado getConexion que se encarga de retornar la instancia ya creada de la clase o en el caso de que aun no exista ninguna instancia crear una nueva.

La idea de aplicar este patrón en la conexión, es para que una vez se haya establecido una conexión a la base de datos, se trabaje sobre la misma sesión y no se creen muchas conexiones dependiendo de lo que se vaya a realizar

11.3 Patrones Estructurales

Definición

Los patrones estructurales se enfocan en como las clases y objetos se componen para formar estructuras mayores, ademas describen como las estructuras compuestas por clases crecen para crear nuevas funcionalidades de manera que se puede agregar a la estructura flexibilidad y que la misma pueda cambiar en tiempo de ejecución lo cual es imposible con una composición de clases estáticas.

Tipos de Patrones

Los patrones que se encuentran en la lista de Patrones Estructurales en el enfoque GOF son:

1. Bridge (Puente)
2. Adapter (Adaptador)
3. Composite (Componente)
4. Decorator (Decorador)
5. Facade (Fachada)
6. Flyweight (Peso Ligero)
7. Proxy (Proxy)

11.3.1 Puente

Modelo

El patrón **Bridge o Puente** desacopla una abstracción de su implementación, de manera que ambas puedan variar de forma independiente. ¿Que quiere decir exactamente esto? Una abstracción se refiere a un comportamiento que una clase debería implementar, ya sea porque esta obligada por una interface o una clase abstracta. Por otro lado, con implementación se refiere a colocarle lógica a dicha obligación. Cuando un objeto tiene unas implementaciones posibles, la manera habitual de implementación es el uso de herencias. Muchas veces la herencia se puede tornar inmanejable y, por otro lado, acopla el código cliente con una implementación concreta.

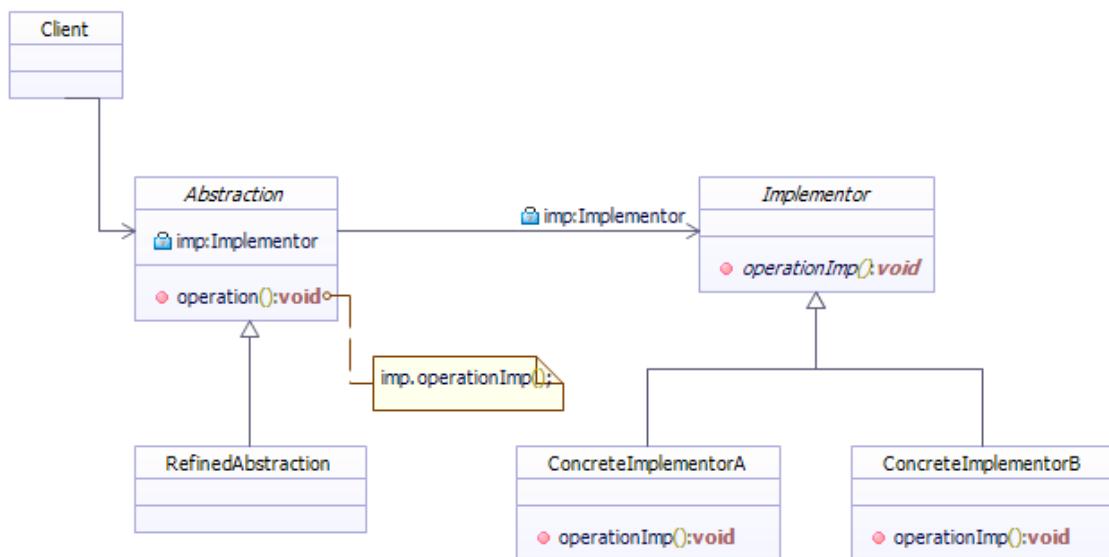


Figura 11.7: Metamodelo Patrón Puente

Este patrón busca eliminar la inconveniencia de esta solución. Las clases manejadas por este patrón son:

- **Abstracción:** Define una interface abstracta. Mantiene una referencia a un objeto de tipo Implementor.
- **Abstracción Refinada:** Extiende la interface definida por Abstracción
- **Implementador:** Define la interface para la implementación de clases. Esta interface no se tiene que corresponder exactamente con la interface de Abstraccion; de hecho, las dos interfaces pueden ser bastante diferentes entre sí. Típicamente la interface Implementador provee sólo operaciones primitivas, y Abstraccion define operaciones de alto nivel basadas en estas primitivas.
- **Implementador Concreto:** Implementa la interface de Implementador y define su implementación concreta.

Caso de estudio

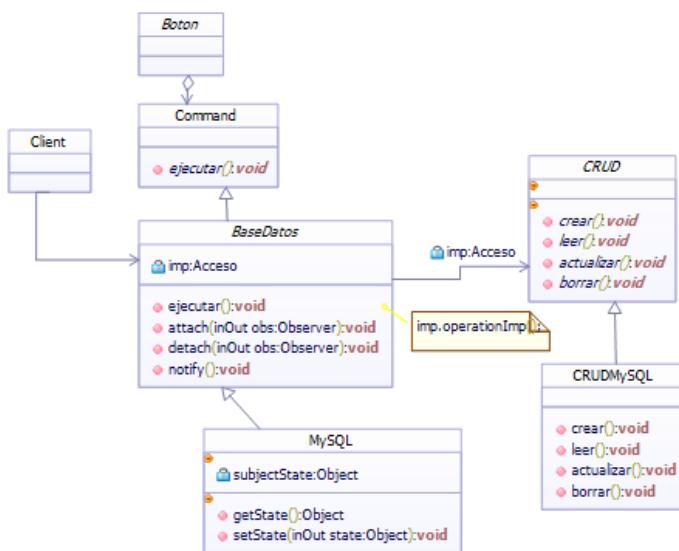


Figura 11.8: Diagrama caso de estudio Patrones Comando y Puente

El patrón comando utilizado en este caso permite separar y simplificar el uso de los botones y la función de cada uno en distintas opciones presentadas al usuario, para este caso particular están definidos los casos en los que el Admin crea, modifica, elimina o agrega un nuevo producto a la base de datos. El patrón bridge, es un puente entre la abstracción de una base de datos y sus funciones con la lógica de estas funciones de acuerdo con el tipo de base de datos, puesto que la manera en que se borra, crea, agrega o modifica es diferente en cada base de datos (MySQL, Postgres, etc), de manera que en la base concreta se define el método en que cada función de la abstracción, realiza su operación. Esto permite que más adelante podamos agregar otras bases de datos a nuestro aplicativo sin la necesidad de modificar el código ya creado, y haciendo escalamiento de nuestro programa de manera horizontal.

11.3.2 Fachada

Modelo

El patrón **Facade o Fachada** busca simplificar el sistema, desde el punto de vista del cliente, proporcionando una interfaz unificada para un conjunto de subsistemas, definiendo una interfaz de nivel más alto. Esto hace que el sistema sea más fácil de usar.

Las clases utilizadas por este patrón son:



Figura 11.9: Metamodelo Patrón Fachada

- **Fachada:** Conoce cuales clases del subsistema son responsables de una petición. Delega las peticiones de los clientes en los objetos del subsistema.
- **Subsistema:** Manejar el trabajo asignado por el objeto Facade. No tienen ningún conocimiento de la Fachada (no guardan referencia de éste).

Caso de estudio

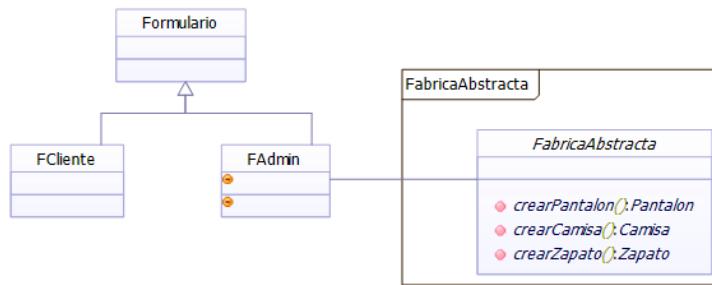


Figura 11.10: Diagrama caso de estudio Patrón Fachada

Este patrón nos permite ocultar al cliente y al admin lo que sucede detrás del formulario que utilizan, es decir oculta la lógica del programa, la cual no es interesante para ellos, podemos incluir varios subsistemas con este patrón y dejarlos solo visibles para los que estén realmente interesados en cada uno.

11.4 Patrones de Comportamiento

Definición

Los patrones de Comportamiento son los patrones que ofrecen soluciones con respecto a la interacción y responsabilidades que hay entre objetos y clases, y las interacciones que hay entre estos además de los algoritmos que estos contiene.

Tipos de Patrones

Los patrones que se encuentran en la lista de Patrones de Comportamiento en el enfoque GOF son:

1. Command (Comando)
2. Chain of Responsibility (Cadena de responsabilidades)
3. Iterator (Iterador)
4. Interpreter (Interprete)
5. Mediator (Mediador)
6. Observer (Observador)
7. State (Estado)
8. Strategy (Estrategia)
9. Template Method (Método Plantilla)
10. Visitor (Visitante)

11.4.1 Comando

Modelo

El patrón **Command** o **Comando** encapsula un mensaje como un objeto. Especifica una forma simple de separar la ejecución de un comando, del entorno que generó dicho comando. Permite solicitar una operación a un objeto sin conocer el contenido ni el receptor real de la misma.

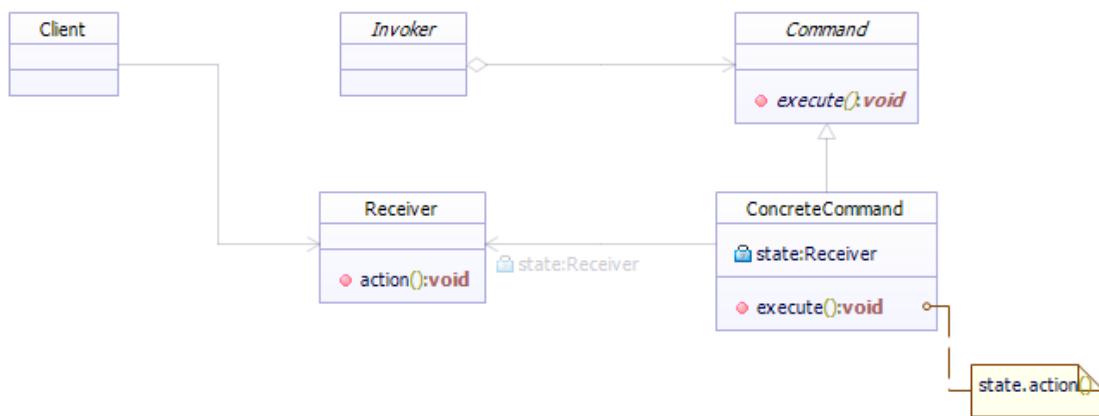


Figura 11.11: Metamodelo Patrón Comando

- **Comando:** Declara una interfaz para ejecutar una operación.
- **Comando Concreto:** Define un enlace entre un objeto “Receptor” y una acción.
- **Cliente:** Crea un objeto “Comando Concreto” y establece su receptor.
- **Invocador:** Le pide a la orden que ejecute la petición.
- **Receptor:** Sabe como llevar a cabo las operaciones asociadas a una petición.

Caso de estudio

El patrón comando utilizado en este caso permite separar y simplificar el uso de los botones y la función de cada uno en distintas opciones presentadas al usuario, para este caso particular están

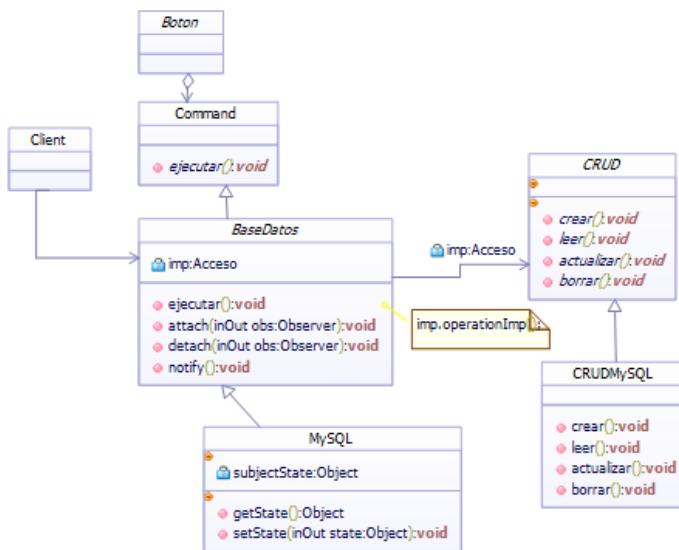


Figura 11.12: Diagrama caso de estudio Patrones Comando y Puente

definidos los casos en los que el Admin crea, modifica, elimina o agrega un nuevo producto a la base de datos. El patrón bridge, es un puente entre la abstracción de una base de datos y sus funciones con la lógica de estas funciones de acuerdo con el tipo de base de datos, puesto que la manera en que se borra, crea, agrega o modifica es diferente en cada base de datos (MySQL, Postgres, etc), de manera que en la base concreta se define el método en que cada función de la abstracción, realiza su operación. Esto permite que más adelante podamos agregar otras bases de datos a nuestro aplicativo sin la necesidad de modificar el código ya creado, y haciendo escalamiento de nuestro programa de manera horizontal.

11.4.2 Iterador

Modelo

El patrón **Iterator o Iterador** provee un mecanismo estándar para acceder secuencialmente a los elementos de una colección; define una interfaz que declara métodos para acceder secuencialmente a los objetos de una colección. Una clase accede a una colección a través de dicha interfaz.

- **Agregado:** Define una interfaz para crear un objeto iterador.
- **Iterador:** Define la interfaz para acceder y recorrer los elementos de un agregado.
- **Iterador Concreto:** Implementa la interfaz del iterador y guarda la posición actual del recorrido en cada momento.
- **Agregado Concreto:** Implementa la interfaz de creación de iteradores devolviendo una instancia del iterador concreto apropiado.
- **Cliente:** Sigue la interfaz Iterator y lo hace siguiendo los métodos otorgados por la interfaz Iterator.

Caso de estudio

En este caso se puede observar en la figura 11.14 que se hace uso del patrón iterador para iterar por cada uno de los elementos del carrito de compras. En este caso nuestro cliente es cada uno de los productos que pertenecen al carrito, el carrito abstracto tiene una lista de productos que son los escogidos por el cliente, cuando el cliente manipula el carrito de compras se le permite agregar o eliminar productos de este, una vez que este ha decidido que desea comprar todos los elementos que tiene en el carrito es necesario para generar la factura y seguir con el proceso de pago saber el total

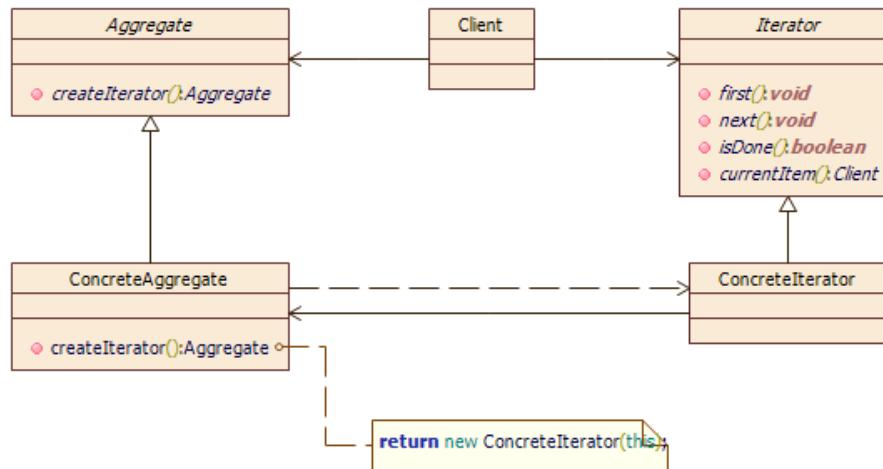


Figura 11.13: Metamodelo Patrón Iterador

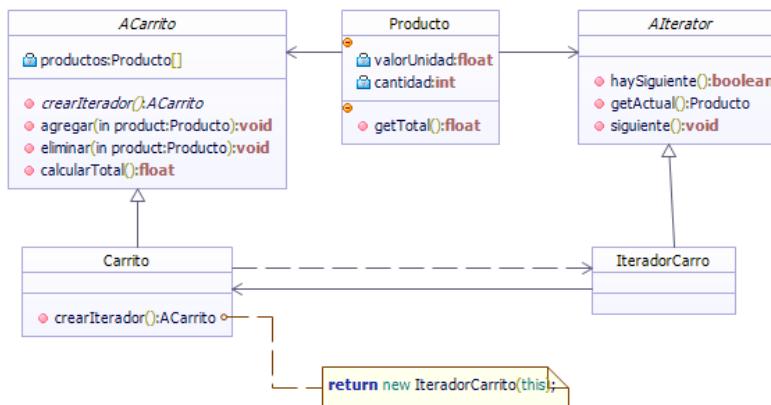


Figura 11.14: Diagrama caso de estudio Patrón Iterador

del valor de los productos que el cliente escogió, para esto se hace uso del **IteradorCarro** el cual recorre el arreglo de productos del carrito preguntando si hay más elementos y obteniendo el valor del producto actual multiplicando el valor de la unidad por la cantidad escogida por el cliente; una vez el iterador define que no hay más elementos siguientes se puede mostrar el valor del total al cliente y continuar con el proceso de pago.

11.4.3 Mediador

Modelo

El patrón **Mediator** o **Mediador** define un objeto que hace de procesador central, y que se encarga coordinar las relaciones entre sus asociados o participantes. Este patrón permite la interacción de varios objetos, sin generar acoplos fuertes en las relaciones. Todos los objetos se comunican con un mediador y es éste quién realiza la comunicación con el resto.

Donde para este patrón se define que:

- **Mediador:** Define una interface para comunicarse con los objetos colegas.
- **Mediador Concreto:** Implementa la interface y define como los colegas se comunican entre

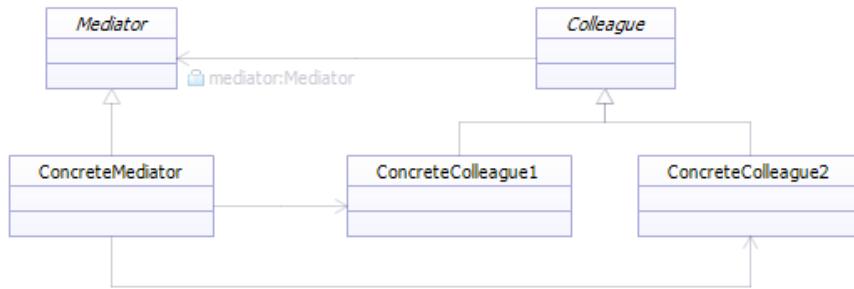


Figura 11.15: Metamodelo Patrón Mediador

ellos.

- **Colega:** Define el comportamiento que debe implementar cada colega para poder comunicarse el mediador de una manera estandarizada para todos.
- **Colega Concreto:** Cada colega conoce su mediador, y lo usa para comunicarse con otros colegas.

Caso de estudio

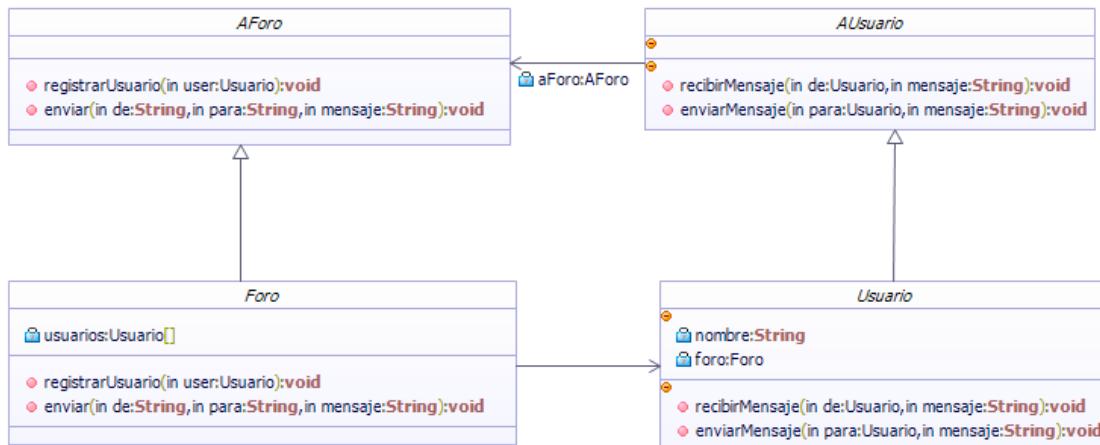


Figura 11.16: Diagrama caso de estudio Patrón Mediador

Para este caso en específico se hace uso de patrón observador en su ejemplo más común no por simplicidad sino porque haciendo uso de este se satisface perfectamente uno de los principales requerimientos del sistema el cual es brindar un mecanismo para que a pesar de ser una tienda virtual los usuarios puedan seguir comunicándose con los asesores.

En este caso se plantea una interfaz abstracta para cada usuario que tiene dos métodos, enviar y recibir mensaje, esta se implementa en cada usuario que a su vez tiene un nombre que lo identifica en el foro y le permitirá distinguirse de los demás y además conoce el foro. La interfaz abstracta **AForo** tiene dos métodos que permiten a los usuarios suscribirse al foro y el método enviar que es donde se realizará todo el proceso de registro y envío del mensaje, esta interfaz la implementa la Clase **Foro** que además tiene un arreglo de usuarios que son los usuarios que pertenecen al chat.

11.4.4 Momento

Modelo

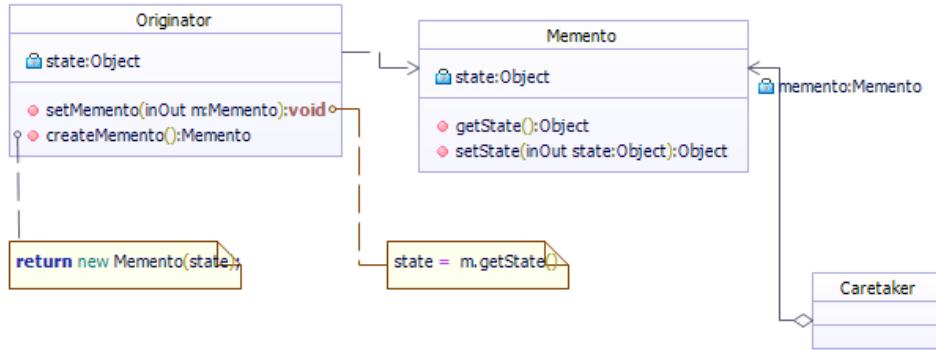


Figura 11.17: Metamodelo patron memento

Este patrón de diseño permite capturar y exportar el estado interno de un objeto para que luego se pueda restaurar, sin romper la encapsulación. Su finalidad es almacenar el estado de un objeto (o del sistema completo) en un momento dado, de manera que se pueda restaurar posteriormente si fuese necesario. Para ello se mantiene almacenado el estado del objeto para un instante de tiempo en una clase independiente de aquella a la que pertenece el objeto (pero sin romper la encapsulación), de forma que ese recuerdo permita que el objeto sea modificado y pueda volver a su estado anterior. Se usa cuando:

1. Se necesite restaurar el sistema desde estados pasados.
2. Se quiera facilitar el hacer y deshacer de determinadas operaciones, para lo que habrá que guardar los estados anteriores de los objetos sobre los que se opere (o bien recordar los cambios de forma incremental).

Este patrón debe ser utilizado cuando se necesite salvar el estado de un objeto y tener disponible los distintos estados históricos que se necesiten.

Los componentes de este patrón son:

1. **Caretaker**: es responsable por mantener a salvo a **Memento**. No opera o examina su contenido.
2. **Memento**: almacena el estado interno de un objeto **Originator**. El **Memento** puede almacenar todo o parte del estado
3. **Originator**: crea un objeto **Memento** conteniendo una fotografía de su estado interno

Caso de estudio

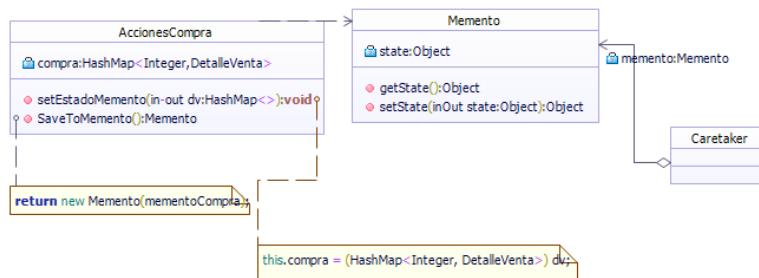


Figura 11.18: Diagrama caso de estudio Patrón Memento

Para nuestro caso de estudio el patrón memento tiene las siguientes clases, una clase AccionesCompra, que se encarga de manejar las acciones relacionadas al carrito de compra, como agregar productos, eliminarlos, confirmar compra entre otras, cada vez que se visualizan los productos agregados al carrito, se genera un detalle de la venta, que tiene los productos y la cantidad que se va a comprar, es decir el estado del carrito, la clase memento tiene un estado que es el estado interno de la clase AccionesCompra en momento dado, y la clase CareTaker es la encarga de almacenar cada uno de los estados internos de la clase AccionesCompra y retornarlos cuando sea necesario. El objetivo del uso de este patron en estas clases, es que una vez se esten visulizando los productos agregados al carrito de compras, y se eliminan por error se pueda deshacer esta accion y retornar al estado anterior del carrito, asi mismo tambien se podra rehacer la accion devolviendo el carrito a su ultimo estado.

11.4.5 Observador

Modelo

El patrón **Observer o Observador** es usado en programación para monitorear el estado de un objeto en un programa. La motivación principal de este patrón es su utilización como un sistema de detección de eventos en tiempo de ejecución.

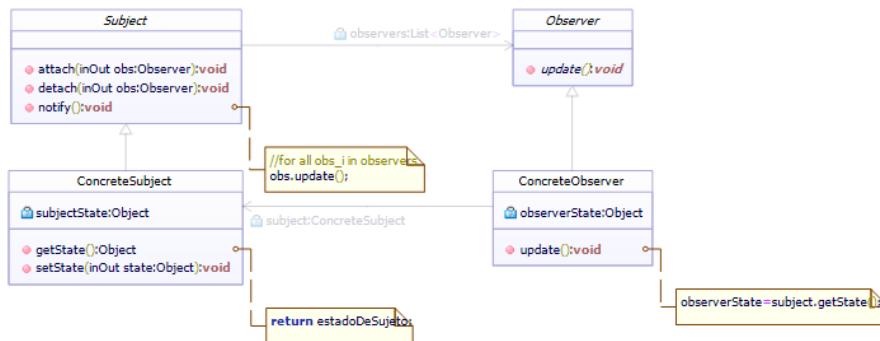


Figura 11.19: Metamodelo del Patrón Observador

Las clases definidas para este patrón son:

- **Tema:** Conoce a sus observadores y ofrece la posibilidad de añadir y eliminar observadores.
- **Observador:** Define la interfaz que sirve para notificar a los observadores los cambios realizados en el Tema.
- **Tema Concreto:** Almacena el estado que es objeto de interés de los observadores y envía un mensaje a sus observadores cuando su estado cambia.
- **Observador Concreto:** Mantiene una referencia a un SubjectConcreto. Almacena el estado del Tema que le resulta de interés.

Caso de estudio

El patrón observador es utilizado en nuestro caso debido a la necesidad que tenemos de estar actualizando constantemente los datos de los productos en los formularios del Admin y el Cliente, ya sea actualizando el estado del producto en el stack, o alguna de sus características cada vez que entramos a modificarlas, este patrón nos permite facilitar esta actualización directamente en nuestra base de datos.

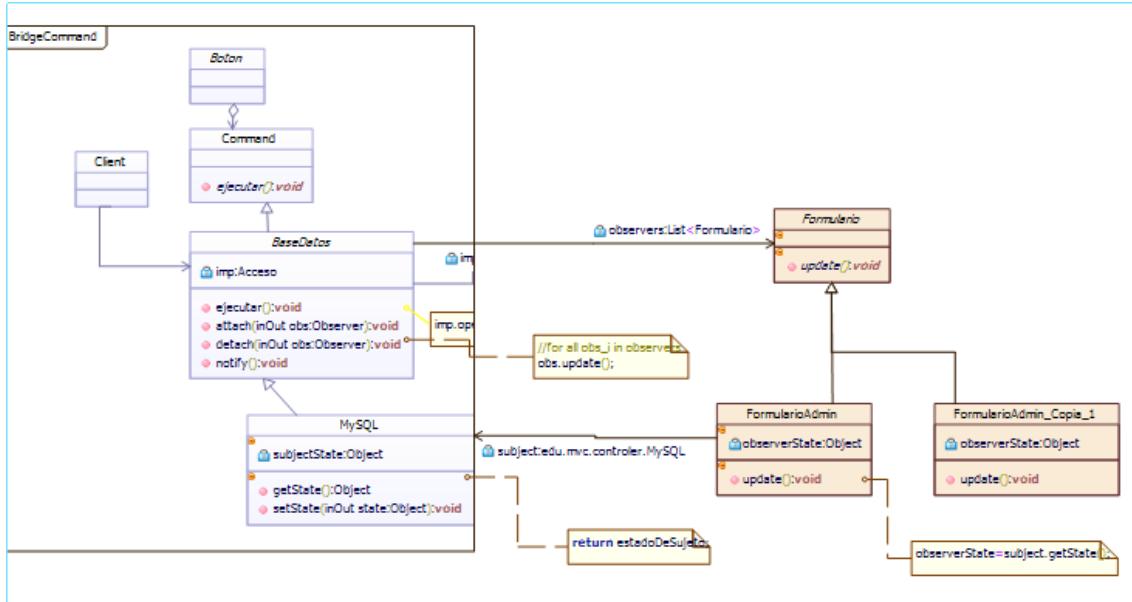


Figura 11.20: Diagrama caso de estudio Patrón Observador

11.4.6 Estado Modelo

El patrón State o estado permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. La intención del State es desacoplar el estado de la clase en cuestión. Las clases utilizadas son:

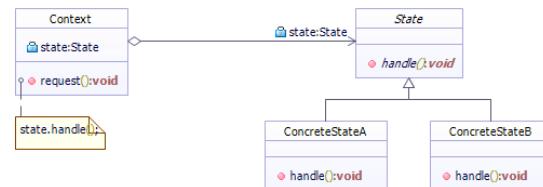


Figura 11.21: metamodelo patrón estado

1. **Contexto:** Mantiene una instancia con el estado actual.
2. **Estado:** Mantiene una instancia con el estado actual.
3. **EstadoConcreto:** Cada subclase implementa el comportamiento asociado con un estado del contexto.

Caso de estudio

En el anterior Diagrama de estados podemos ver la transición que tienen los productos desde el momento en que se cargan a la página, hasta que se realiza exitosamente una compra, para esto nos basamos en 4 estados posibles, en inventario, en carrito, vendido, agotado, el producto empieza en un estado base de en inventario con una cantidad disponible, cuando se agrega el producto a carrito

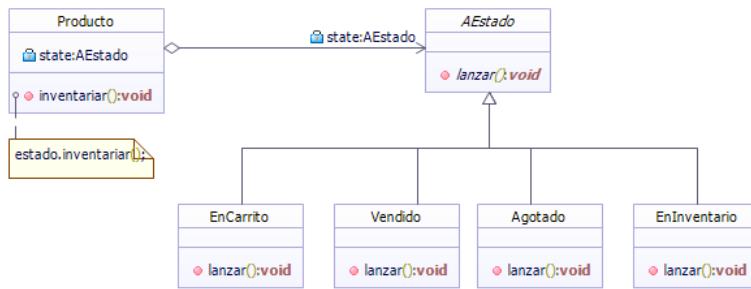


Figura 11.22: Diagrama caso de estudio Patrón Estado

se cambia de estado a en carrito, en donde tiene dos posibles disparadores, que son desagregar de carrito y confirmar compra, si se desagrega del carrito, regresara a su estado base en inventario, si se confirma la compra cambiara de estado ha vendido, en donde se actualizara el inventario, si la cantidad comprada es igual a la disponibles en inventario, el producto quedara en un estado de agotado, si aún hay unidades disponibles volverá a un estado de inventario, a partir de ahí podemos crear una clase producto la cual podrá contener cualquiera de los 4 estados

11.4.7 Estrategia Modelo

El patrón **Strategy** o **Estrategia** encapsula algoritmos en clases, permitiendo que éstos sean re-utilizados e intercambiables. En base a un parámetro, que puede ser cualquier objeto, permite a una aplicación decidir en tiempo de ejecución el algoritmo que debe ejecutar.



Figura 11.23: Metamodelo del Patrón Estrategia

Las clases que usa este patrón son:

- **Estrategia:** Declara una interfaz común a todos los algoritmos soportados.
- **Estrategia Concreta:** Implementa un algoritmo utilizando la interfaz Estrategia. Es la representación de un algoritmo.
- **Contexto:** Mantiene una referencia a Estrategia y según las características del contexto, optará por una estrategia determinada.
- **Cliente:** Sigue un servicio a Estrategia y este debe devolver el resultado de la Estrategia Concreta.

Caso de estudio

Como se puede observar en la figura 11.24 en este caso se aplica la estrategia para solicitar al cliente el recaudo del pago, debido a que se tienen diferentes formas o posibilidades de pago y cada pago se hace de forma diferente, en este caso la factura solicita recaudar el pago y conforme al método escogido por el cliente se ejecutará la estrategia correspondiente.

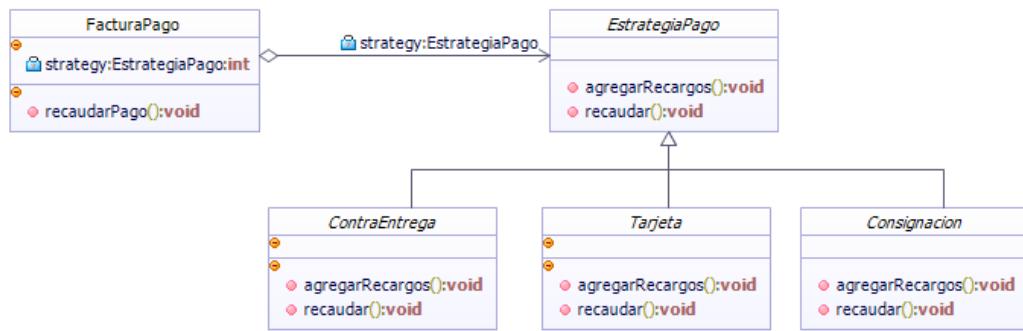
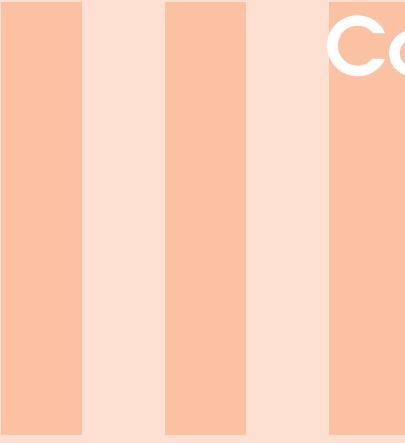


Figura 11.24: Diagrama caso de estudio Patrón Estrategia

Debido a que cada medio de pago requiere agregar recargos diferentes, como es el caso de la forma de pago de consignación y de tarjeta que requieren agregar los impuestos correspondientes, o el caso del pago a contra entrega en donde es necesario agregar el recargo del mensajero que se encargará de entregar el producto al cliente y cobrar el valor correspondiente al total.

Al igual que los recargos que se agregan son diferentes, también son diferentes las formas de recaudar el pago para cada modalidad, en el caso de la contra entrega el usuario es redirigido a confirmar los datos de envío, el método de pago con tarjeta requiere comunicarse con la plataforma de pagos y la consignación redirigir al banco correspondiente.

El uso de este patrón nos permite que a futuro sea posible agregar nuevas formas de pago sin tener que modificar el código y la implementación ya existente.



Conclusiones y Reflexiones



12. Conclusiones

Usar Archimate como lenguaje de modelado para todo el diseño y desarrollo del proyecto permitió comprender y analizar con cada punto de vista todos y cada uno de los aspectos que tiene el proyecto y que dejó como resultado una lista de diagramas que son de fácil lectura y comprensión para cualquier persona que conozca el lenguaje o que tenga solo las nociones principales.

Cada una de las cinco capas de Archimate nos permitió modelar el sistema desde diferentes puntos de vista y no solo desde el punto de vista del desarrollador, permitiendo así tener en cuenta aspectos que son esenciales para el negocio y que un desarrollador podría omitir por diferentes causales. Se observó para cada capa lo siguiente:

- **Capa de Negocio o Empresarial:** La capa de negocio permite modelar todo el negocio desde los actores y ubicaciones con las que se cuentan hasta los productos y servicios que se brindan.
- **Capa de Aplicación:** La capa de aplicación es compatible con la capa de negocio pero esta vez implementados en el software.
- **Capa de Tecnología o Infraestructura:** La capa de tecnología nos permitió modelar el sistema desde un punto de vista de lo necesario para la implementación del mismo ayudando así a entender todos los aspectos involucrados y necesarios para llevar a cabo este proceso.
- **Capa de Motivación:** La capa de Motivación nos permitió modelar las razones por las que se está planteando el desarrollo del proyecto.
- **Capa de Proyecto:** La capa de proyecto nos permitió modelar el desarrollo actual del producto y las arquitecturas futuras que se desean como parte todo de un solo proceso que se lleva a cabo en una secuencia.

Durante el desarrollo del análisis y durante la implementación del proyecto se trabajó con un enfoque de componentes, usar este enfoque trae consigo varias ventajas como que no solo se logra que el sistema tenga un grado bajo de acoplamiento sino que también se permite la extensión de este fácilmente debido a que no solo se ordena el sistema y se hace más fácil de entender sino que también agregar componentes es más fácil que agregar códigos. Además de la extensión se tiene que destacar que se facilita el mantenimiento del software.

Al usar los patrones de diseño no solo se ve que los diseños y la implementación se hacen más fáciles y entendibles sino que también se observa que a futuro hacer usos de estos permitirá extender las funcionalidades del sistema de una forma mucho más fácil y que no será necesario que la misma persona lo haga debido a que se sigue un estandar que es entendible para todos.



13. Trabajos Futuros

La siguiente versión de este proyecto estará enfocada a convertir a JSJSports web en una plataforma e-commerce general que no solo se aplique a la venta de artículos deportivos sino que en esta se pueda ofrecer cualquier tipo de artículo y se cuente con un vendedor que brinde soporte al cliente durante el proceso de compra, para esto es necesario aumentar el número de módulos y de productos disponibles, es decir, diversificar la página.

Pero esta diversificación no solo se logrará agregando módulos y productos sino que también se hace necesario que se diversifiquen las formas de pago que se tienen actualmente y por qué no se permita una interacción más cercana con el cliente mejorando los canales de comunicación e implementando nuevos métodos como las videollamadas.

Al igual que se explica en el punto de vista de Migración de la capa de proyecto analizada en el presente trabajo como arquitectura final del e-commerce se plantea una plataforma que proporcione un servicio para diferentes páginas, es decir que no sea necesario que el cliente se encuentre en el dominio de la página sino que este pueda consultar productos por medio de otras o de terceros al igual que para que para que el cliente vea la publicidad no necesariamente tenga que estar en el dominio de la página.



Bibliografía

- [1] Patrones gof.
- [2] Sandro BOLAÑOS. *Punto de Vista de Comportamiento de Aplicación*.
- [3] Sandro BOLAÑOS. *Punto de Vista de Cooperación de Actor*.
- [4] Sandro BOLAÑOS. *Punto de Vista de Cooperación de Aplicación*.
- [5] Sandro BOLAÑOS. *Punto de Vista de Cooperación de Proceso de Negocio*.
- [6] Sandro BOLAÑOS. *Punto de Vista de Estructura de Aplicación*.
- [7] Sandro BOLAÑOS. *Punto de Vista de Función de Negocio*.
- [8] Sandro BOLAÑOS. *Punto de Vista de Organización*.
- [9] Sandro BOLAÑOS. *Punto de Vista de Proceso de Negocio*.
- [10] Sandro BOLAÑOS. *Punto de Vista de Producto*.
- [11] Sandro BOLAÑOS. *Punto de Vista de Uso de Aplicación*.
- [12] The Open Group. *ArchiMate 2.0 Specification*.
- [13] The Open Group. *ArchiMate 3.0.1 Specification*.
- [14] The Open Group. *TOGAF 9.1. Part II: Architecture Development Method (ADM)*.
- [15] Etienne Venter. *Integrating TOGAF ADM, ArchiMate, BPMN and UML into your SDLC*.



Anexos

13.1 Códigos

13.1.1 Códigos de las clases del Patrón Estado

Estados

Estado Abstracto

```
1 package productoestados;
2
3
4 public abstract class AEstado {
5     private Producto producto;
6
7     public abstract void lanzar(Producto producto);
8     public abstract void setCompra(boolean compra);
9
10 }
```

Estados Concretos

```
1 package productoestados;
2
3
4
5 public class Agotado extends AEstado {
6
7     @Override
8     public void lanzar(Producto producto) {
9         System.out.println("el producto se encuentra agotado");
10    }
11
12     @Override
```

```
13     public void setCompra(boolean compra) {
14
15     }
16 }
```



```
1
2 package productoestados;
3
4
5 public class EnCarrito extends AEstado {
6
7     private boolean compra;
8
9     @Override
10    public void lanzar(Producto producto) {
11        if(producto.getEstado().equals(this) && compra==true){
12            System.out.println("el producto ha sido comprado");
13        }else if(producto.getEstado().equals(this) && compra==false
14            ){
15
16                System.out.println("la compra ha sido cancelada, el
17                    producto ha sido retirado del carrito");
18            }else{
19                System.out.println("no se ha podido efectuar la
20                    operacion");
21            }
22        }
23
24        public void lanzar(Producto producto , boolean compra){
25            this.compra=compra;
26            lanzar(producto);
27        }
28
29
30     @Override
31    public void setCompra(boolean compra) {
32        this.compra = compra;
33    }
34 }
```



```
1
2 package productoestados;
3
4
5 public class EnInventario extends AEstado {
6
7     @Override
8    public void lanzar(Producto producto) {
9        if (producto.getEstado().equals(this) && producto.
10           getInventario() >= producto.getSeleccionado()
11               && producto.getInventario() > 1 && producto.
```

```

11         getSeleccionado()>0) {
12             System.out.println("se han agregado "+producto.
13                 getSeleccionado()+" unidades del producto al carrito
14                 ");
15             }else if(producto.getEstado().equals(this) && producto.
16                 getSeleccionado()==0){
17                 System.out.println("ahi "+producto.getInventario()+"
18                     unidades del producto en inventario");
19             }else{
20                 System.out.println("No se puede ejecutar la operacion"
21                     );
22             }
23         }

```

```

1 package productoestados;
2
3 public class Vendido extends AEstado {
4
5     @Override
6     public void lanzar(Producto producto) {
7         producto.setInventario(producto.getInventario()-producto.
8             getSeleccionado());
9         if (producto.getInventario() > 1) {
10
11             System.out.println(producto.getNombre()+" inventario
12                 disponible :"+ producto.getInventario());
13             producto.inventariar();
14         } else {
15
16             System.out.println("producto agotado");
17             producto.agotar();
18         }
19
20     @Override
21     public void setCompra(boolean compra) {
22
23     }
24
25 }

```

Producto

```

1
2 package productoestados;
3
4

```

```
5  public class Producto {  
6  
7      private AEstado estado;  
8      private int inventario;  
9      private int seleccionado;  
10     private String nombre;  
11  
12     public Producto(int inventario, String nombre){  
13         this.inventario=inventario;  
14         this.seleccionado=0;  
15         this.nombre=nombre;  
16     }  
17  
18     public AEstado getEstado() {  
19         if(estado==null){  
20             estado=new EnInventario();  
21         }  
22         return estado;  
23     }  
24  
25     public void setEstado(AEstado estado) {  
26         this.estado = estado;  
27     }  
28  
29  
30  
31  
32     public int getInventario() {  
33  
34         return inventario;  
35     }  
36  
37     public void setInventario(int inventario) {  
38         this.inventario = inventario;  
39     }  
40  
41     public int getSeleccionado() {  
42         return seleccionado;  
43     }  
44  
45     public void setSeleccionado(int seleccionado) {  
46         this.seleccionado = seleccionado;  
47     }  
48  
49     public String getNombre() {  
50         return nombre;  
51     }  
52  
53     public void setNombre(String nombre) {  
54         this.nombre = nombre;  
55     }  
56  
57     public void agregarCarrito(int cantidad){  
58         seleccionado=cantidad;  
59         estado.lanzar(this);  
60         estado = new EnCarrito();
```

```

61     }
62
63     public void desagregarCarrito(){
64         estado.setCompra(false);
65         estado.lanzar(this);
66         estado = new EnInventario();
67     }
68
69     public void confirmarCompra(){
70         estado.setCompra(true);
71         estado.lanzar(this);
72         estado = new Vendido();
73         estado.lanzar(this);
74     }
75
76     public void agotar(){
77         seleccionado = inventario;
78         estado = new Agotado();
79     }
80
81     public void inventariar(){
82         seleccionado = 0;
83         estado = new EnInventario();
84     }
85
86     public void lanzar(){
87         getEstado().lanzar(this);
88     }
89
90
91 }
```

13.1.2 Códigos de las clases del Patrón Singleton

```

1
2 package Persistencia;
3
4 import java.sql.Connection;
5 import java.sql.DriverManager;
6 import java.util.logging.Level;
7 import java.util.logging.Logger;
8
9 public class ConexionDB {
10
11     private static Connection conexion = null;
12     private ConexionDB(){}
13
14     public static Connection getConexion() {
15         if (ConexionDB.conexion == null) {
16             try {
17                 Class.forName("com.mysql.jdbc.Driver");
18                 conexion = DriverManager.getConnection("jdbc:mysql
19                     ://localhost/basedatosjsj", "root", "");
20                 System.out.println("Conexion realizada");
21             } catch (Exception e) {
22                 e.printStackTrace();
23             }
24         }
25     }
26 }
```

```

20         return conexion;
21     } catch (Exception e) {
22         Logger.getLogger(ConexionDB.class.getName()).log(
23             Level.SEVERE, null, e);
24     }
25 } else {
26     return conexion;
27 }
28 return conexion;
29 }
30 }

```

13.1.3 Clase Cargador del paquete Utilidades

```

1 package com.logica;
2
3 import java.io.File;
4 import java.net.URL;
5 import java.net.URLClassLoader;
6 import java.util.HashMap;
7 import java.util.jar.Attributes;
8 import java.util.jar.JarFile;
9
10 public class Cargador {
11     public static String COMPONENTE = "COMPONENTE";
12     URLClassLoader cargador;
13     HashMap<String, String> mapa=new HashMap<String, String>();
14
15     public Cargador(String dir, ClassLoader cl) {
16         File archivos[]=new File(dir).listFiles();
17         URL [] urls = new URL[archivos.length];
18         for(int i=0;i<archivos.length;i++){
19             try {
20                 urls[i] = archivos[i].toURI().toURL
21                     ();
22                 Attributes atr=new JarFile(
23                     archivos[i]).getManifest().
24                     getAttributes(COMPONENTE);
25                 if(atr!=null){
26                     String nombre=atr.getValue(""
27                         "nombre");
28                     String calse=atr.getValue("clase
29                         ");
30                     mapa.put(nombre, calse);
31                 }
32             } catch (Exception e) {
33                 e.printStackTrace();
34             }
35         }
36         cargador = new URLClassLoader(urls);
37     }
38 }

```

```

34     public Class<?> cargarConElComponente(String
35         nombreDelComponente){
36             try {
37                 return cargador.loadClass((String)mapa.get(
38                     nombreDelComponente));
39             } catch (ClassNotFoundException e) {
40                 e.printStackTrace();
41             }
42             return null;
43         }

```

13.1.4 Clase Cableado del componente central

```

1 package com.cableado;
2
3 import com.logica.Cargador;
4 import com.logica.IAsesoriaCliente;
5 import com.logica.IDB;
6 import com.logica.IGestionCarrito;
7 import com.logica.IGestionInventario;
8 import com.logica.IGestionUsuarios;
9 import com.logica.IPagos;
10 import com.logica.ISesionCliente;
11
12
13 public class JSJSport {
14     public static void main(String[] args) {
15         Cargador cBack = new Cargador("backOffice",
16             ClassLoader.getSystemClassLoader());
17         Cargador cFront = new Cargador("frontOffice",
18             ClassLoader.getSystemClassLoader());
19         try {
20             IPagos pagos;
21             IGestionCarrito carrito;
22             IGestionInventario inventario;
23             IAsesoriaCliente asesor;
24             IGestionUsuarios usuarios;
25             ISesionCliente sesion;
26             IDB db;
27             db = (IDB) cBack.cargarConElComponente(IDB.
28                 class.getSimpleName()).newInstance();
29             db.connect(" ", " ", " ");
30             sesion = (ISesionCliente) cFront.
31                 cargarConElComponente(ISesionCliente.
32                     class.getSimpleName()).newInstance();
33             sesion.login(" ", " ");
34             usuarios = (IGestionUsuarios) cBack.
35                 cargarConElComponente(IGestionUsuarios.
36                     class.getSimpleName()).newInstance();
37             usuarios.listarUsuarios();
38             asesor = (IAsesoriaCliente) cFront.
39                 cargarConElComponente(IAsesoriaCliente.

```

```
32         class.getSimpleName()).newInstance();
33         asesor.mandarMensaje("Se envia el mensaje",
34             " ", " ");
35         inventario= (IGestionInventario) cBack.
36             cargarConElComponente(IGestionInventario
37                 .class.getSimpleName()).newInstance();
38         inventario.agregarProducto("");
39         carrito = (IGestionCarrito) cFront.
40             cargarConElComponente(IGestionCarrito.
41                 class.getSimpleName()).newInstance();
42         carrito.eliminarCarrito(" ", " ");
43         pagos = (IPagos)cBack.cargarConElComponente
44             (IPagos.class.getSimpleName()).
45             newInstance();
46         pagos.verificarPago("Interfaz de pagos");
47     } catch (Exception e) {e.printStackTrace();}
48 }
```

13.2 Desarrollo en Eclipse

13.2.1 Estructura Proyecto

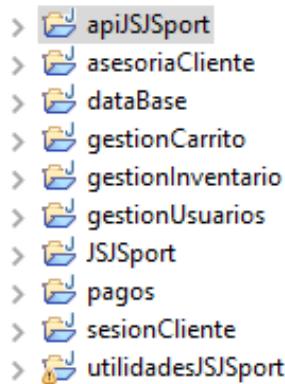


Figura 13.1: Estructura del proyecto en Eclipse

13.2.2 Estructura del API

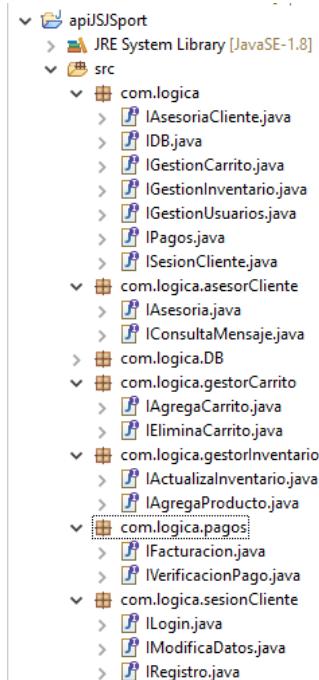


Figura 13.2: Estructura del API e interfaces

13.3 Prototipo pagina web

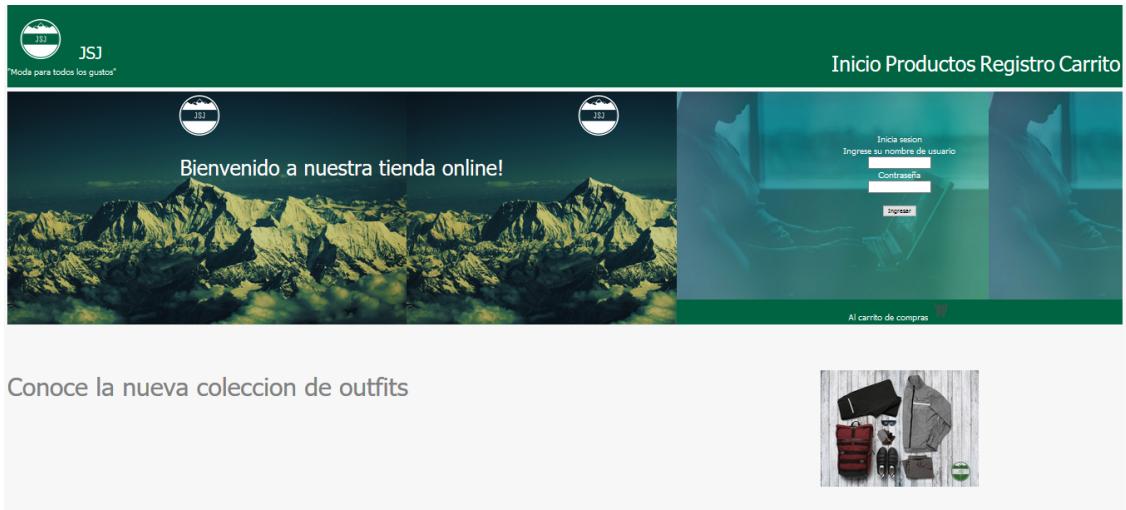


Figura 13.3: Prototipo de la página web Inicio



Figura 13.4: Prototipo de la página web Opiniones



Figura 13.5: JSJSports web Sección Productos



Figura 13.6: JSJSports web Sección Registro de Usuario

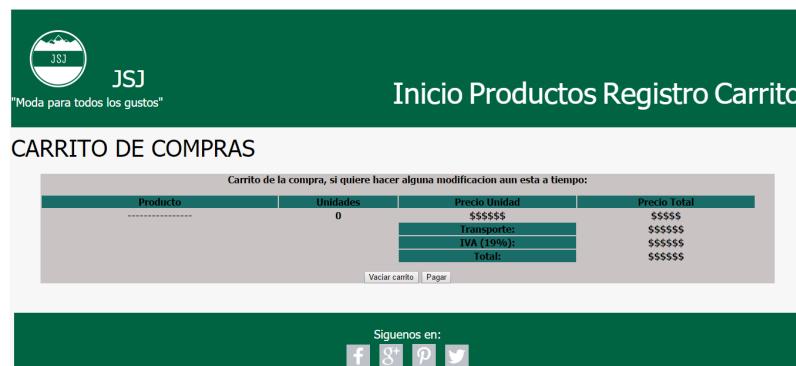


Figura 13.7: JSJSports web Sección Carrito