Cine+

Andrés Leonardo Arias Uribe
Andrés Felipe Rentería Velandia

Índice general

I Proyecto

1. Caso de Estudio

1.1 Introducción

En los últimos cincuenta años se ha desarrollado una revolución tecnologica de una manera tan marcada que ha obligado al mundo a adaptarse y evolucionar a un ritmo vetigínoso; este cambio es algo que afecta a todos los miembros activos de la sociedad, en especial a las empresas, pues han visto como su funcionamiento es cada vez más complejo, puesto que cada vez están en contacto con un número mayor de clientela, lo cual las obliga a buscar ser más productivas e implementan una mayor cantidad de personal. Por todo esto, en los últimos años las empresas han empezado a buscar soluciones tecnológicas que permitan simplificar los procesos que se desarrollan dentro de ellas y lograr tal objetivo de productividad.

1.2 Descripción del problema

Una empresa de cines, ubicada en Bogotá, busca desarrollar un software que le permita administrar su funcionamiento. Dicho software debe estar en la capacidad de gestionar la boletería de todos los cinemas que tiene a lo largo de la ciudad para cada una de las diversas funciones ofrecidas en las distintas horas del día en tareas como compra y reserva de boletas, al igual que las ventas de confiteria de cada cinema. Además estará en la capacidad de llevar un registro y monitoreo de los empleados y el manejo de insumos y maquinaria de confitería que estén a disposición de cada cinema.

Durante el siguiente libro se realizará un estudio del problema planteado, buscando encontrar una solución eficiente mediante la implementación de la ingeniería de software.

1.3 Objetivo General

Generar una solución de software completa que permita administrar y monitorear las actividades principales de una cadena de cines, mediante el uso de metodologías y fundamentos básicos de ingeniería de software.

1.3.1 Objetivos específicos

1. Gestionar el manejo de boletería y funciones para cada uno de los cines a lo largo de la ciudad de Bogotá D.C. incluyendo tanto la compra y reserva de boletas como selección de asientos de las funciones.
2. Administar el manejo de los insumos en la sección de confitería para cada uno de los cinemas de la ciudad, mediante el uso de bases de datos.
3. Desarrollar un cliente de uso para el público general y uno diferente de uso para personal administrativo.
4. Implementar un registro de los empleados que tiene la empresa en cada uno de los cinemas.

1.4 Alcances

Se implementará un aplicativo de uso para público general que podrá realizar las siguientes actividades:

- Compra y reserva de boletas.
- Selección de asientos.
- Visualización de peliculas en cartelera y próximos estrenos.
- Gestión de suscripción especial.
- Exposición y compra de confiteria.
- Generación de tickets de pago.

Se implementará un aplicativo de uso para personal administrativo que podrá realizar las siguientes actividades:

- Gestionamiento de cartelera, próximos estrenos y confiteria.
- Asignación de funciones.
- Manejo del inventario de confiteria.
- Gestionamiento de personal a nivel de registro.

1.5 Limitaciones

- El cliente público se limitará a la emisión de tickets de pago, no contará con la posibilidad de un módulo de pago.
- El cliente de personal administrativo no contará con el manejo de nómina, solamente se restringirá al registro de personal de trabajo.

2. Proceso

Para la realización de este proyecto, se utilizará el proceso de software 'Prototipo'; proceso que permite generar iteraciones en las que se genera un prototipo en cada una de ellas,y que posteriormente será expuesto a pruebas y labores de mantenimiento para iniciar la siguiente iteración. Todo esto con la finalidad de obtener un producto final que cumpla con todos los requerimientos planteados para el buen funcionamiento.

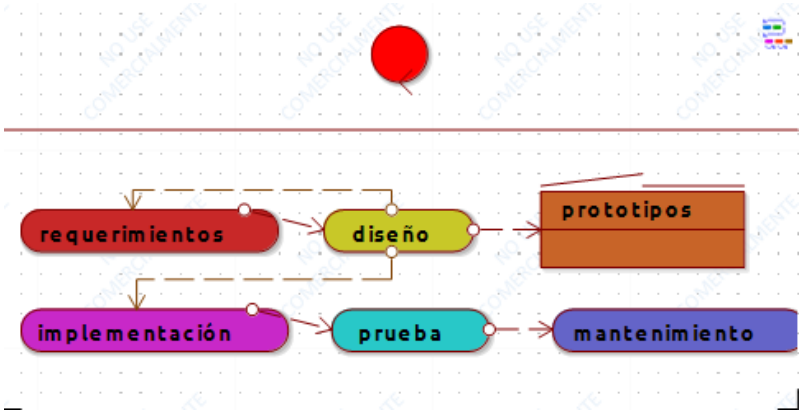

Figura 2.1: Proceso Prototipo

2.1 Implementación

El proyecto será estructurado como un conjunto de módulos o componentes, los cuales serán organizados en una escala de dependencia, siendo desarrollados del más independiente al más dependiente. El proceso 'Prototipo' permitirá realizar una iteración por cada módulo propuesto, sin descartar hacer subiteraciones dentro de cada iteración; esto permitirá construir el prototipo final componente por componente. Es de aclarar que un componente no será agregado al prototipo final hasta que no esté totalmente depurado y funcional, tarea para la cual las iteraciones serán claves.

2.1.1 Módulos

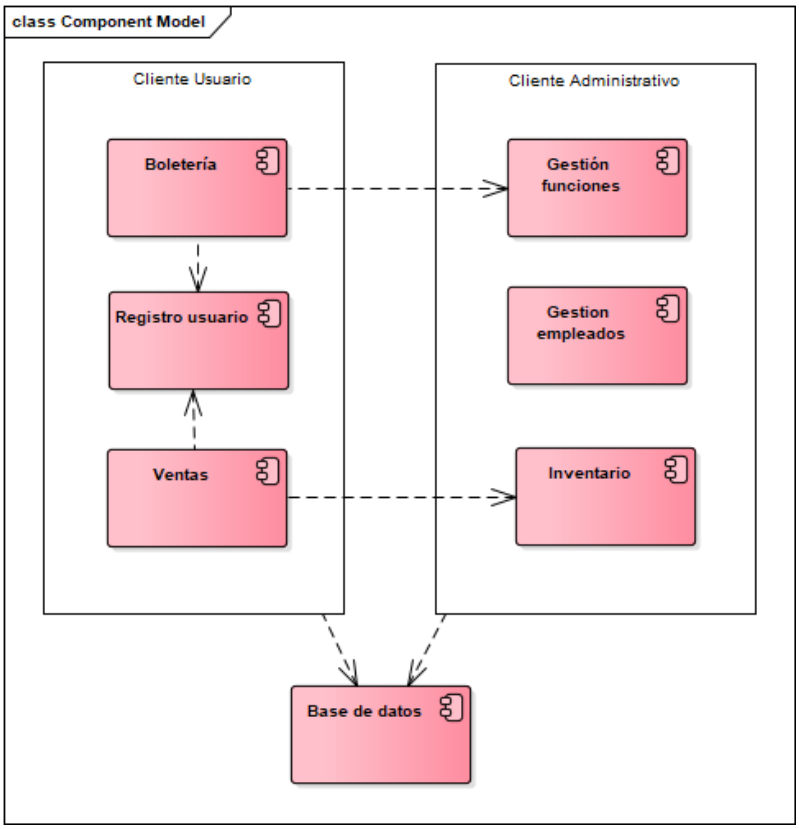

Figura 2.2: Módulos del sistema

- Aplicativo Cliente
  1. **Registro usuario:** Módulo encargado de la inscripción, modificación y eliminación de usuarios dentro de la plataforma y de la gestión de las suscripciones especiales que existan dentro de la cadena de cines.
  2. **Boletería:** Módulo encargado de la visualización y selección de películas, selección de asientos y reservas y ventas de boletería.
  3. **Ventas:** Módulo encargado de visualizar el catálogo de comidas y accesorios y generar tickets de pago para las distintas ventas.

- Aplicativo Administrativo
  1. **Gestión funciones:** Módulo encargado de agregar, eliminar y asignar funciones y salas a las distintas películas disponibles en el cinema, así como agregar próximos estrenos y eliminar películas de la cartelera.
  2. **Registro empleados:** Módulo encargado de llevar el registro de los empleados que trabajan en cada cinema, así como agregarlos o eliminarlos.
  3. **Inventario:** Módulo encargado de llevar control sobre el inventario de materia prima de confitería y maquinaria dentro de cada cinema.

- **Base de datos:** Módulo encargado de la insersión, actualización, modificación y almacenamiento de toda información que posteriormente utilizará el sistema

2.2 Diagrama de Gantt

Se estimó el tiempo del proyecto en total 74 días, y bajo la premisa de que en promedio se dedicarán 10 días a cada uno de los modulos como máximo, teniendo como fecha de inicio el día 11 de abril del 2019 y como fecha final el 23 de junio de 2019. Teniendo en cuenta los módulos mostrados en la imagen 2.2 se llegó al acuerdo de que la forma más eficiente de sería desarrollarlos en el siguiente orden:

- Módulo de bases de datos.
- Módulo de Gestión de empleados
- Módulo de Inventario.
- Módulo de Gestión de funciones.
- Módulo de Registro de Usuario.
- Módulo de ventas.
- Módulo de Boletería

En las imágenes 2.3 y 2.4 se muestran los tiempos dedicados para el desarrollo de cada uno de estos módules, mediante un diagrama de Gantt

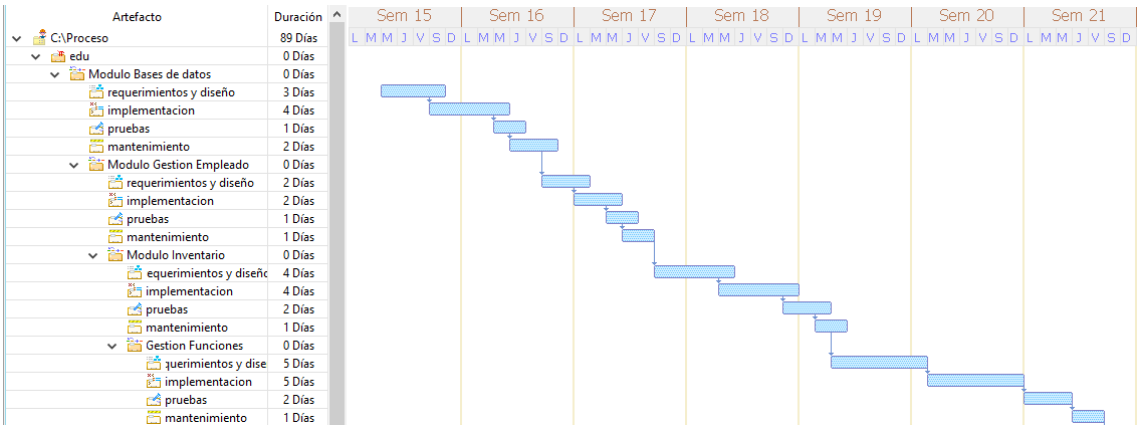

Figura 2.3: Proyección del desarrollo (1)

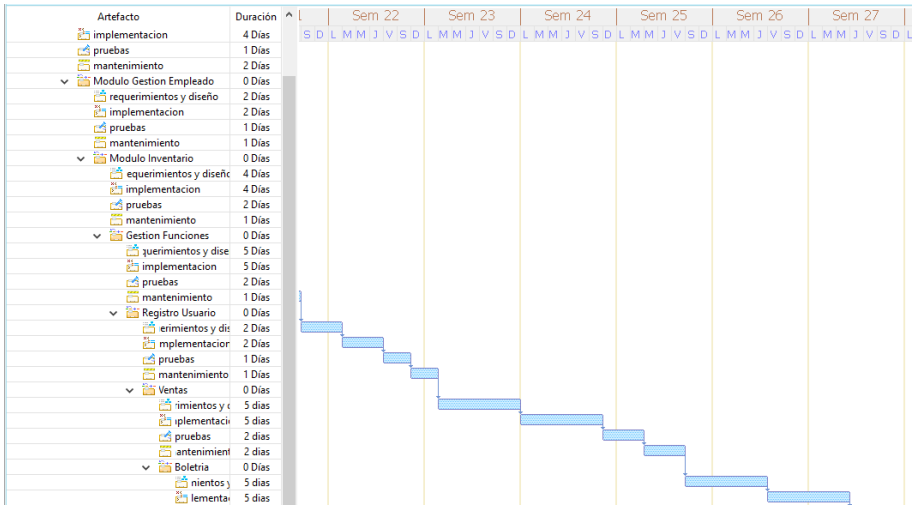

Figura 2.4: Proyección del desarrollo (2)

2.3 Metodología

La metodología a usar durante la implementación de cada uno de los módulos del proyecto será la metodología 'SCRUM', la cual está estrechamente relacionada con el proceso de desarrollo 'Prototipo', ya que permite definir unos objetivos claros al inicio de cada iteración y posteriormente generar un entregable que permita evaluar lo hecho para iniciar una nueva iteración. Además de eso, esta metodología permite un desarrollo incremental, que permitirá corregir y agregar funcionalidades no previstas en un principio o que hayan sido mal concebidas en iteraciones anteriores.

2.4 Open Source

En la actualidad el concepto de Open source ha cobrado fuerza ycada vez es más importante esta corriente de desarrollo, pues no solamente permite tener exceso a código fuente de otras personas, sino que da la posibilidad de encontrar respuestas a problemas que otras personas han tenido y poder integrarlas en el proyecto. Por razones como estas es que a lo largo del desarrollo se tendrán en cuenta recursos Open Source para ayudarse en el desarrollo y así poder resolverlo de la mejor manera posible.

II

DISEÑO

3. Requerimientos

3.1 Introducción

Dentro del aplicativo, se plantean un serie de requerimientos de tipo cliente y tipo desarrollador,los cuales buscan plantear plasmar las necesidades del cliente y las temas que deben tener en cuenta los desarrolladores para crear un producto. Dichos requerimientos serán abordados y explicados a lo largo de este capítulo.

3.2 Requerimientos tipo C

Estos son directamente los requerimientos funcionales del cliente, son aquellos requerimientos que están completamente ligados con las necesidades que necesita el cliente.

A continuación se espeficarán y describirán cada uno de los casos de uso expuestos:

3.3 Diagrama de Casos de Uso

Estos diagramas son los que permiten mostrar la relación que existe entre los usuarios y el sistema, mediante ellos se busca explicar qué es lo que esperan que el aplicativo haga.

3.4 Diagrama de Secuencia

Este tipo de diagrama es el que permite entender cómo se espera que funcione el aplicativo ante los deseos del usuario, están muy relacionados con los diagramas de casos de uso, relacionan lo que quieren los usuarios con el funcionamiento interno del sistema para llevarlo a cabo.

3.5 Diagramas de Comunicación

Este diagrama es un diagrama es quizás el más complejo de los tres, no porque tenga una realización díficil, más bien es un diagrama que busca mostrar como es la interacción entre los distintos objetos que componen el requerimiento, está muy ligado con el diagrama de secuencia, pues muestra información muy similar.

A continuación se muestran los diagramas de casos de uso, secuencia y comunicación respectivamente que se han encontrado para este aplicativo:

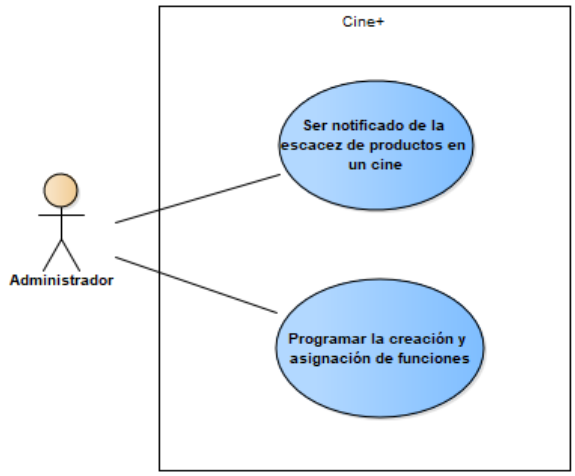

Figura 3.1: Casos de uso del sistema

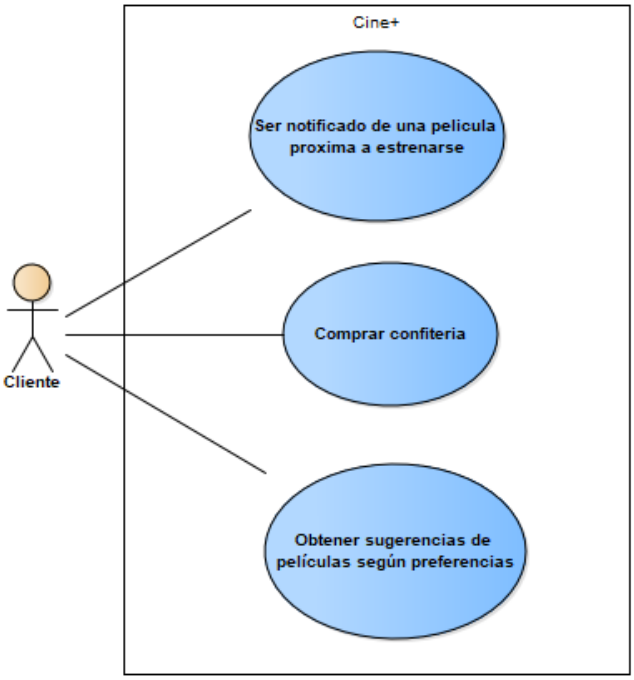

Figura 3.2: Casos de uso del sistema

RF- Notificación de escasez de productos	Ser notificado de la escasez de productos en un cine	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un administrador ingrese a la plataforma.	
Precondición	El cine del cual esté encargado el administrador debe tener productos registrados Debe establecer un límite mínimo de existencia de productos	
	Paso	**Acción**
	1	El administrador ingresa a la plataforma con su usuario y contraseña.
	2	El sistema emite un mensaje por pantalla indicando escasez de un producto dentro del cine.
	3	El administrador recibe la notificación y se dirige hacia la pestaña de inventario para verificar la existencia del producto.
Postcondición	El sistema debe emitir la notificación cada vez que se ingrese a la plataforma hasta que la existencia de dicho producto sea aumentada.	
Excepciones	**Paso**	**Acción**
	1	El sistema notifica que el cine no tiene ningún producto registrado.
	2	El administrador es notificado y dirigido a la pestaña de inventario para actualizar existencias.
Rendimiento	**Paso**	**Cota de tiempo**
	1	10 segundos
	2	1 segundos
Importancia	Importante	
Urgencia	Inmediatamente	

Figura 3.3: Requerimientos Tipo C

RF- Creación funciones	Programar la creación y asignación de funciones	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando un super administrador ingrese una nueva película a la cartelera de cines	
Precondición	El usuario dentro de la plataforma debe ser super administrador El usuario ya ha ingresado a la plataforma	
	Paso	**Acción**
	1	El administrador ingresa una nueva película para ser agregada a la cartelera
	2	El administrador indica el tiempo que estará en cartelera dicha película y la cantidad de funciones por día en cada cine que deberán presentarse
	3	El sistema notifica que se han realizado las asignaciones y muestra por pantalla la distribución de las funciones en cada cine
Postcondición	Cada administrador de cada cine podrá consultar las funciones que fueron creadas	
Excepciones	**Paso**	**Acción**
	1	El sistema notifica que no es posible crear las funciones debido a disponibilidad de horas en los distintos cines
	2	El administrador ingresa nuevamente los datos de funciones por día y tiempo en cartelera modificados para asignar de nuevo las funciones.
Rendimiento	**Paso**	**Cota de tiempo**
	1	120 segundos
	2	1 minuto
Importancia	Importante	
Urgencia	Inmediatamente	

Figura 3.4: Requerimientos Tipo C

RF-3	Ser notificado del estreno próximo de una película.	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando se publique una nueva película en cartelera.	
Precondición	El cliente debe estar registrado en la plataforma de cine	
Secuencia Normal	**Paso**	**Acción**
	1	El sistema debe reconocer la película que va a ser estrenada en el caso de uso RF-2
	2	El sistema generará una alerta para los usuarios que tengan preferencia por la película a estrenar.
Postcondición	El Cliente podrá reservar boletas para el estreno de la película.	
Excepciones	**Paso**	**Acción**
	1	Si el sistema no encuentra ningún usuario con preferencias por ese tipo de películas, notificará a todos.
Rendimiento	**Paso**	**Cota de tiempo**
	1	120 segundos
Importancia	importante	
Urgencia	Hay presión	

Figura 3.5: Requerimientos Tipo C

RF- 4	Obtener sugerencia de películas según preferencias	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso, mostrar a los usuarios sugerencias de las películas en cartelera según sus preferencias.	
Precondición	El Usuario deberá estar registrado, además deberá haber registrado sus preferencias.	
Secuencia Normal	**Paso**	**Acción**
	1	El sistema va a comparar las preferencias del usuario con las películas en cartelera.
	2	El sistema notificará las películas que están en
	El Clien	cartelera con sus preferencias.
Postcondición	El Cliente podrá reservar boletas para el estreno de la película.	
Excepciones	**Paso**	**Acción**
	1	Si el usuario no registra géneros preferidos se direccionará para que pueda registrarlos.
	2	Si no existen películas que coincidan con sus preferencias, se notificará y mostrará toda la cartelera.
Rendimiento	**Paso**	**Cota de tiempo**
	1	120 segundos
Importancia	importante	
Urgencia	inmediatamente	

Figura 3.6: Requerimientos Tipo C

RF- 5	Comprar comida por la plataforma	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso: Permitir compras de comida por medio de la plataforma	
Precondición	El cliente deberá estar registrado previamente, La comida seleccionada deberá estar habilitado para la venta.	
Secuencia Normal	**Paso**	**Acción**
	1	El Cliente seleccionará los combos de comida que quiere consumir.
	2	El sistema generará un seria de confirmación para la compra realizada.
Postcondición	El sistema estará en disposición de comparar el serial recibido y relacionarlo con la compra realizada.	
Excepciones	**Paso**	**Acción**
	1	Si el cliente no está registrado se le dará la opción de registrarse y poder acceder a esta oferta.
Rendimiento	**Paso**	**Cota de tiempo**
	1	120 segundos
Importancia	importante	
Urgencia	inmediatamente	

Figura 3.7: Requerimientos Tipo C

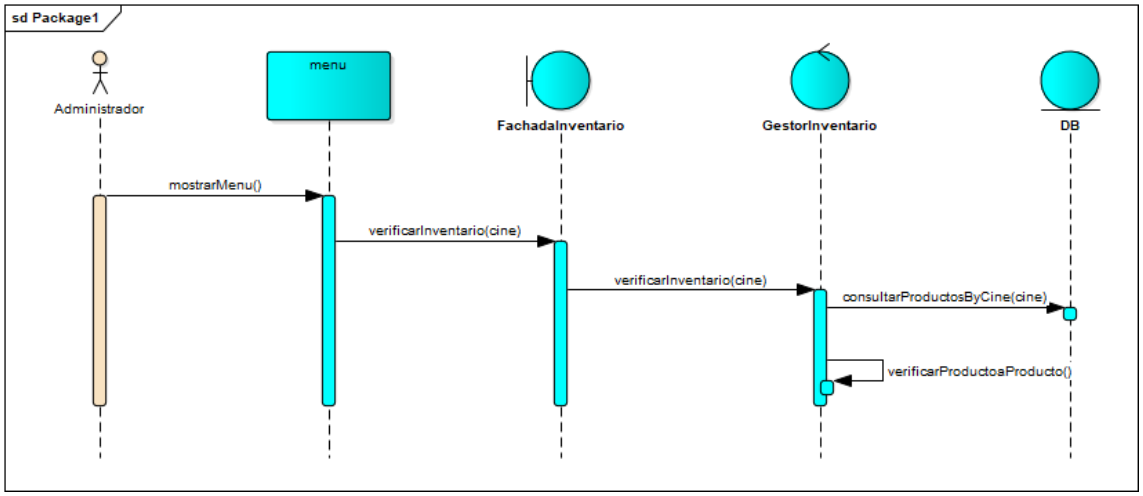

Figura 3.8: Diagrama de secuecia: Notifición de escacez de un producto

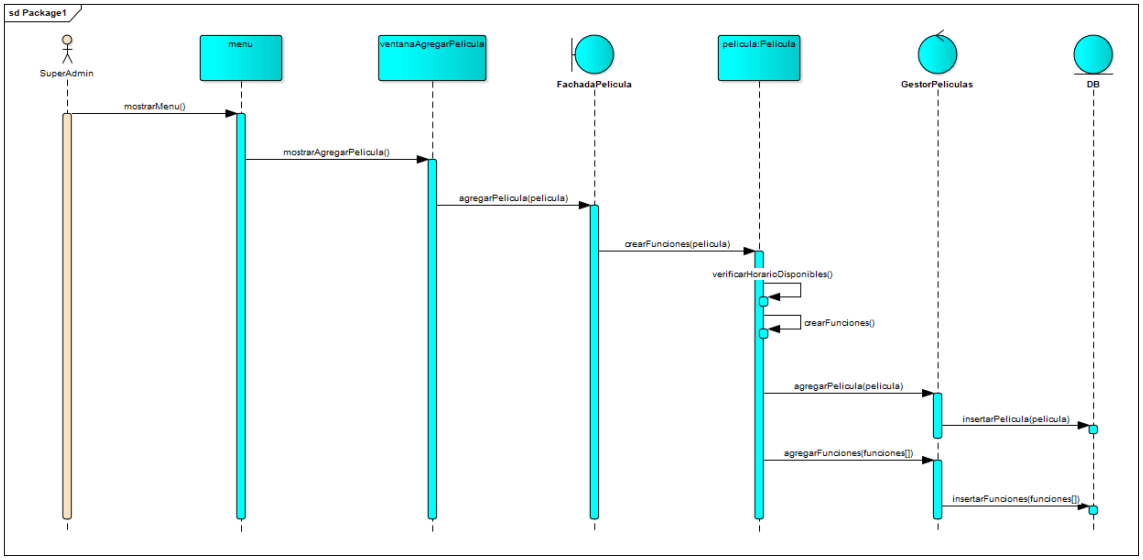

Figura 3.9: Diagrama de secuecia: Programar la asignación y creación de funciones

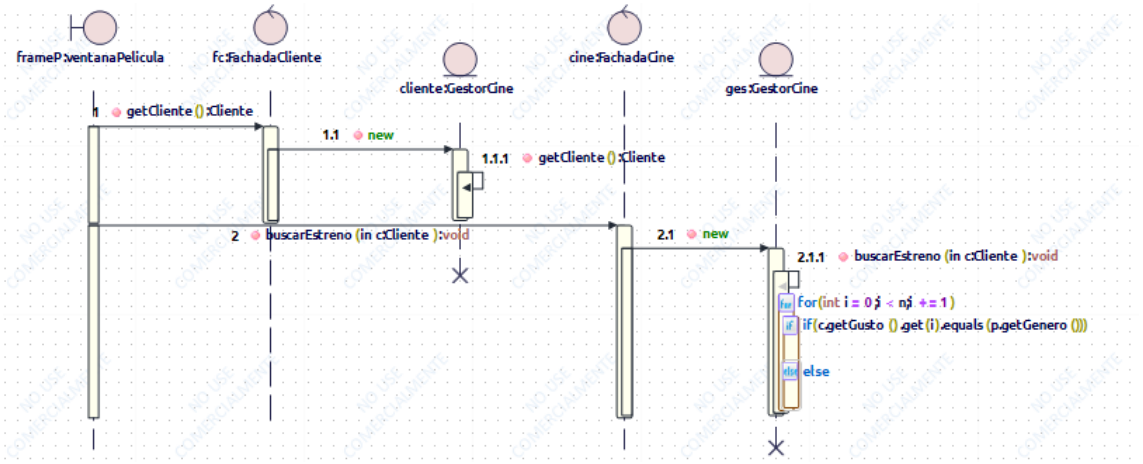

Figura 3.10: Diagrama de secuecia: Ser notificado del estreno próximo de una película

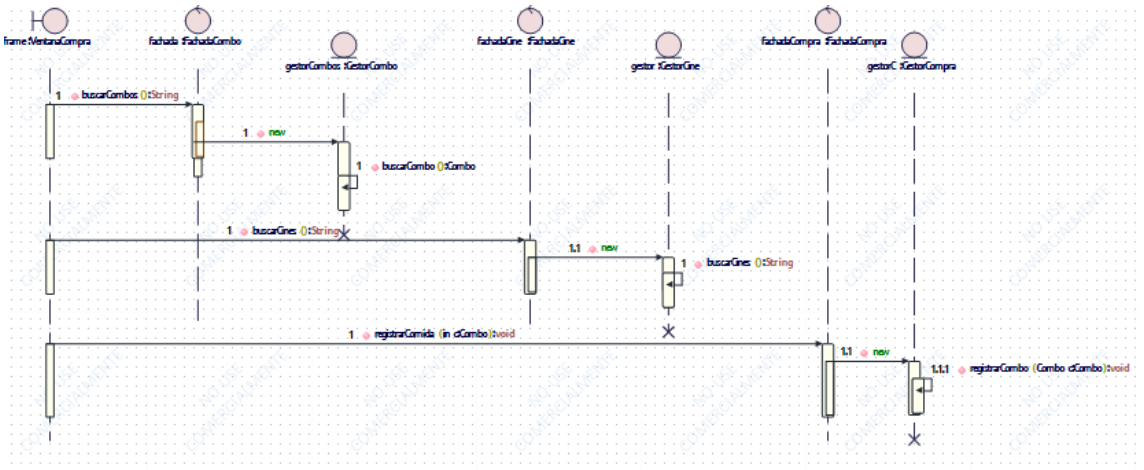

Figura 3.11: Diagrama de secuecia: Comprar comida por la plataforma

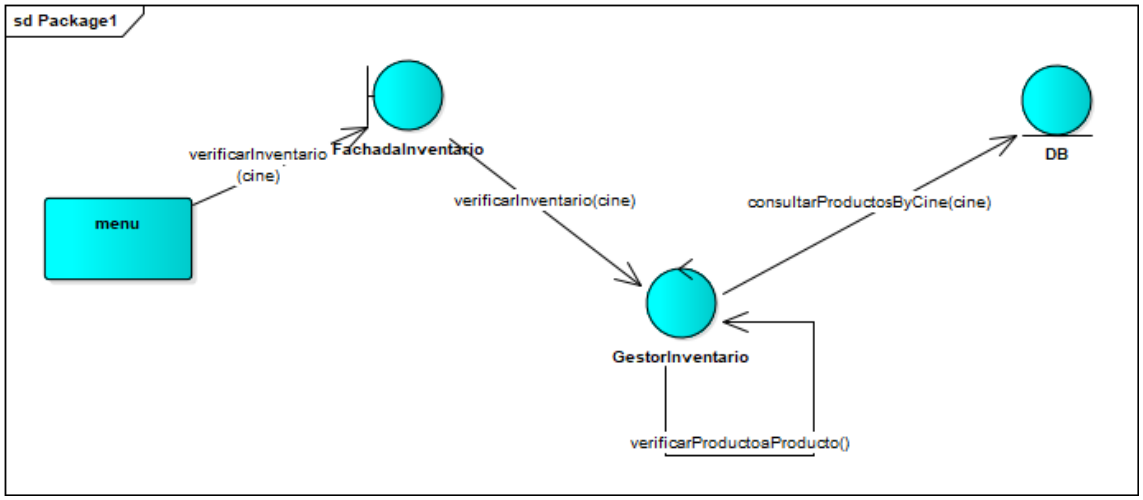

Figura 3.12: Diagrama de comunicación: Notificación de escacez de un producto

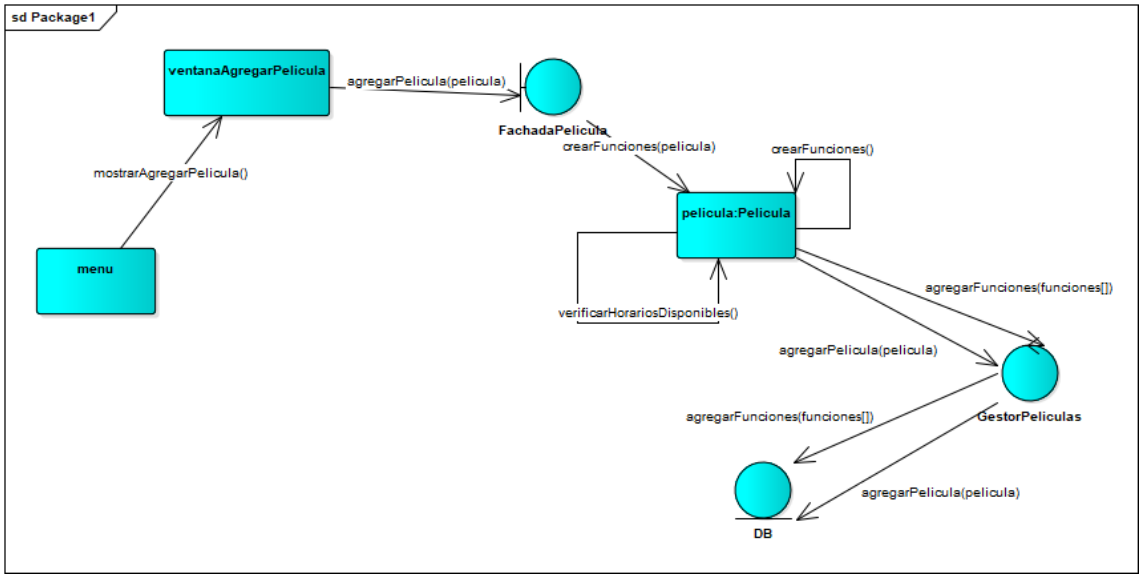

Figura 3.13: Diagrama de comunicación: Programar creación y asignación de funciones

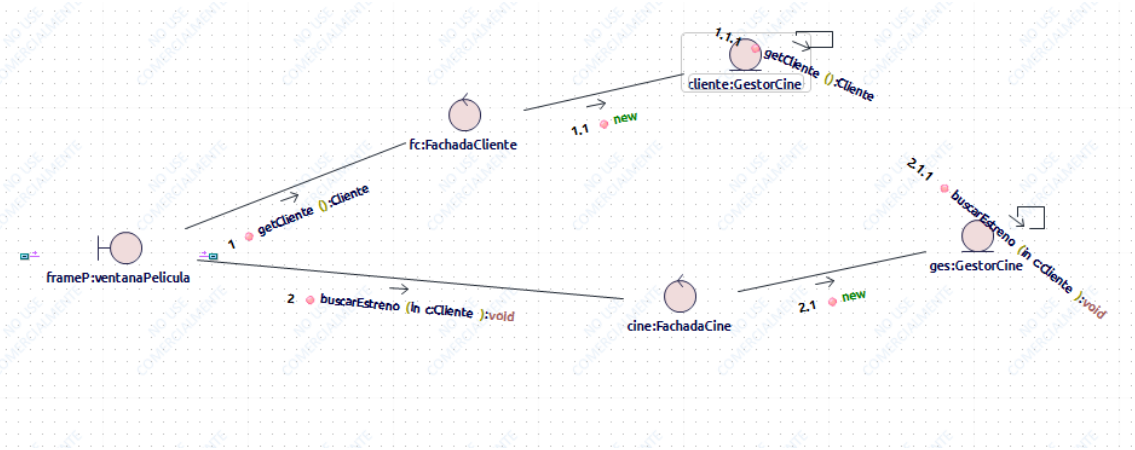

Figura 3.14: Diagrama de comunicación: Ser notificado del estreno próximo de una película

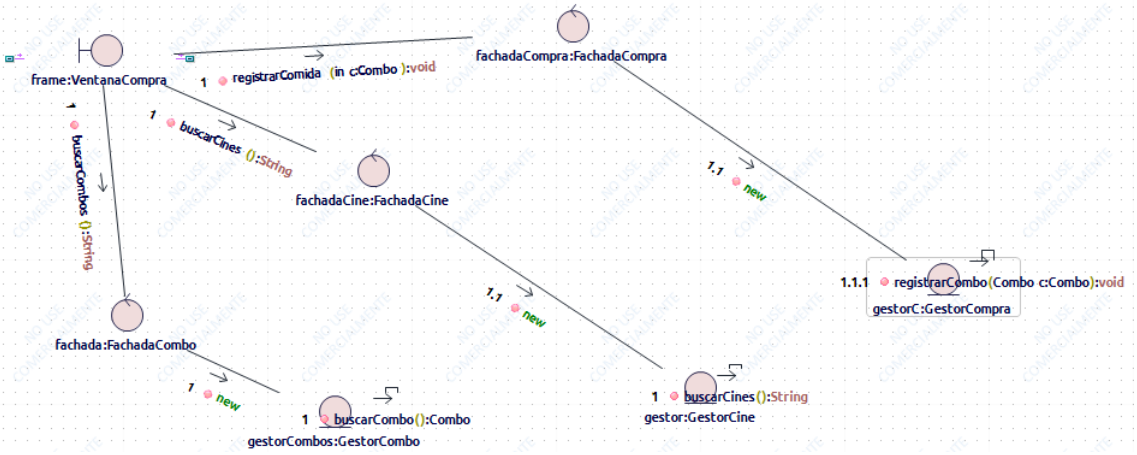

Figura 3.15: Diagrama de comunicación: Comprar comida por la plataforma

4. Diagrama de clases

Los diagramas de clase son una de las herramientas UML más utilizadas para el diseño de software, ya que mediante ellos se puede demostrar la manera en que interactúan las distintas clases de un programa entre sí, además de la manera de ver la manera en que interactúan con el programa. Las clases están conformadas por tres partes importantes: el nombre de clase, los atributos y las

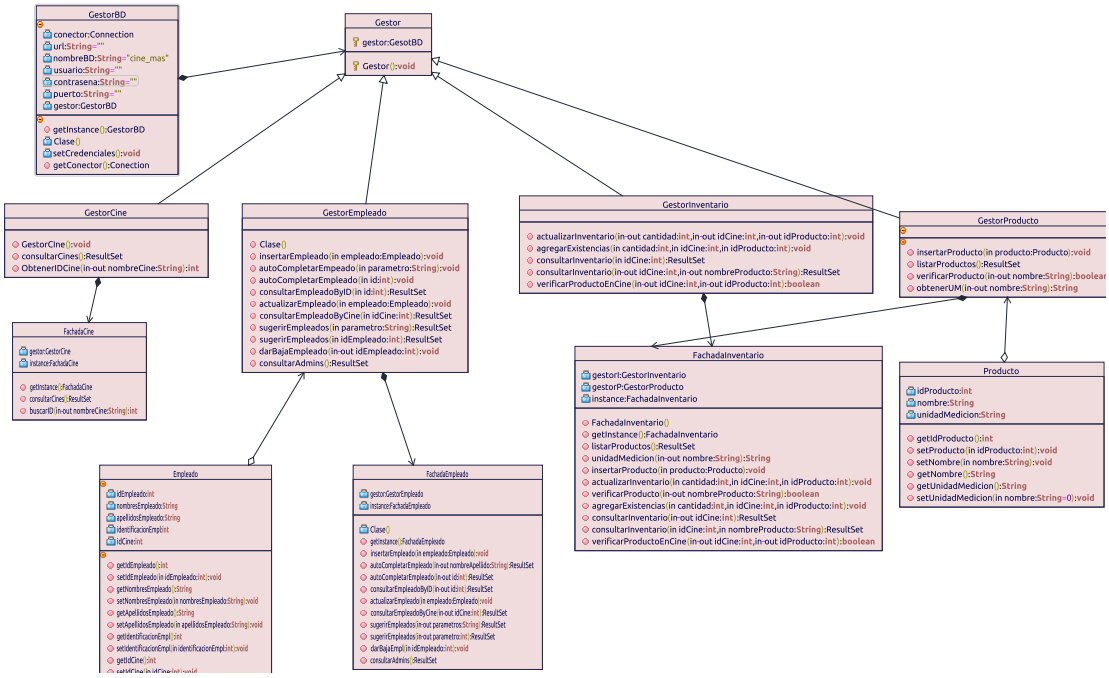

Figura 4.1: Diagrama de clases

operaciones.

4.0.1 NOMBRE DE CLASE:

En esta sección es donde se define la clase, generalmente se nombran con sustantivos y se busca, mediante estos expresar de manera implícita lo que representa la clase.

4.0.2 ATRIBUTOS

Son las características propias de la clase, los elementos que la hacen única. Todos los atributos tienen ciertas características especiales que determinarán la manera con la que se interactúa con ellos.

VISIBILIDAD

Cuando hablamos de la visibilidad de un atributo se hace referencia a facilidad que tienen otras clases de ver los atributos de una clase en específico, existen cuatro tipos de visibilidad.

- **Privada:** este tipo hace referencia a que la única clase que tiene acceso a un atributo es la clase dueña del atributo.
- **Protegida:** este tipo está fuertemente ligado con la herencia, pues significa que los atributos son privados para todas las clases excepto aquellas que extiendan de la clase que tenga el atributo protegido.
- **Paquete:** este tipo indica que el atributo es privado para todas las clases que no pertenezcan al mismo paquete del atributo dueño del atributo.
- **Pública:** finalmente este tipo indica que cualquier clase existe puede ver el atributo de la clase.

ALCANCE

Esta característica de los atributos nos indica el lugar que va a ocupar en memoria, partiendo del hecho de que en todo programa se cuenta con tres tipos de memoria: memoria estática, memoria de pila, memoria de montículo.

- **Memoria estática:** Esto nos indica que un atributo no puede cambiar en tiempo de ejecución, es decir, siempre tendrá el mismo valor.
- **Memoria de pila:** la mayoría de los atributos se encuentra en esta memoria, son datos que cambian constantemente.
- **Memoria de montículo** En esta memoria se encuentran todos los atributos que necesitan ser instanciados, generalmente con la palabra reservada **new**

PROPIEDADES

Esta característica de los atributos es lo que nos indica la posibilidad que se tiene de ser modificados en tiempo de ejecución, se tienen de do tipos: Final y volátil.

- **Final o Constante:** ste tipo de atributo es un atributo que una vez definido no puede ser modificado en tiempo de ejecución.
- **Volátil:** Por otro lado esta propiedad indica que el atributo puede ser optimizado en tiempo de ejecución.

La definición de un atributo dentro de una clase se realiza de la siguiente manera:

Visibilidad nombre [ ]: tipo = valor propiedad

Donde los "[ ]" hacen referencia a la cantidad de elementos que se tienen y el subrayado al alcance que tienen dichos atributos.

4.0.3 OPERACIONES

Cuando se habla de las operaciones de una clase se hace referencia a las "habilidades" que esta tiene para poder interactuar con el programa, al igual que los atributos tiene ciertas características que las diferencian las unas de las otras.

VISIBILIDAD

La visibilidad de una operación es muy parecida con la visibilidad de los atributos, con la diferenciación de que en general solo cuenta con visibilidad pública y privada.

Para saber si una operación es pública o privada hay que tener un término que se llama cohesión secuencial, es decir, si la operación necesita de otras operaciones para funcionar correctamente.

Una operación es privada si dicha operación realiza una actividad que solamente tiene sentido dentro de la clase.

una operación es pública si la operación permite la interacción con otras clases. Generalmente dentro de ella aparece la cohesión secuencial, es decir, ella llama a las operaciones privadas de la clase. Se denomina interfaz, pues es un mediador con otras clases.

PROPIEDADES

En el caso de las operaciones las propiedades indican el tipo de acción que permite realizar.

- **Final:** La operación no permite sobrecarga, es decir, las clases hijas no podrán utilizarla.
- **Query:** Es una operación que no permite modificaciones, los atributos de la clase no son modificables.
- **Modify:** indica que la operación tiene la habilidad de modificar de manera permanente los atributos de la clase.
- **Synchronized:** Esta propiedad tiene sentido cuando se habla de hilos, pues determina que un hilo no pasará la responsabilidad hasta no haber terminado el proceso.

PARÁMETROS

Los parámetro son la información que recibe la operación para poder realizar el proceso asignado, tienen una dirección que indica la característica del atributo:

- **in:** El atributo se pasa por valor.
- **out:** el parámetro será retornado.
- **in-out:** el parámetro se pasa por referencia.

La manera en que se escriben las operaciones dentro de una clase es la siguiente:

Visibilidad nombre (parámetros): retorno propiedad

4.0.4 RELACIONES

Estas relaciones tienen la característica de tener reuso de caja negra, es decir que las clases que se relacionan mediante ellas no tienen conocimiento de cómo se lleva a cabo una operación, pero aún así pueden hacer uso de ellas. Por otro lado, tienen el problema de que tienen un acoplamiento muy alto, lo que significa que la necesidad entre las clases es algo muy elevado.

Cliente/proveedor

Estas relaciones se dividen en dos grandes subramas: Dependencias y Asociaciones.

Dependencia: Estos casos se dan cuando una clase utiliza otra clase para poder llevar a cabo una función (include, call, instance of, etc).

Asociación: Esta tiene de dos tipos, el primero, la agregación, indica que la relación entre ellos no es vital, es decir, el cliente puede vivir sin el proveedor. Por otro lado la composición indica una relación vital, es decir, el cliente no puede vivir, ni tener sentido a no ser que tenga al proveedor.

Generalización/Implementación

Estas relaciones tienen la característica de tener reuso de caja blanca, es decir que las clases que se relacionan mediante ellas tienen conocimiento de cómo se lleva a cabo una operación, e incluso pueden llegar a modificarlo. Por otro lado, tienen la ventaja de que tienen un acoplamiento muy bajo, lo que significa que la necesidad entre las clases es algo que puede pasar a segundo plano.

Por un lado las generalizaciones permiten crear una clase a partir de otra, con las mismas características, pero con la idea de que tenga un rol diferente. Por otro lado las implementaciones indican el uso de algunas operaciones de una interfaz en una clase determinada.

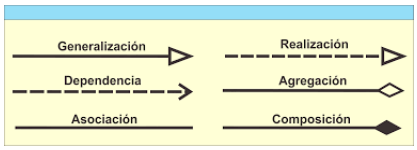

Figura 4.2: Relaciones entre clases

5. Patrones de diseño

Para la estructuración del proyecto en curso se contemplaron los siguientes patrones de diseño:

- **Abstract Factory:** Abstract Factory es un patrón de diseño que provee una interfaz para la creación de familias de objetos o objetos dependientes relacionados entre sí sin especificar sus clases concretas. Concretamente, se busca implementar este patrón de diseño para

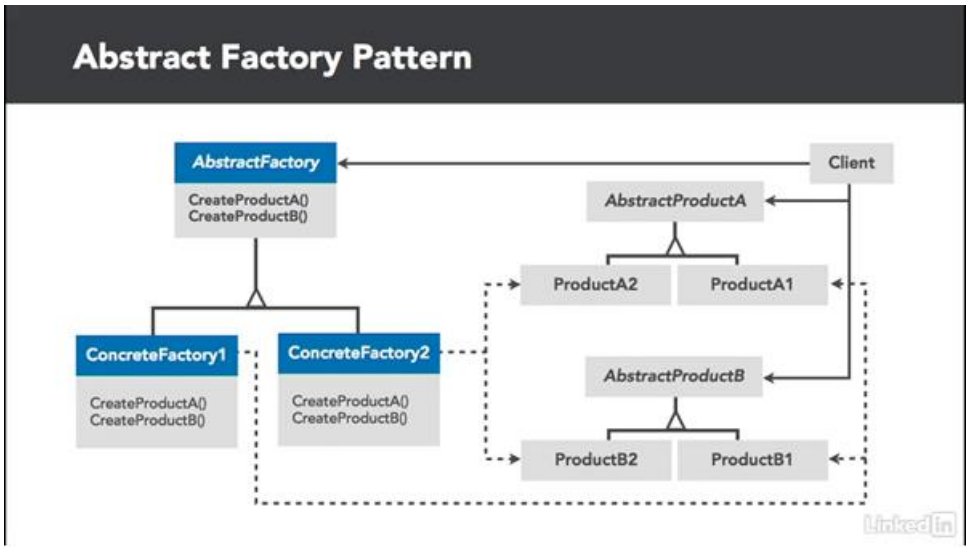

Figura 5.1: Patrón de diseño 'Abstract Factory'

otorgar flexibilidad al momento de querer migrar todas operaciones y consultas de la base de datos a otro DBMS. Para este proyecto inicialmente se está trabajando con el DBMS PostgreSQL, pero si se quisiera migrar a otro tal como Oracle, MySQL u otro, este patrón otorga flexibilidad y facilidad para hacerlo. Para implementarlo, se parte de la base de que las operaciones sobre la base de datos se pueden ver como una familia de dos objetos: una

conexión y un gestor. La conexión transportará todas las operaciones al DBMS que el gestor le solicite.

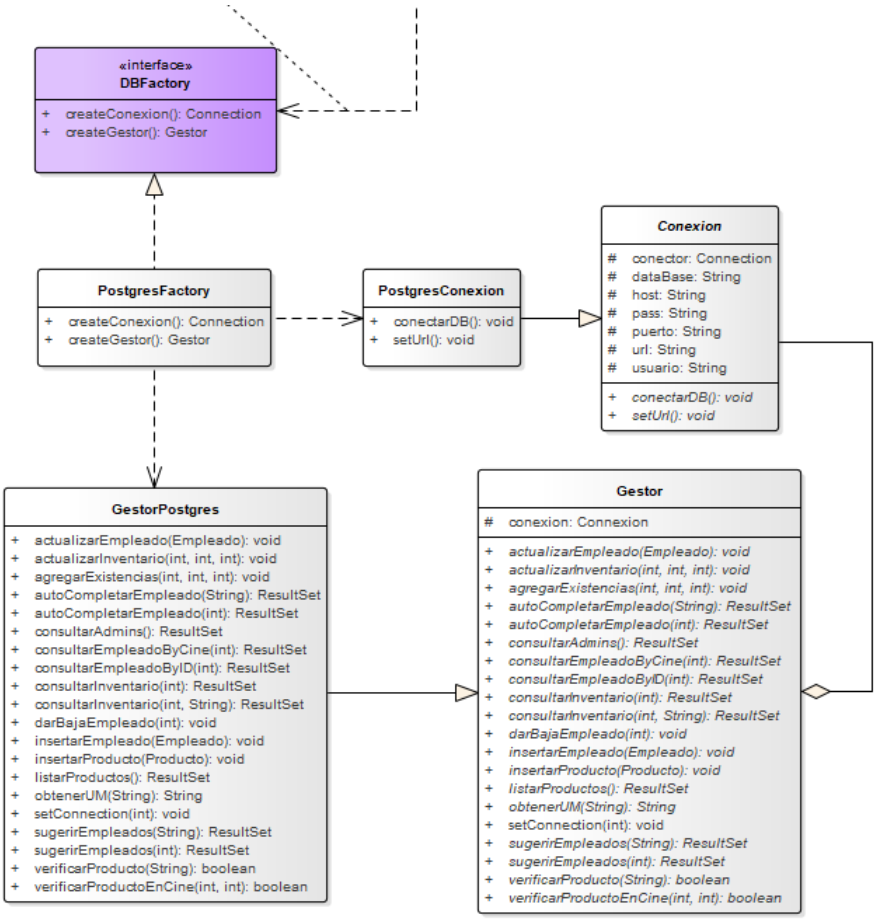

Figura 5.2: Patrón de diseño 'Abstract Factory' implementado dentro del proyecto

- **Facade:** El patrón de diseño fachada provee una interfaz unificada para un conjunto de interfaces en un subsistema. En el patrón de diseño 'Abstract Factory' que se planteó en el

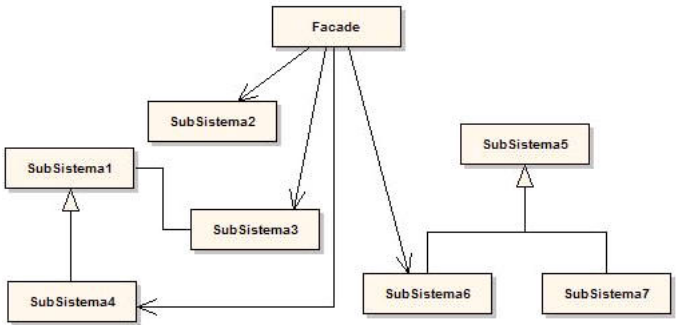

Figura 5.3: Patrón de diseño 'Facade'

punto anterior, la interacción de la base de datos se da por medio de la clase 'DBFactory', que hace el papel de fábrica de los componentes que permiten dicha conexión; pero aquellos clientes que deseen interactuar y realizar operaciones con dicha conexión no deberían tener una relación con la clase 'DBFactory', de tal manera que se plantea el uso de una **Fachada**

que establezca una interfaz que sirva de intermediaria entre ambas partes.

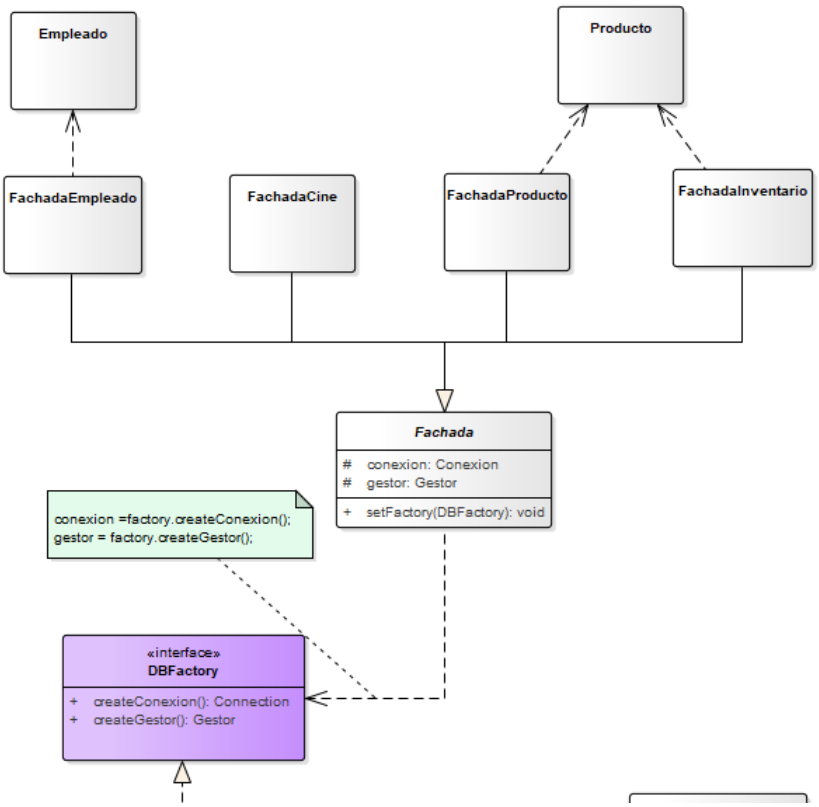

Figura 5.4: Patrón de diseño 'Facade' implementado

- **Singularidad (Patrón Prs):** El patrón de diseño de singularidad es un patrón -R el cuál es una mejora para la implementación relacionanda con el patrón singletón, con la ventaja de que mediante este patrón se pueden heredar todas las caraterísticas y ventajas que ofrece la clase padre.

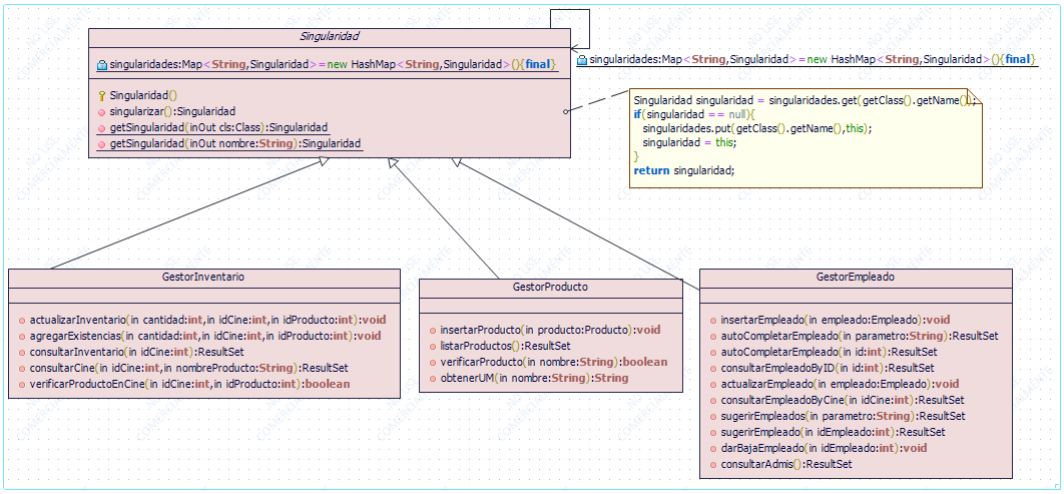

Figura 5.5: Patrón de diseño 'Singularidad' implementado

6. Diagramas de estado

Los diagramas de estado son una de las herramientas que permiten mostrar como es el comportamiento de los objeto o las variables dentro del programa, esto quiere decir, que permiten registarar como evolucionan lentamente los objetos en tiempo de ejecución.

Estos diagramas tienen dos elementos sobresalientes: los estados y las transiciones.

Los estados son los que indican los distintos cambios que tienen los objetos en el tiempo de ejecución, por otro lado las trancisiones son los eventos que suceden para que un objeto pase de un estado a otro. Las transiciones tienen tres elementos importantes: el disparador, la condición y el efecto.

El disparador es la acción que permite que genera el cambio; la condición, como su nombre lo dice, es el prerequisito que debe solventarse para que se genere e cambio y finalmente el efecto es el resultado que se produce en el objeto para ue pase al siguiente estado.

A continuación mostraremos un aplicación de estos diagramas, aplicandolos en el proyecto Cine+:

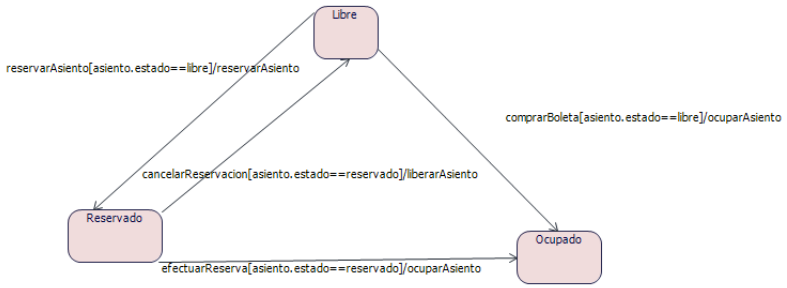

Figura 6.1: Diagrama de Estado

Como se puede ver en la anterior imagen, existen un cambio entre la dispoibilidad de un asiento,

pero una vez ocupado nunca puede cambiar de estado. La ventaja de los diagramas de estado es que permite ser incluidos en un diagrama de clases, tal y como se muestra en la siguiente imagen.

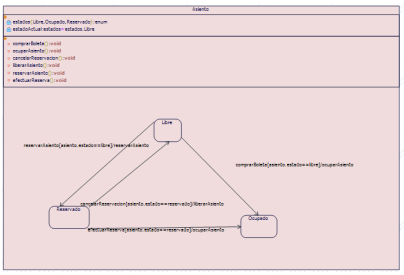

Figura 6.2: Diagrama de clases con estados

Una clase que contiene todas estas carecterísticas no es un diseño que sea correcto, esto se debe que en caso de que se tenga que agregar una nueva clase de disponibilidad se entrará en el problema de modificar toda la clase. Para evitar esta dificultad se puede utilizar un patron GoF de diseño, en este caso se usará el patrón estado, el cual da la posibilidad de extender las necesidades que puedan llegar a tenerse.

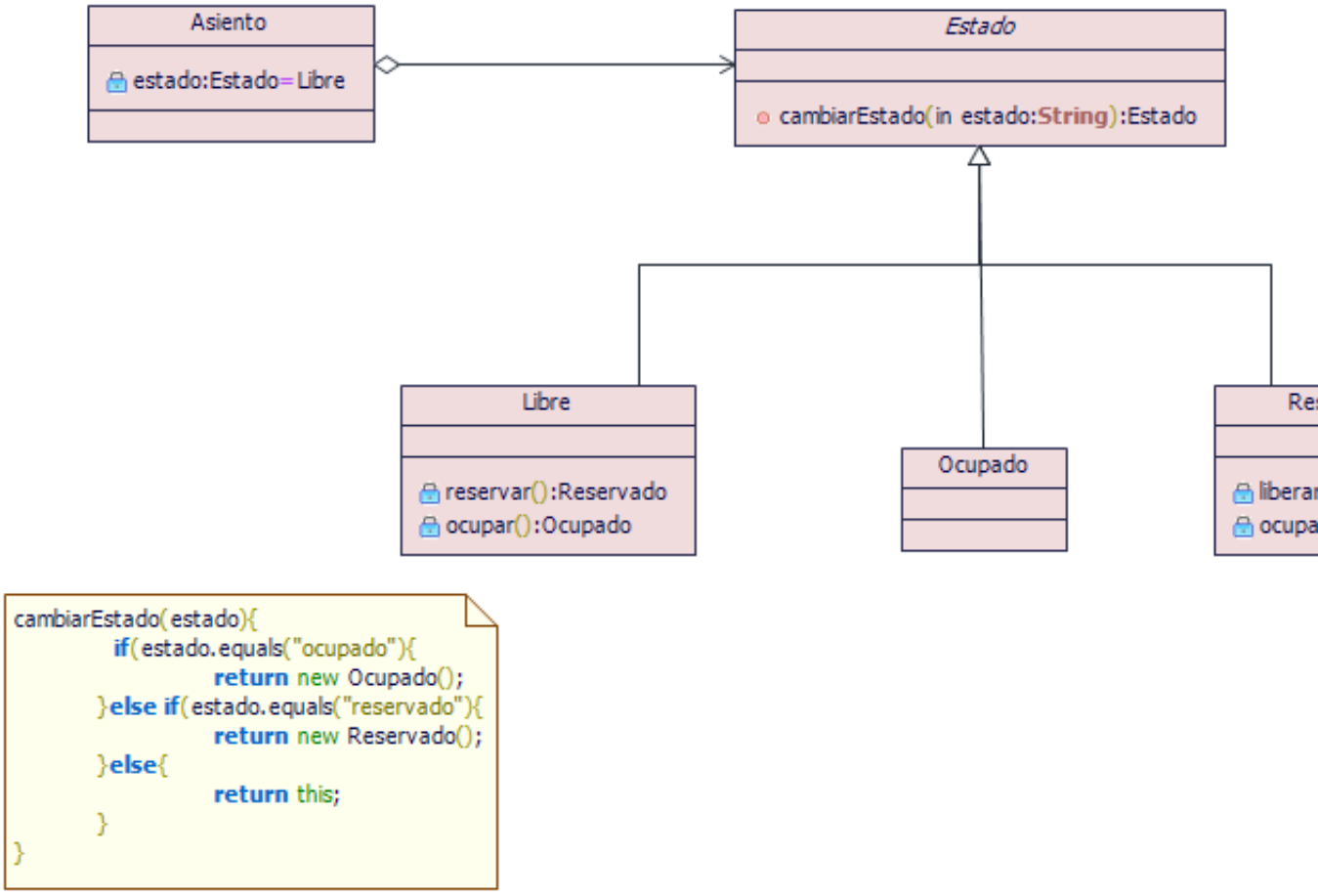

Figura 6.3: Patrón Estado