Proyecto Convivium

Documento de Diseño

EC-DA. Versión 1.0
8 de marzo de 2022
Estatus: Restringido

Resumen

Este documento contiene una especificación formal de la arquitectura del sistema *Convivium*, la cual se ha determinado en base al análisis y educción de casos de uso que se presentaban en el Documento de Análisis.

Fundamentalmente, en este documento se dará una explicación introductoria a la arquitectura de clases, siguiendo el patrón Modelo-Vista-Controlador, junto con un diagrama explicativo en UML. Después, se incluirán los pertinentes diagramas de secuencia UML para los casos de uso detallados en el Documento de Análisis.

Índice de Contenidos

1. Descripción de la Arquitectura del Sistema

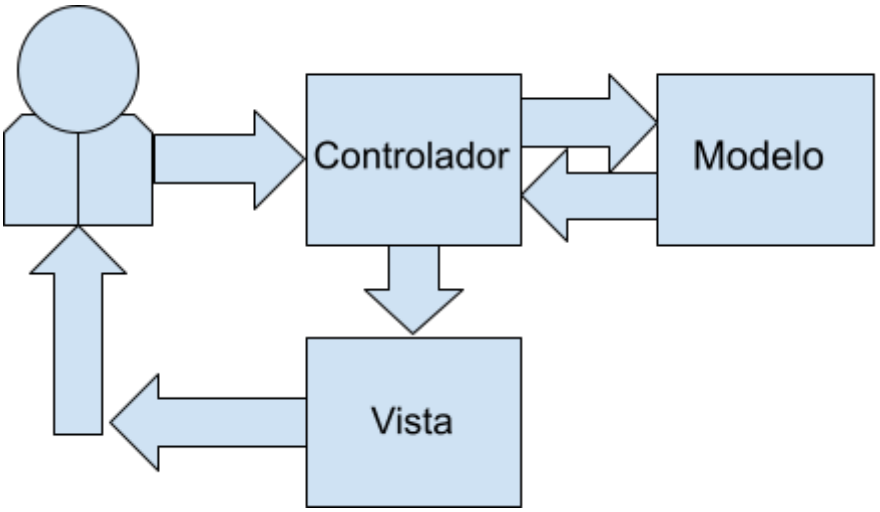
La arquitectura de este sistema informático se basa en un patrón MVC (Modelo Vista Controlador).

El patrón MVC, el cual parte de las iniciales de *Model-View-Controller*, es un patrón arquitectural basado en capas, las cuales buscan lograr su principal objetivo que consiste en la separación de los componentes de la aplicación, diferenciando los datos, lógica de negocio y su representación para cubrir la necesidad de la facilidad de reutilización eficiente de código.

La capa de modelo será la encargada de trabajar y operar con los datos con mecanismos tanto para consultar los mismos, como para actualizaciones de dichos datos.

La capa de vista es aquella que contiene la parte del código para presentar la información del modelo, produciendo la interfaz que muestre de forma adecuada dichos datos.

La capa de controlador es aquella encargada de responder a los eventos y acciones de los usuarios, invocando como respuesta de estas acciones al modelo y enviando respuestas a la vista, actuando como intermediario entre usuario y sistema, así como coordinador general del sistema final.



Para cumplir con este diseño, el diagrama de clases refleja el equivalente. En otras palabras, el controlador es la clase *Sistema*, que maneja todo el flujo de datos para actualizar el modelo *BaseDeDatos* de forma que la vista *GUI* muestre la información correcta al usuario final. El resto de clases (Usuario, Pedido, Restaurante…) serían el equivalente al modelo *BaseDeDatos* pero en memoria volátil, o sea en memoria RAM. ¿Qué significa esto último? Dichas clases permitirán reflejar la información obtenida de la base de datos en formato objeto, es decir, otorga un formato que la clase *Sistema*

podrá manejar. ¿Su propósito? Permitir la comunicación entre la aplicación y el usuario final, aportando un formato consistente de almacenamiento temporal que luego pueda ser utilizado para guardar o cargar datos en la base de datos.
Ahora que entendemos el propósito del diseño, sólo queda aclarar para qué sirve cada clase (aunque su propio nombre ya lo indica):

- Usuario. Clase abstracta que define los atributos y métodos comunes para los diferentes tipos de usuario de la aplicación.
- Administrador. Usuario capaz de editar, añadir o eliminar funcionalidad de la aplicación (por ejemplo, eliminar la posibilidad de hacer menús con productos externos).
- Propietario. Usuario que puede registrar sus restaurantes, así como los productos que ofrecen éstos.
- Repartidor. Usuario que recibirá notificaciones y avisos cuando haya un pedido realizado por su zona de tránsito.
- Cliente. Usuario capaz de realizar pedidos, aceptarlos o cancelarlos y pagarlos.
- Restaurante. Clase que almacenará los datos de un restaurante. Depende fuertemente de su Propietario.
- Item. Clase abstracta que define los atributos y métodos comunes para los diferentes tipos de productos de los restaurantes.
- ProductoLocal. Tipo de producto, de carácter local, que registra el Propietario en uno o varios de sus restaurantes. Almacena los datos de dicho producto local.
- ProductoExterno. Tipo de producto, de carácter ajeno, que registra el Propietario en uno o varios de sus restaurantes. Almacena los datos de dicho producto externo.
- Menú. Tipo de producto que consiste en la agrupación de uno o varios Item. Sigue el patrón Composite.
- Pedido. Clase que almacenará los datos de un pedido realizado por un Cliente. Depende fuertemente del Cliente.
- EstadoPedido. Enumeración que determina el estado de un pedido en un momento dado.
- Domicilio. Clase fuertemente dependiente del Cliente que indica todo lo necesario para poder entregar el Pedido de un Cliente en el lugar adecuado.
- Sistema. Clase *maestra* que controla todo el flujo de datos de la aplicación. Se encarga de comunicarse con la base de datos para la persistencia en guardado y cargado de datos, así como con la interfaz GUI para que muestre datos actualizados.
- GUI. Clase *representativa* que se identifica con la interfaz de la aplicación. No contiene métodos concretos pues poseerá todo el código necesario para implementar la interfaz gráfica (es la Vista del patrón MVC). Será el controlador quien esté “pendiente” de la GUI para realizar la lógica pertinente.
- BaseDeDatos. Clases *representativa* vacía que indica la existencia de una base de datos para la persistencia de datos, pero no se implementará una clase como tal. La lógica de las consultas a base de datos las hará el propio Sistema.
- SistemaGeolocalización. Sigue la misma línea que BaseDeDatos. Es un objeto externo que ayudará a distribuir la carga de pedidos entre los repartidores.

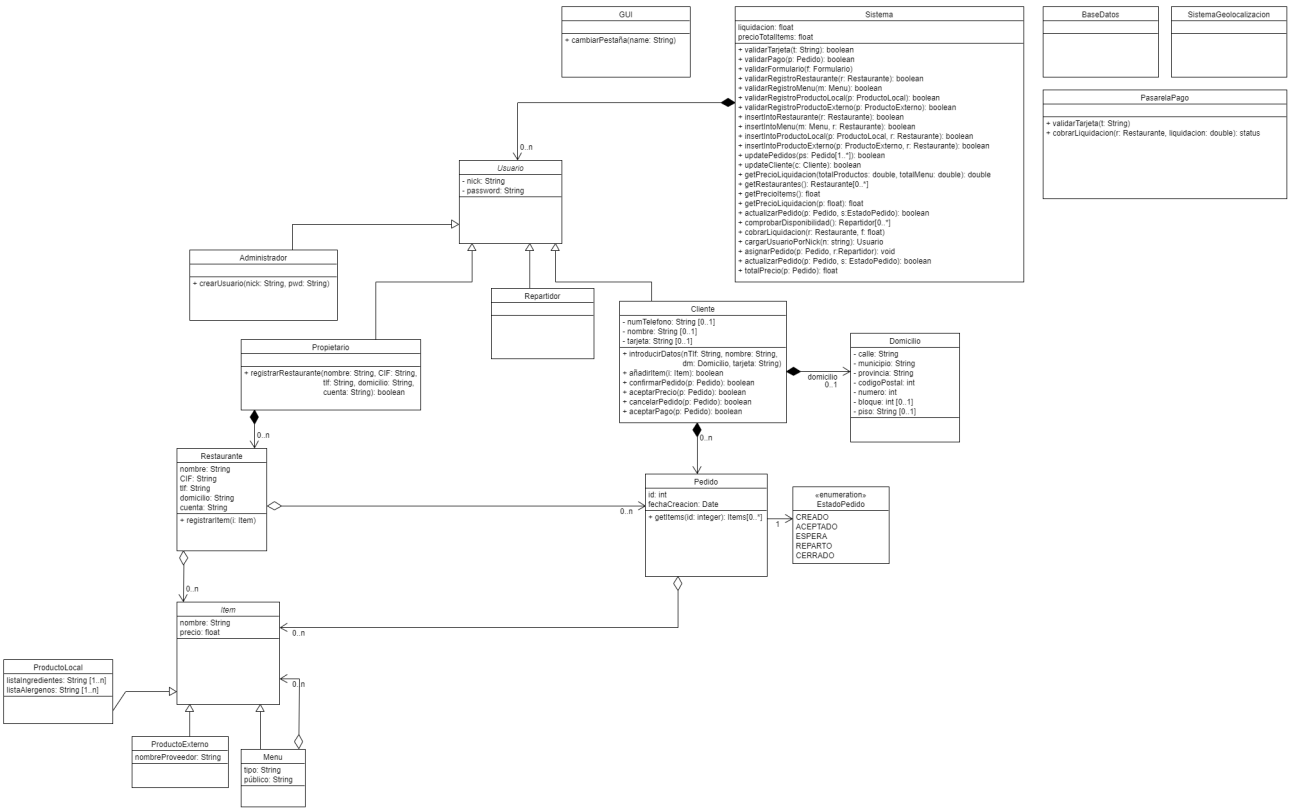
- PasarelaPago. Clase que hará las comprobaciones de tarjeta y las liquidaciones gracias a la comunicación con un objeto externo dado por un servicio de terceros.

2. Diagrama de Clases

El presente diagrama de clases muestra una aplicación donde gran parte de la funcionalidad reside entre la clase Sistema y la Base de Datos (a partir de ahora BBDD). ¿Esto qué refleja?

Existen dos tipos de aplicaciones: por un lado aquellas que guardan todo en memoria volátil, es decir, en la aplicación, y luego, antes de terminar la ejecución, vuelcan toda la información en la BBDD. Luego, están aquellas aplicaciones que guardan cada cambio que se hace en la BBDD, en "tiempo real".

En este proyecto se ha optado por seguir el diseño del segundo caso expuesto, el cual nos permite proporcionar un servicio de bajo consumo capaz de funcionar en sistemas informáticos con hardware no necesariamente puntero. También es posible recurrir a este diseño ya que se cuenta con una conexión a BBDD estable, de lo contrario no sería la solución más óptima en cuanto a diseño.



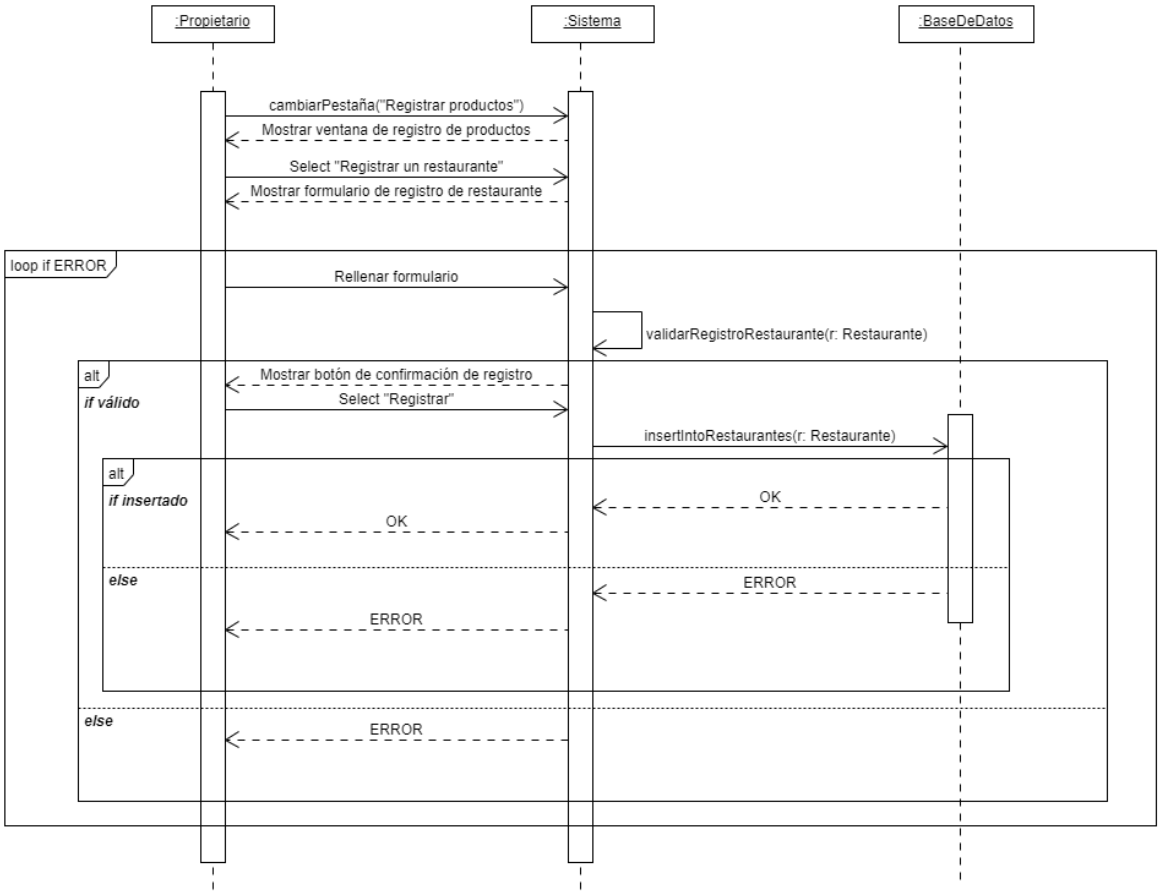
Los getters incluidos en el diagrama no son simples, ya que acceden a la base de datos para realizar consultas.

3. Diagramas de Secuencia

En este apartado se muestra, de forma detallada, el flujo de acciones e interacciones entre clases que tienen lugar en el desarrollo de diversos casos de uso. Esto se hace mediante diagramas de secuencia, cada uno asociado a uno de los casos de uso anteriormente expuestos en el Documento de Análisis.

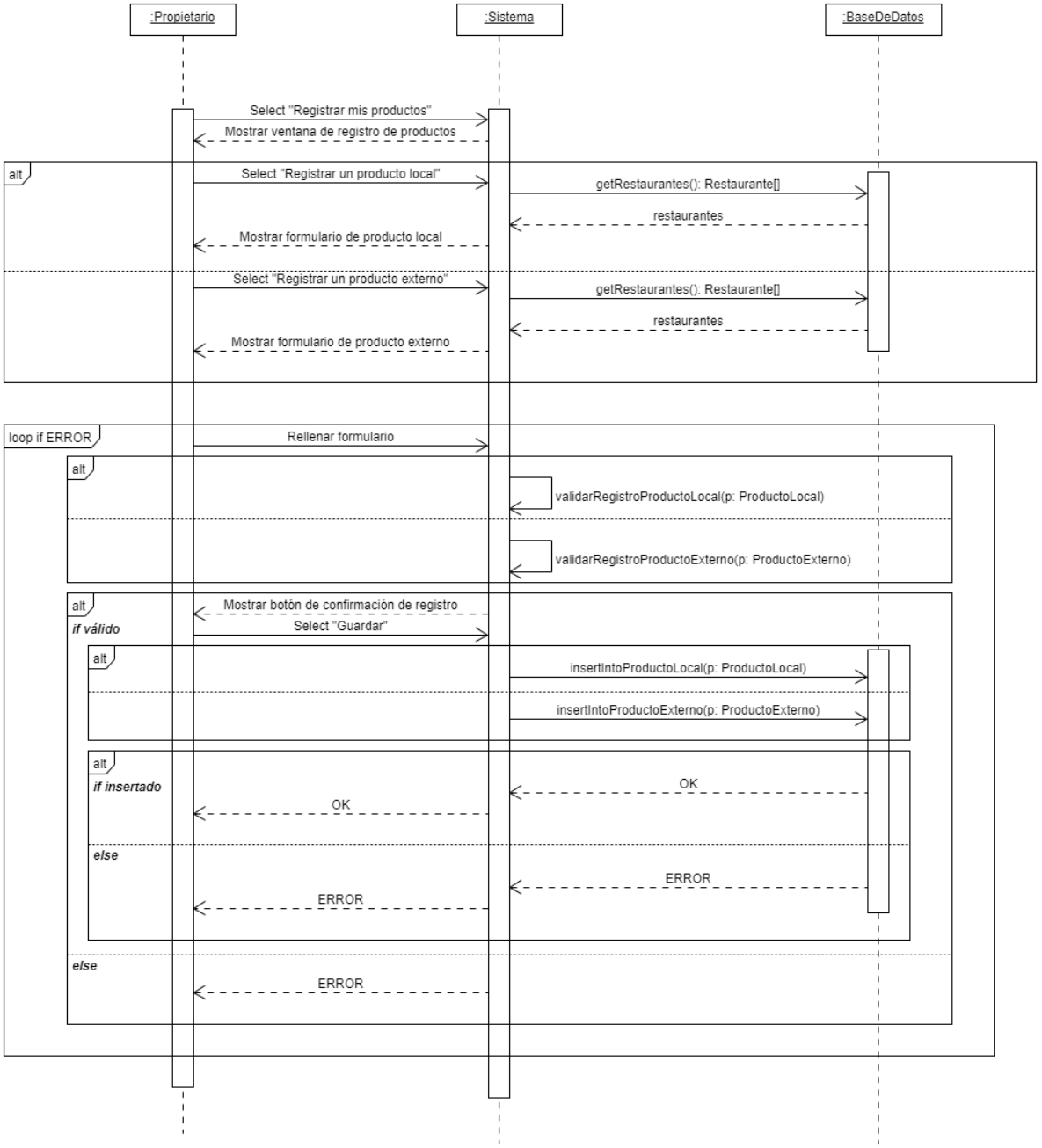
3.1. Diagrama de Secuencia Caso 4

Este caso de uso modela el registro de un restaurante por parte de un usuario con rol "Propietario de Restaurantes".



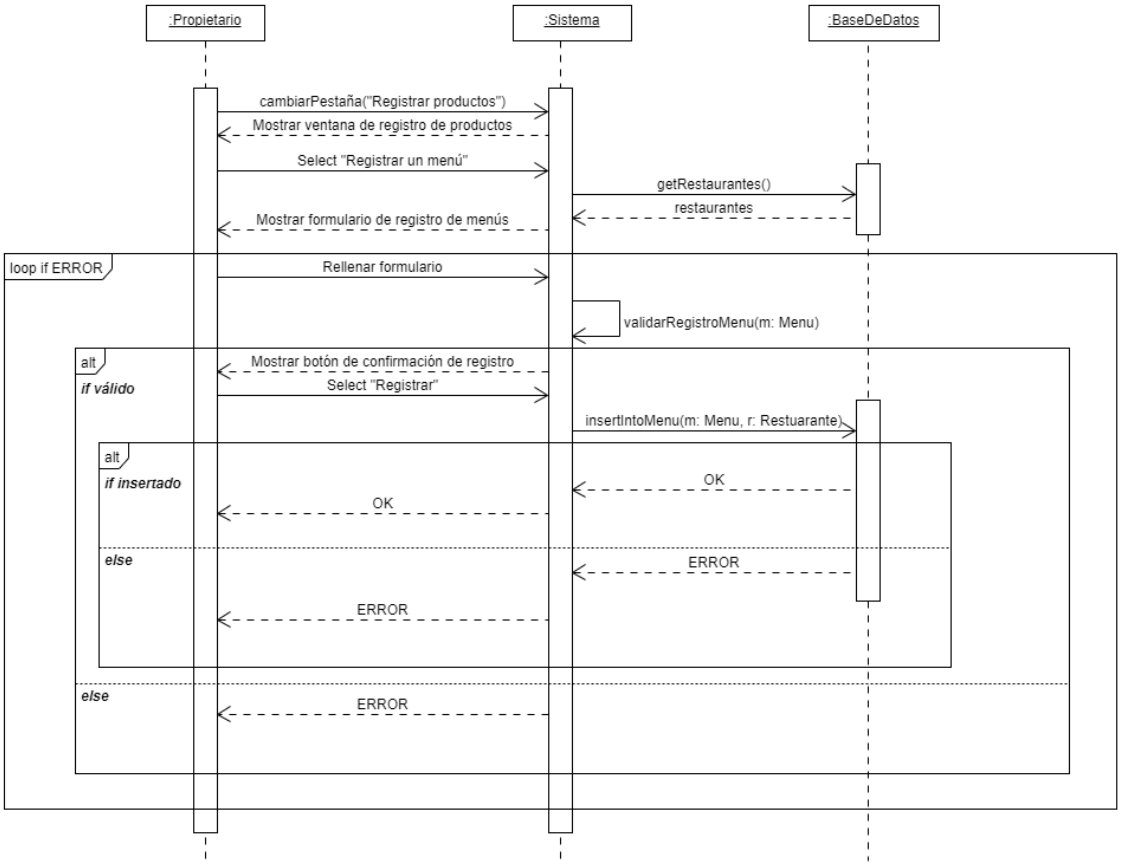
3.2. Diagrama de Secuencia Caso 5

Este caso de uso modela el registro de un producto (local o externo) por parte de un usuario con rol “Propietario de Restaurantes”.



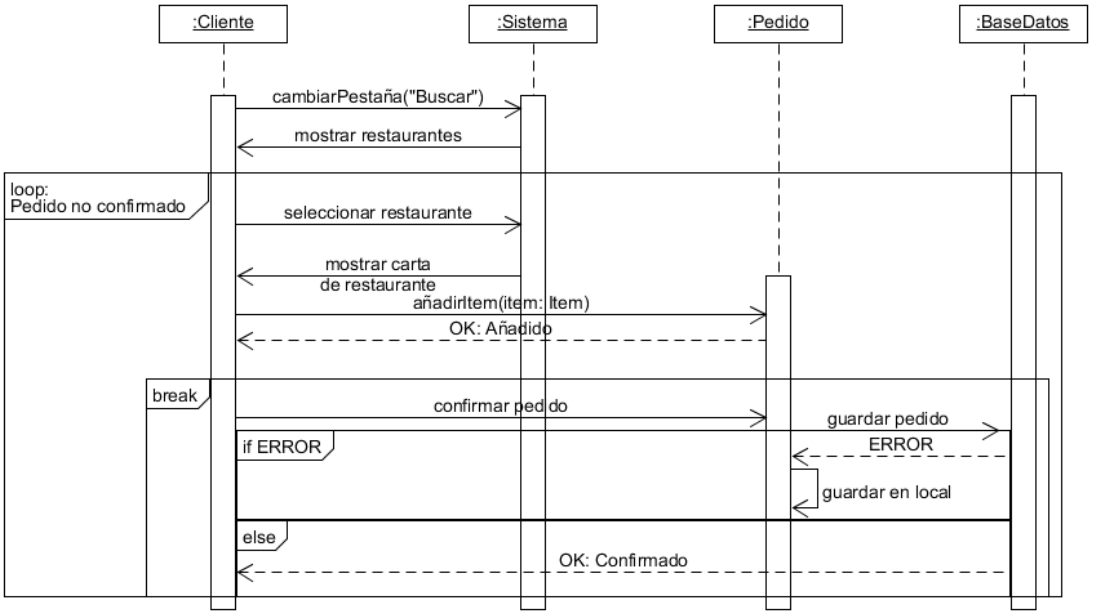
3.3. Diagrama de Secuencia Caso 6

Este caso de uso modela el registro de un menú por parte de un usuario con rol "Propietario de Restaurantes".



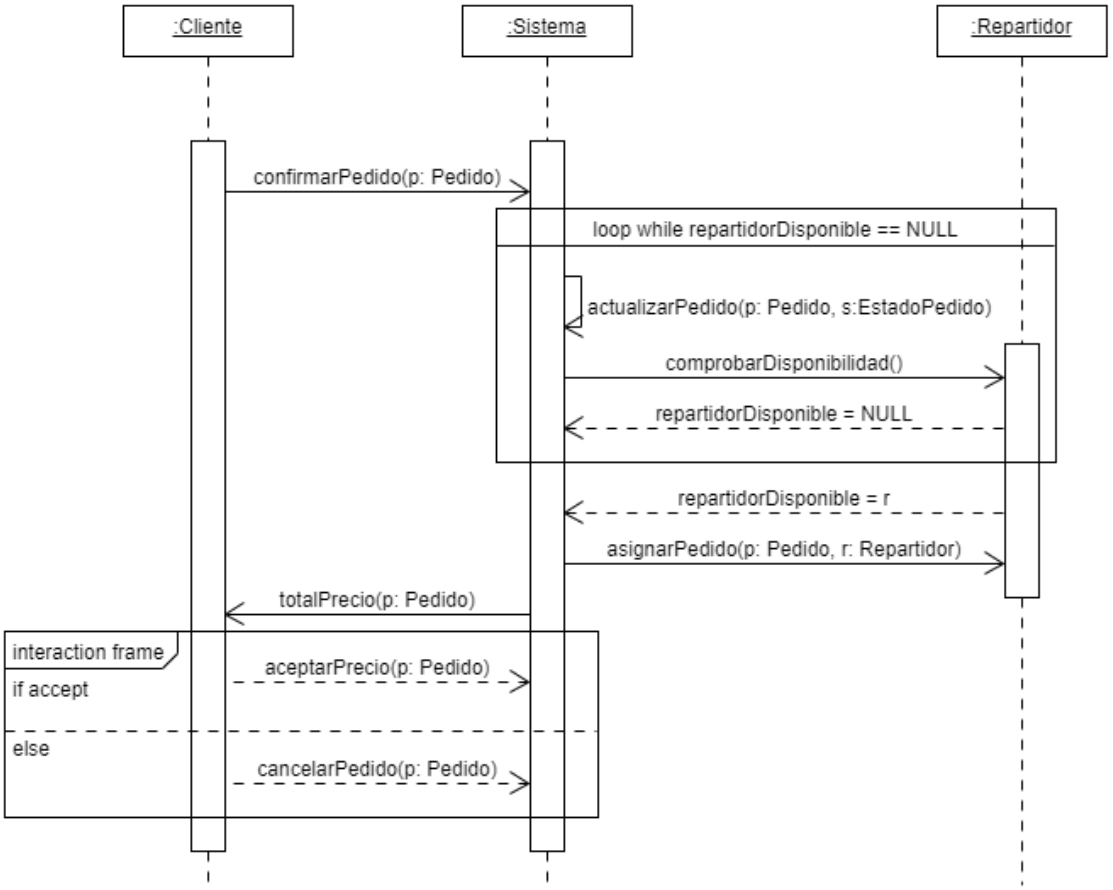
3.4. Diagrama de Secuencia Caso 10

Este caso de uso modela la realización de un pedido por parte de un usuario con rol "Cliente del Servicio".



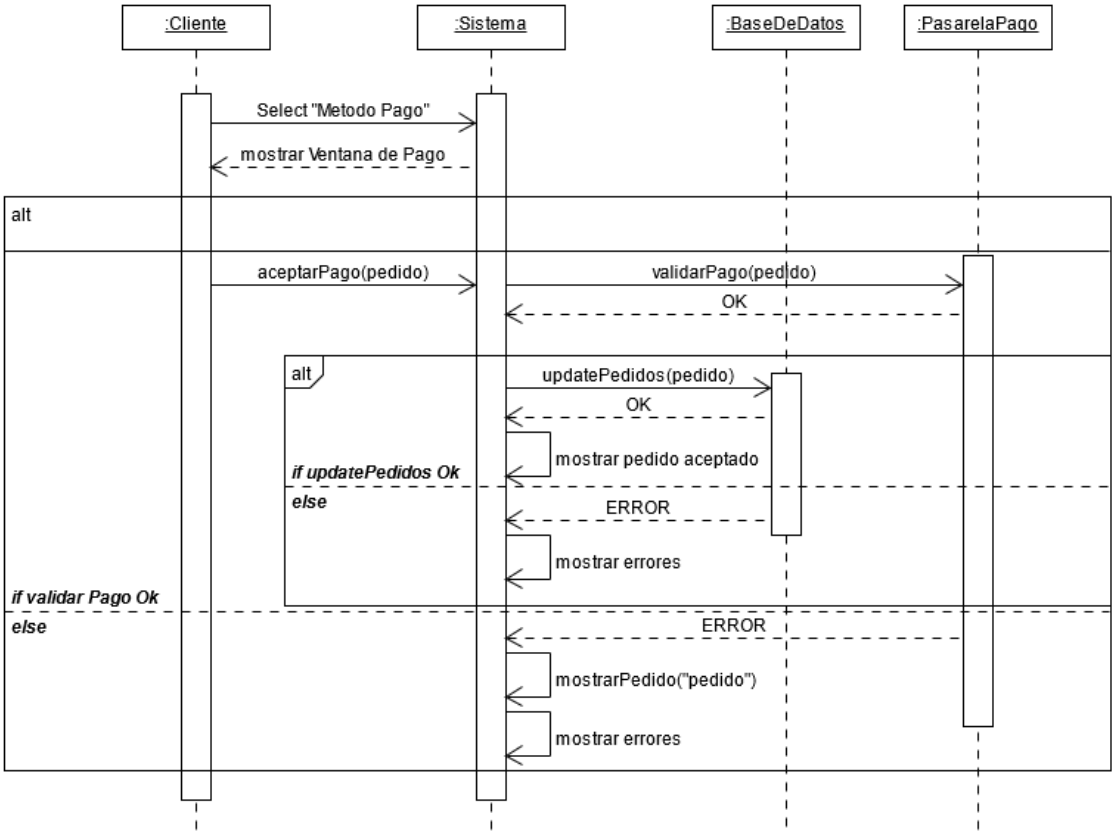
3.5. Diagrama de Secuencia Caso 12

Este caso de uso modela la aceptación de un pedido por parte de un usuario con rol "Cliente del Servicio".



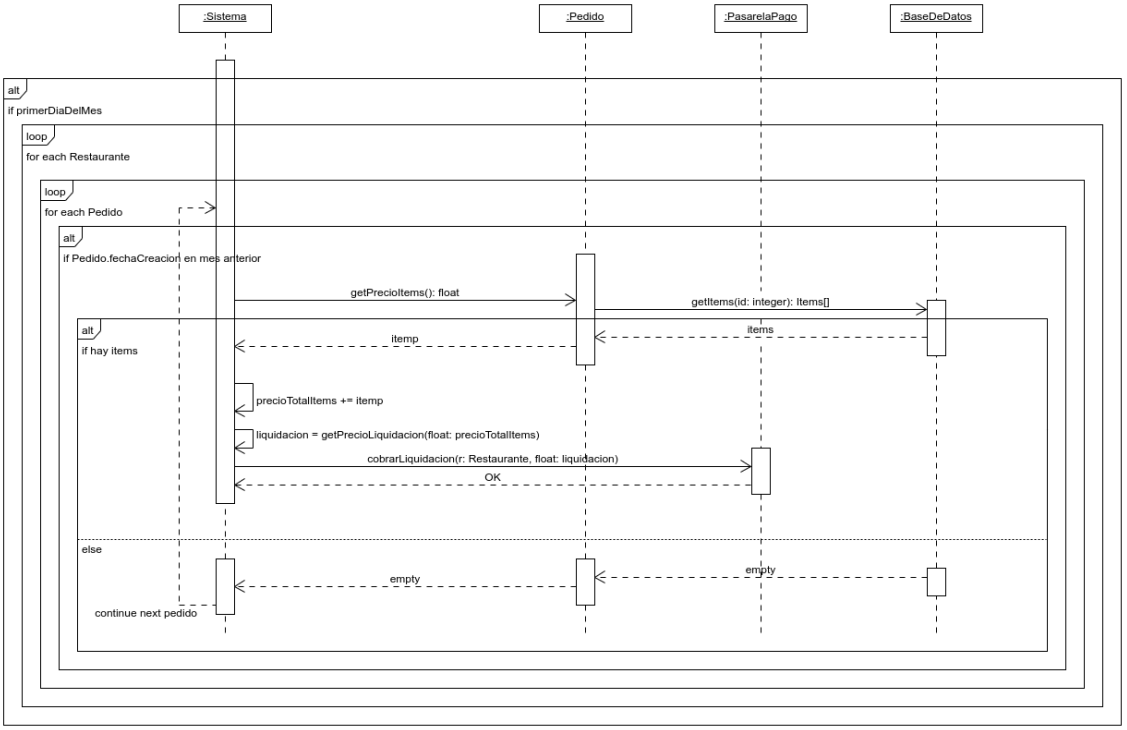
3.6. Diagrama de Secuencia Caso 13

Este caso de uso modela el pago de un pedido por parte de un usuario con rol “Cliente del Servicio”.



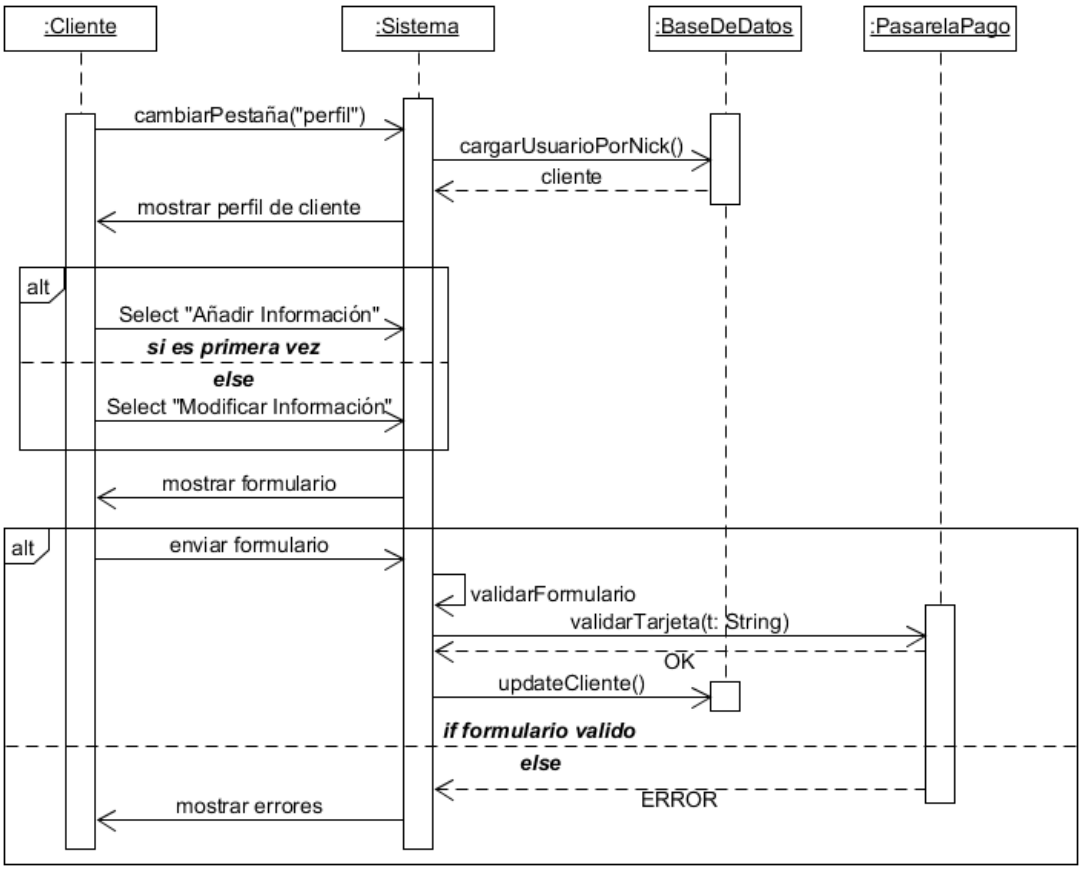
3.7. Diagrama de Secuencia Caso 14

Este caso de uso modela la liquidación de un restaurante por parte del sistema.



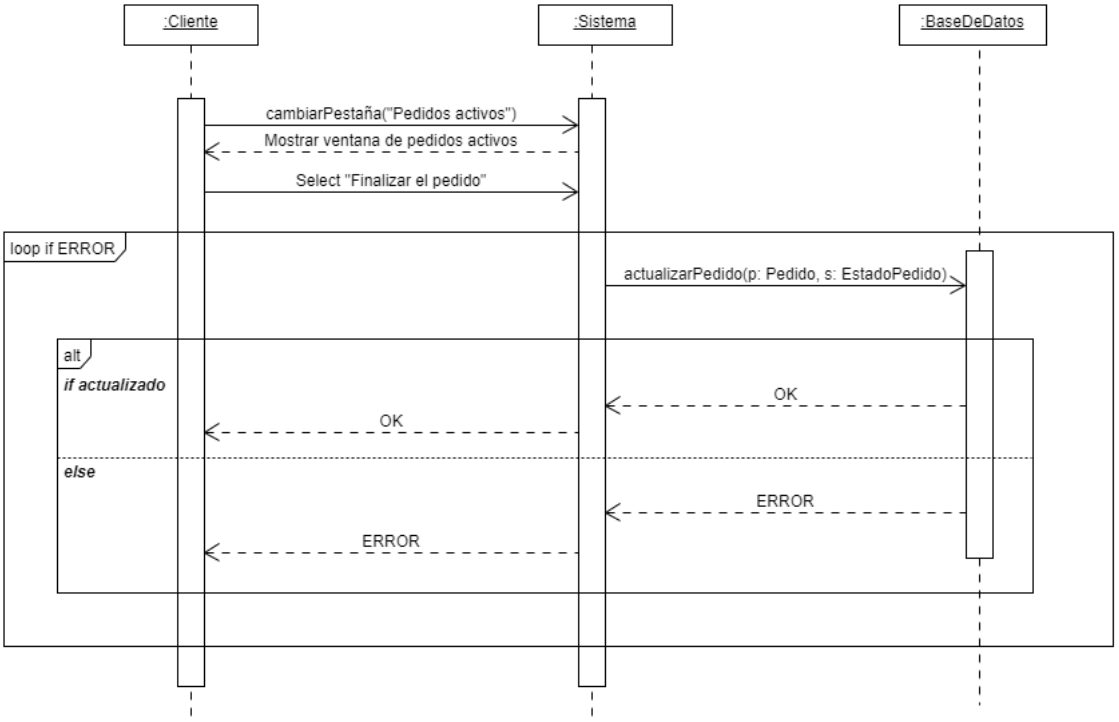
3.8. Diagrama de Secuencia Caso 16

Este caso de uso modela la introducción de los datos del cliente por parte de un usuario con rol “Cliente del Servicio”.



3.9. Diagrama de Secuencia Caso 20

Este caso de uso modela el cerrado de un pedido por parte de un usuario con rol "Cliente del Servicio".



4. Glosario

TÉRMINO	DESCRIPCIÓN
BBDD	Base de datos.
Clase maestra	Clase, del diagrama de clases, que maneja la mayoría de la funcionalidad de la aplicación. Se da, sobre todo, cuando se desarrolla el patrón de diseño Modelo-Vista-Controlador, siendo ésta el controlador.
Clase representativa	Clase, del diagrama de clases, que no requiere de una implementación como tal a la hora de entrar en la fase de programación. Es una clase de carácter informativo.
Diagrama de secuencia	Diagrama en el que se representa la secuencia de acciones que se realizarán para completar un requisito.
Educción	Deducción.
Formato consistente	Se dice del formato de un objeto cuando es común para dos o más comunicadores. Es decir, si dos entornos diferentes son capaces de comunicarse mediante un objeto X, se dice que ese objeto X tiene un formato consistente (p.e. la base de datos maneja datos simples y la aplicación maneja datos complejos: se necesita una clase que transforme la información simple de la base de datos a un objeto complejo que entienda la aplicación).
Getter	Función que trata de devolver un atributo de una clase o de obtener algún dato de algún objeto.
Memoria volátil o memoria RAM	Memoria que no almacena su estado al sufrir una desconexión. Esto es, si una aplicación se cierra por completo o el ordenador que la ejecuta se apaga, la información recabada por dicha aplicación se pierde.