MEMORIA

Miembros del equipo:

Laura Pérez	Esther Ávila
Miguel Ascanio	Guillermo Romero
Javier Toledano	Daniel Tocino
Álvaro Miguel Rodríguez	David Quirce

Indice

1 Introducción

A lo largo de esta memoria se van a explicar y mostrar de forma sencilla los avances realizados durante el segundo cuatrimestre con el fin de finalizar el proyecto.

En primer lugar tenemos que hablar del diagrama de casos de uso, un mapa que nos ayudara a entender que puede hacer nuestra aplicación, quien lo puede hacer y los distintos módulos de trabajo descritos en el srs. Este diagrama por su gran tamaño lo hemos dividido por usuarios para su mejor comprensión.

Por otro lado y muy ligado a lo anterior tenemos un diagrama de clases, otro mapa que en este caso es de carácter más técnico, su fin no es otro que ayudar a la implementación de la aplicación, de forma que sea correcta, no se realicen partes del trabajo en doblado por culpa de no tener claro que vamos a hacer.

El cómo lo vamos a hacer es lo realmente complicado, mediante una serie de diagramas que se detallan en esta memoria hemos sido capaces de estructurar de forma correcta el proyecto, y sobre todo de documentar como funciona todo para que en posibles revisiones de la aplicación no sea necesario empezar por entender que se hizo la última vez.

También se han realizado una serie de pruebas y test al proyecto para poder comprobar si la aplicación funciona correctamente, a pesar de estar convencidos de la eficacia de las pruebas realizadas hay que destacar que las labores de test y mejora de una aplicación nunca terminan.

Por último destacar que durante este segundo cuatrimestre las labores de organización y gestión del proyecto no han sido nada fáciles, sobre todo por los riesgos con los que nos hemos ido encontrando, esto está más detallado en el punto 11 Gestión del proyecto.

2 Diagrama de casos de uso

A continuación se muestran los diagramas de casos de uso. Han sido divididos por Actores, para que quepan en el documento, en el RSA se puede ver la versión completa.

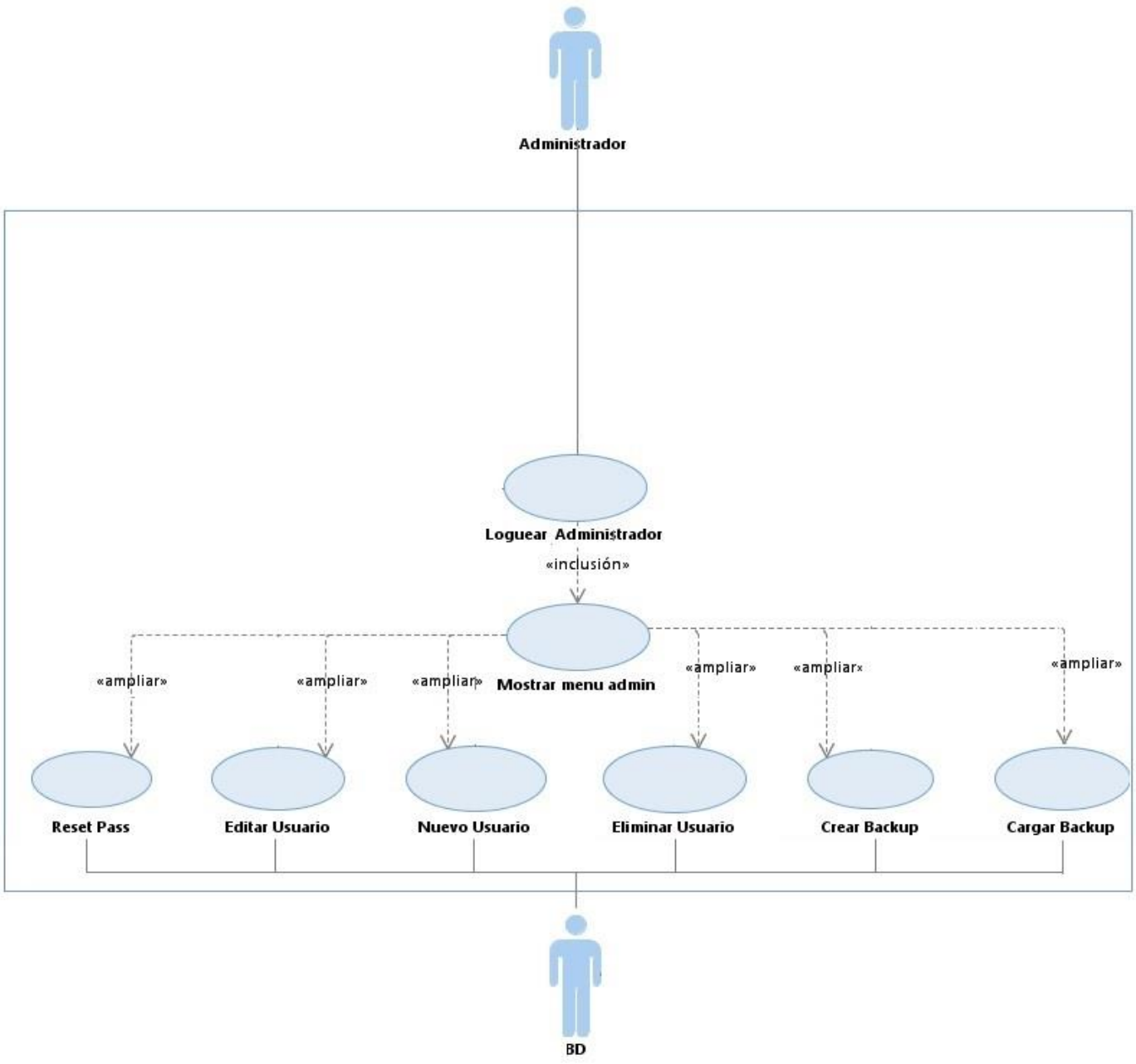
Se han corregido errores previos, como el no mostrar el límite del sistema.

Aunque no sea del todo correcto, para facilitar su comprensión, se han añadido casos de uso como "Mostrar menú".

2.1 CU Administrador

El primer actor al que hacemos referencia es el Administrador,el cual deberá de entrar en su sesión, para poder realizar las actividades que le corresponden. Estas actividades aparecerán en su menú, "Menu Admin", que son las actividades que la persona podrá desempañar en relación al cargo que ostenta en la empresa.

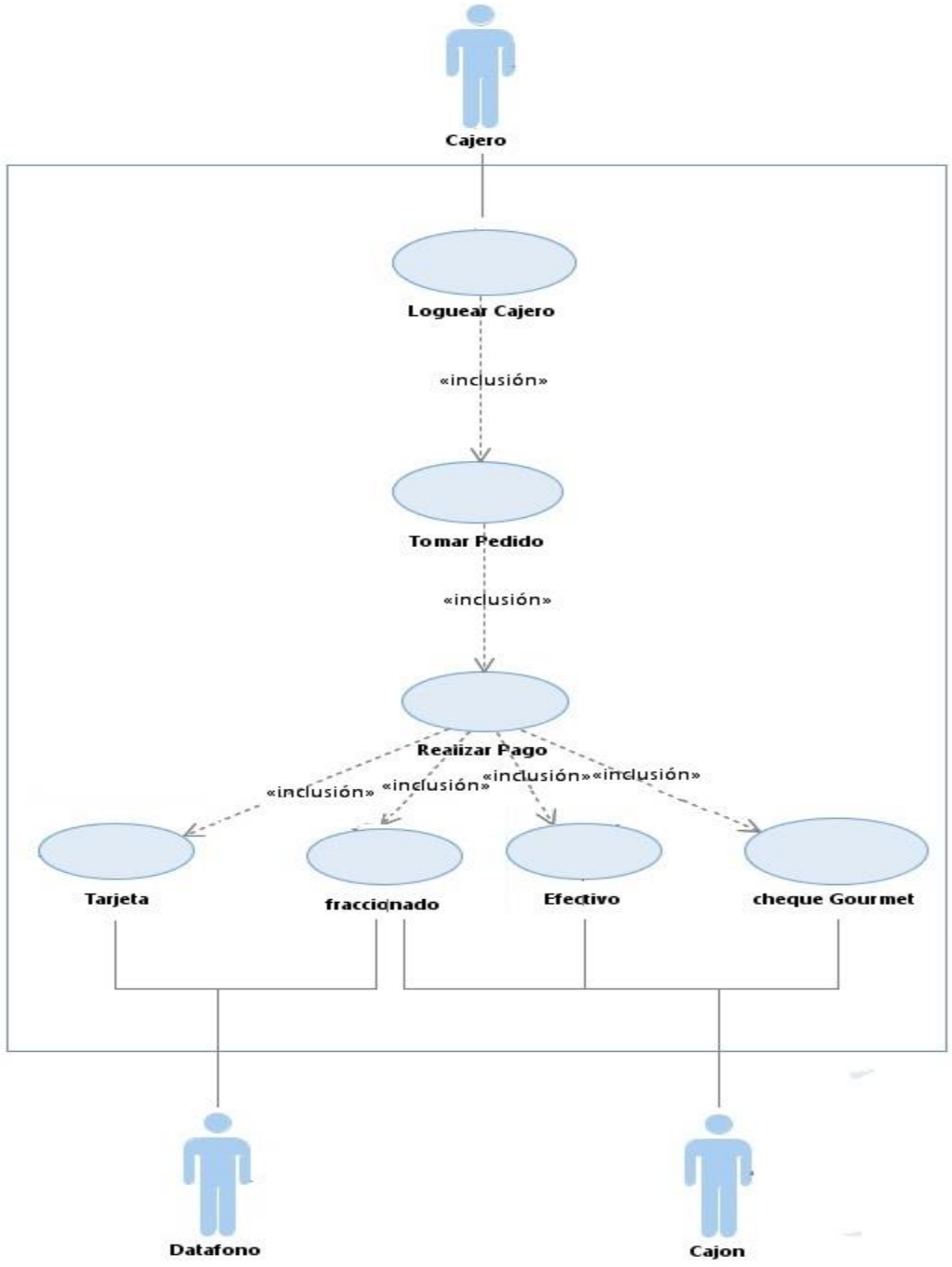
A lo largo de este proceso pueden ocurrir una serie de errores, como equivocarse de cargo a la hora de loguearse o introducir una id o contraseñas no válidas o intentar comenzar con una actividad por equivocación. La solución que hemos dado a estos problemas, es que el usuario cuando vaya a utilizar el programa siempre tenga la opción de volver para atrás y que su id o contraseñas se pueden introducir numerables veces hasta que sean correctas.



2.2 CU Cajero

El siguiente actor al que hacemos referencia es el Cajero. El cual deberá de entrar en su sesiónpara poder realizar los pedidos de los clientes en la caja.

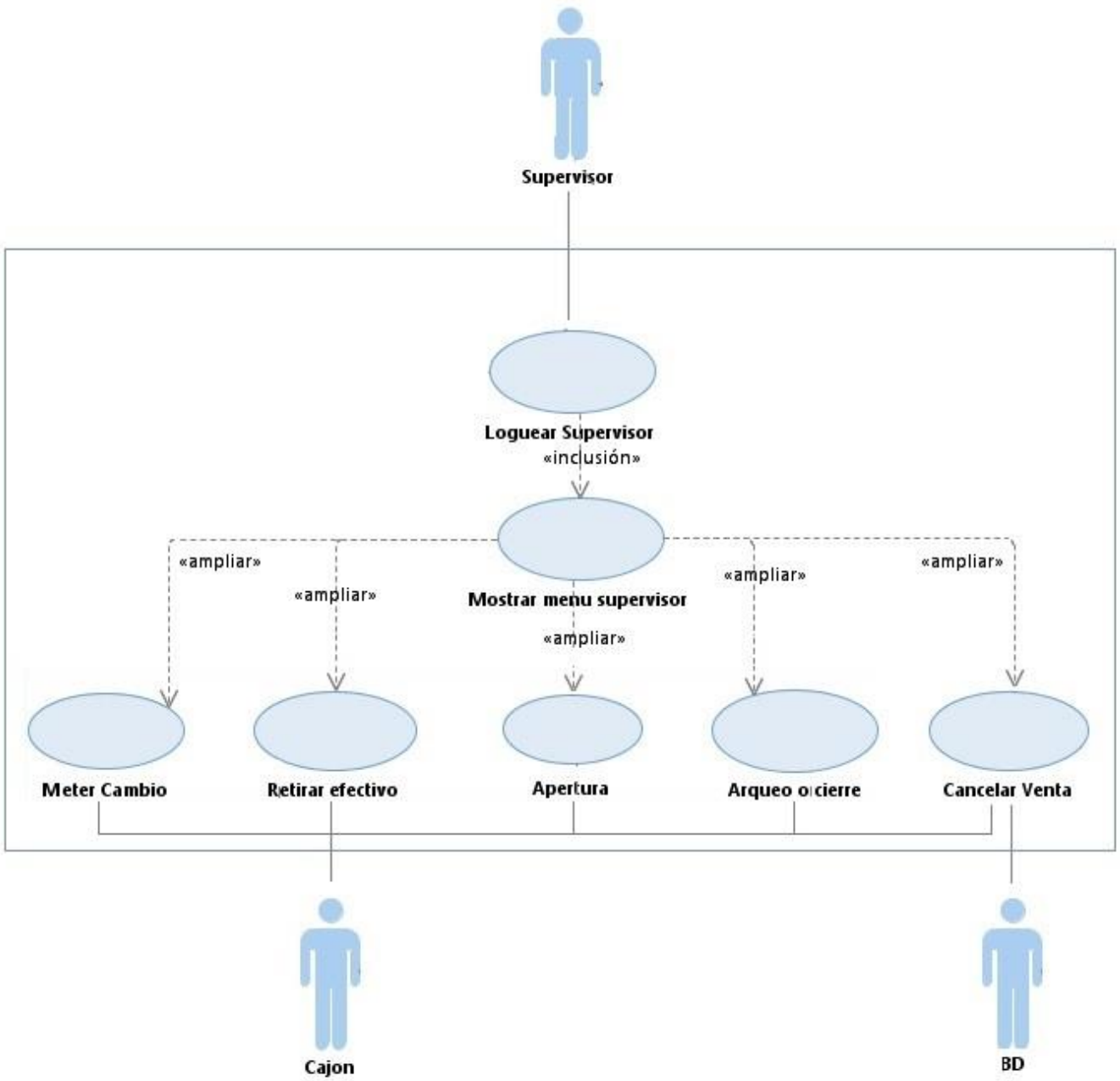
En este proceso también pueden ocurrir una serie de problemas, como equivocarse de cargo a la hora de loguearse o introducir una id o contraseñas no válidas o intentar comenzar con una actividad por equivocación. La solución que hemos dado a estos problemas, es que el cajero cuando vaya a utilizar el programa siempre tenga la opción de volver para atrás y que su id o contraseñas se pueden introducir numerables veces hasta que sean correctas. Hemos tenido también en cuenta una serie de condiciones que se tienen que cumplir a la hora de proceder al pago, como que la cantidad que se abona sea mayor que la cantidad de la compra total y nunca menor, o también que cuando se quiera hacer un pago fraccionado siempre se utilice primero la tarjeta y luego ya el siguiente método de pago, para evitar menos errores.



2.3 CU Supervisor

El siguiente actor es el supervisor. El cual deberá de entrar en su sesiónpara poder realizar también los pedidos de los clientes en la caja, la apertura y cierre de está y la cancelación de pagos.

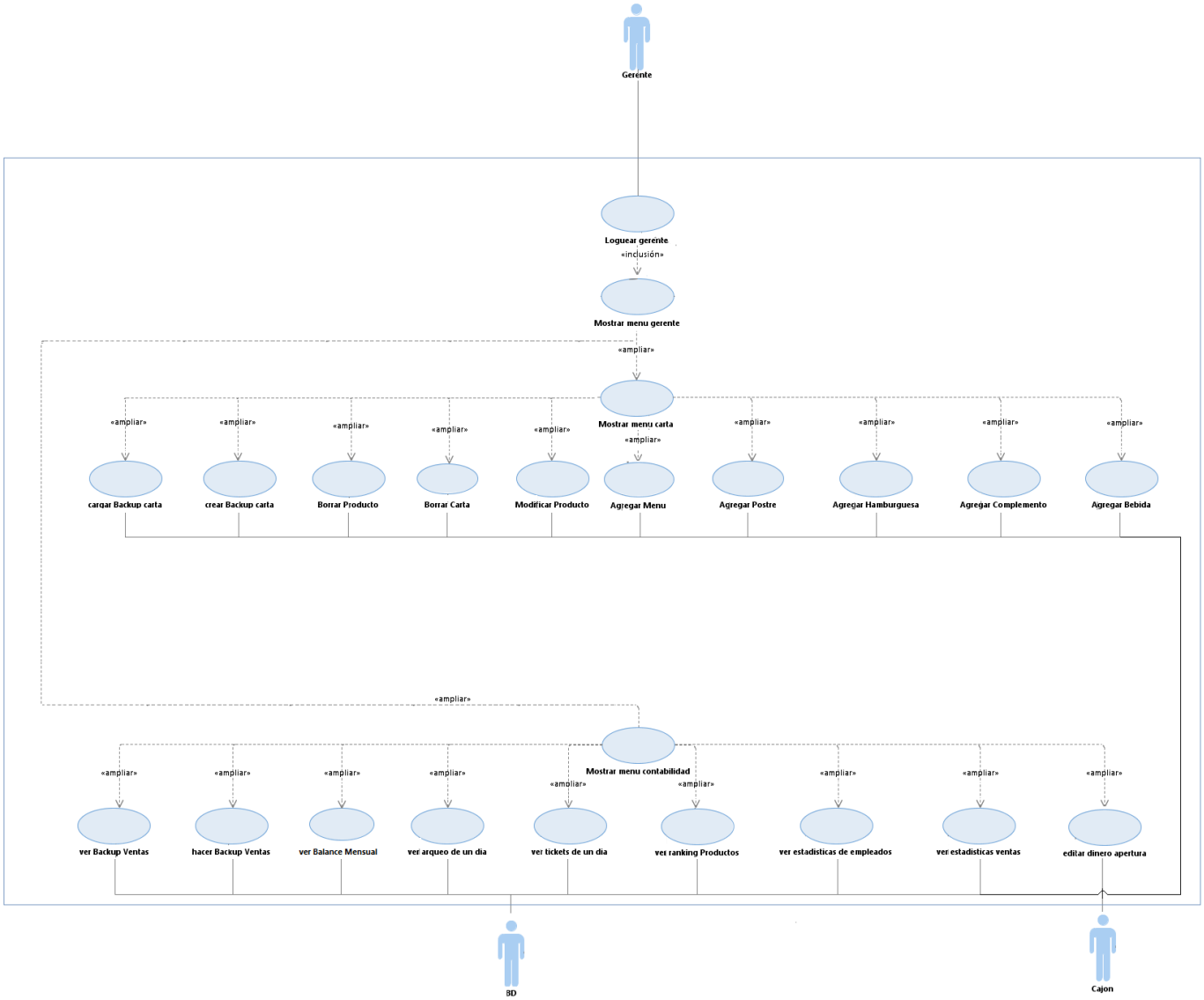
También pueden ocurrir una serie de problemas como en los anteriores casos de uso, equivocarse de cargo a la hora de loguearse, introducir una id o contraseñas no válidas, intentar comenzar con una actividad por equivocación. La solución siempre será tener la opción de volver para atrás y que su id o contraseñas se pueden introducir numerables veces hasta que sean correctas.



2.4 **CU Gerente**

El siguiente actor responsable de este CU, es el gerente. El cual deberá de entrar en su sesión para poder realizar sus actividades. Este actor tiene uno de los papeles más importantes en la tienda pues es el responsablede la dirección, control y toma de decisiones en el funcionamiento operativo, administrativo y comercial de la tienda. A su vez puede desempeñar las labores de cajero y supervisor.

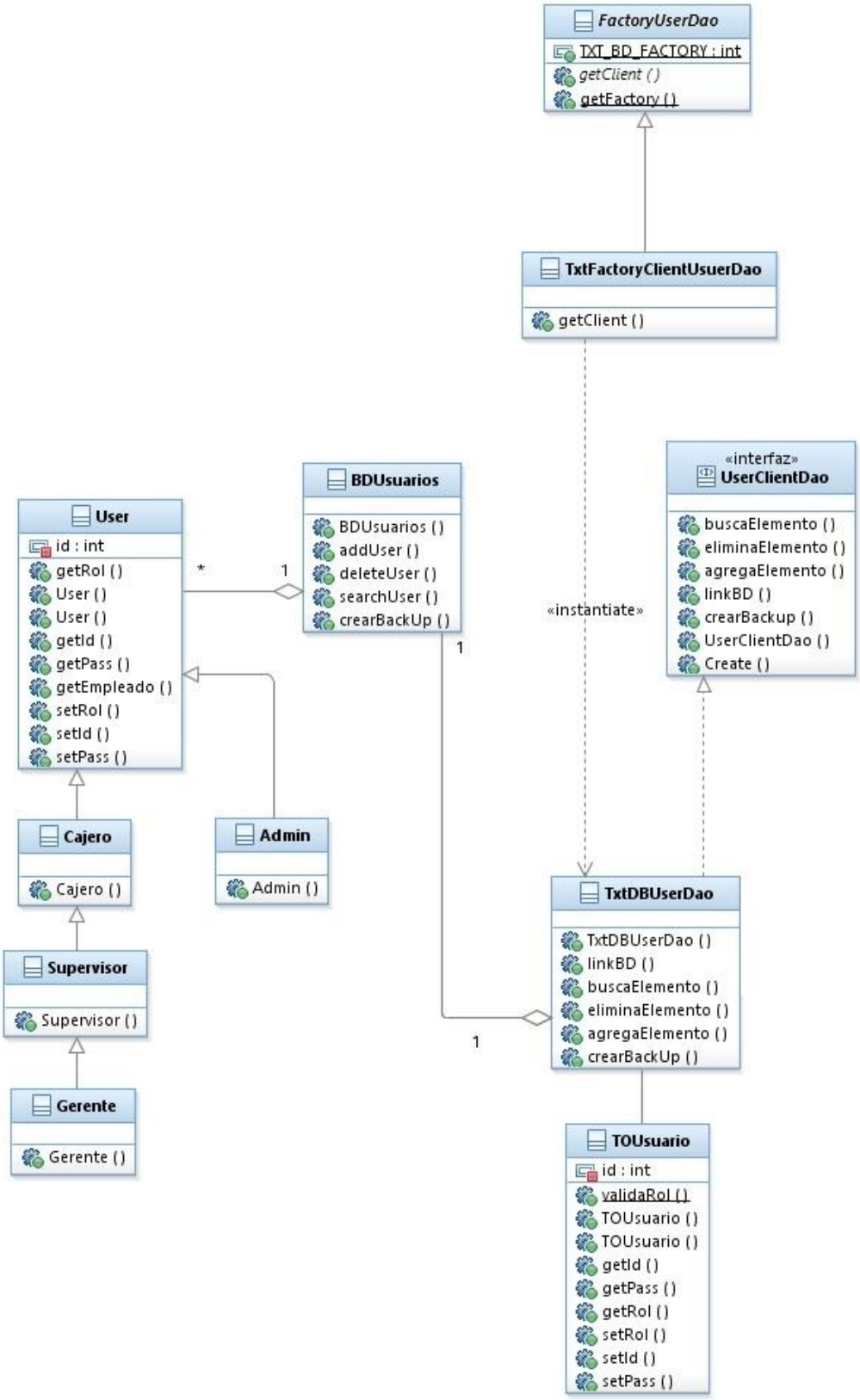
Al desempeñar tantas labores y actividades también pueden ocurrir como en los anteriores casos de uso expuestos, que se equivoque a la hora de loguearse, introduzca una contraseña o id no válidas, o equivocarse a la hora de elegir una actividad. La solución siempre será tener la opción de volver para atrás y que su id o contraseñas se pueden introducir numerables veces hasta que sean correctas.



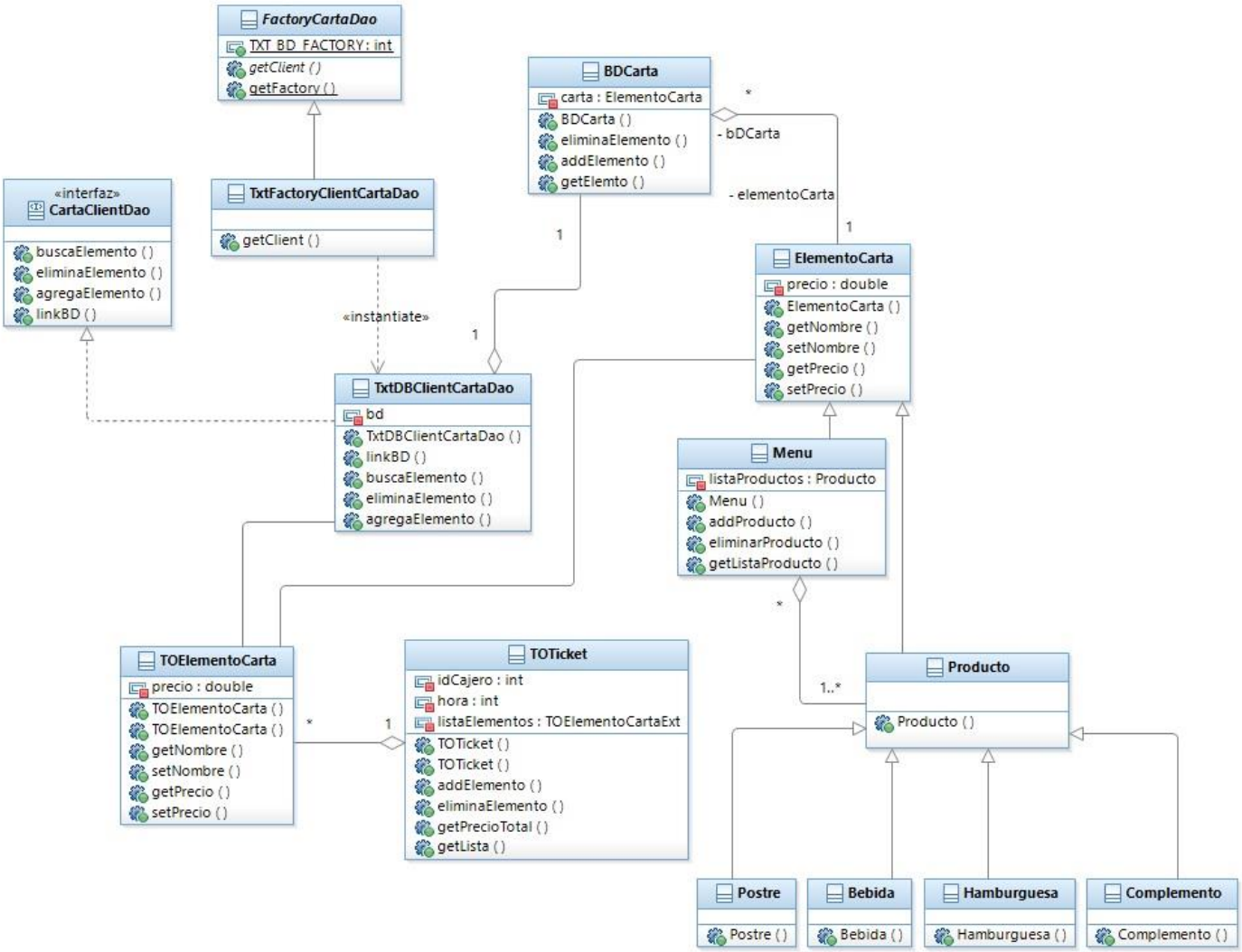
3 Diagrama de Clases

A continuación se mostrará el diagrama de clases. Al ser muy grande, en este documento se muestra particionado en paquetes lógicos, la versión completa se encuentra en el proyecto de RSA. Cabe destacar que tiene grandes diferencias con el de entregas anteriores, especialmente por el modelo de Tres Capas, aunque también porque a la hora de implementar nos dimos cuenta que algunas cosas no eran posibles implementarlas como se indicó, o bien porque consideramos que había mejores formas de hacerlo. En los puntos sucesivos del proyecto se seguirá haciendo referencia a este cambio; así como en el punto 6 se describirán con más detalle las clases.

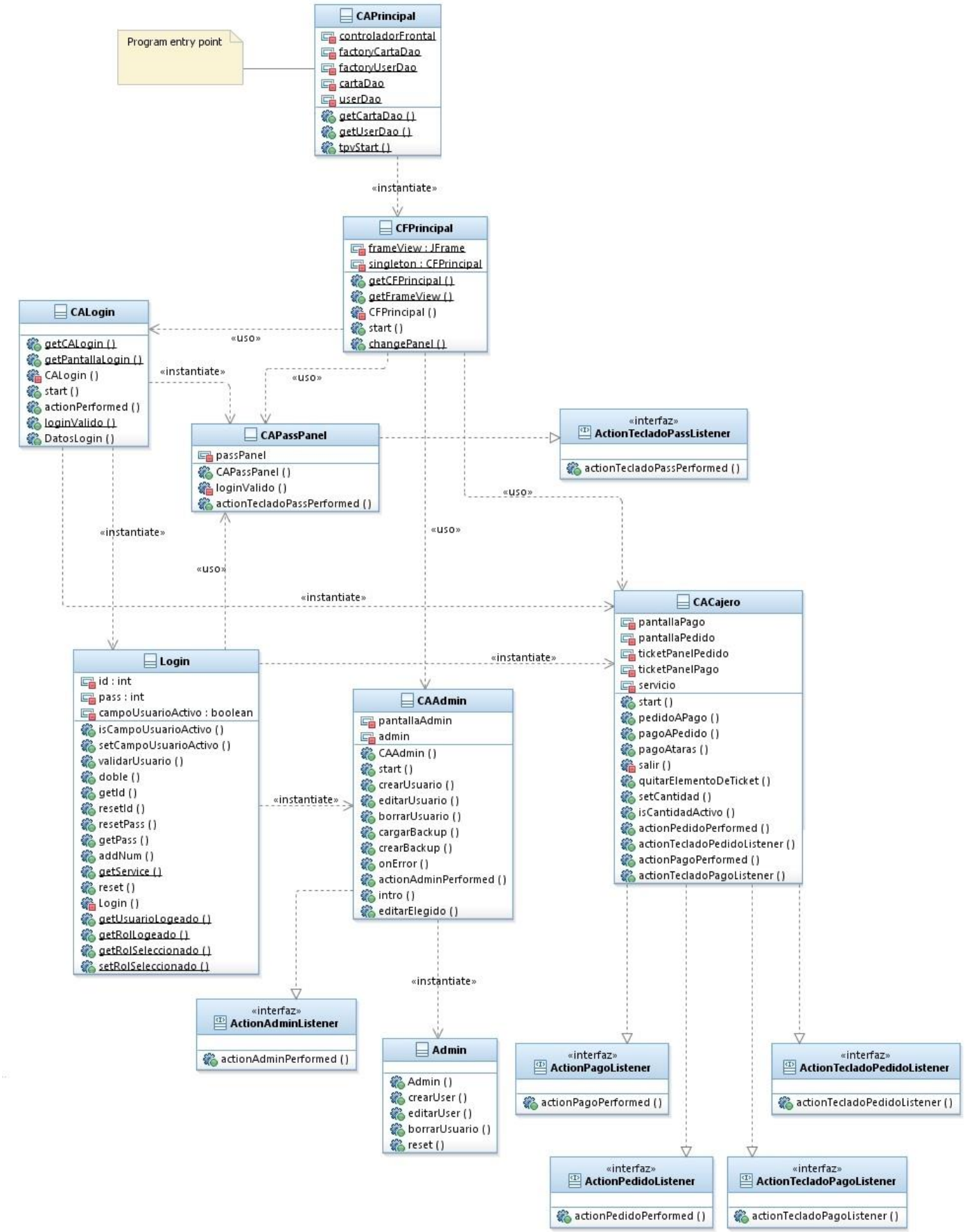
3.1 Usuarios



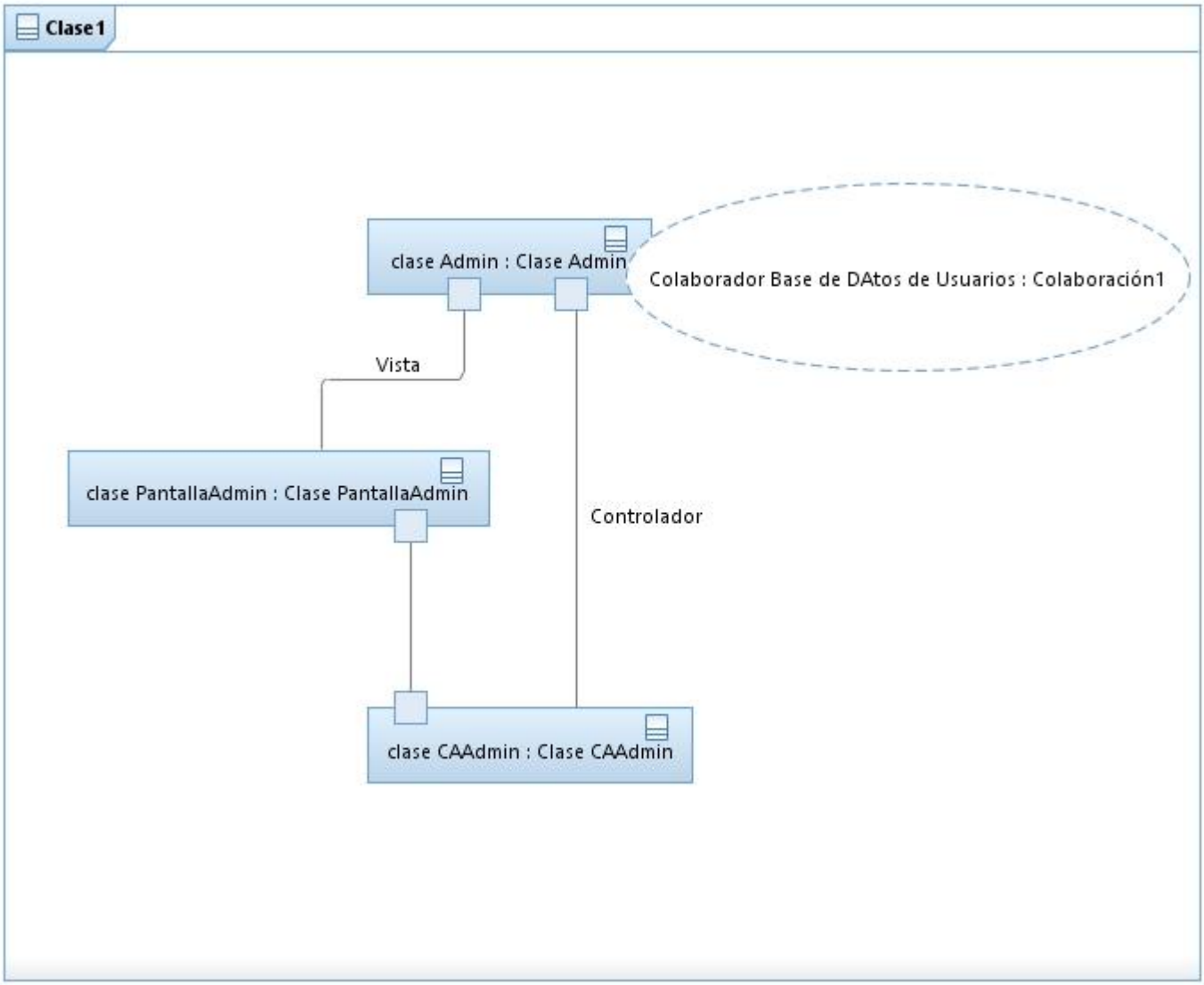
3.2 **Carta**


FactoryCartaDao
TXT_BD_FACTORY: int
getClient ()
getFactory ()
BDCarta
carta : ElementoCarta
BDCarta ()
eliminaElemento ()
addElemento ()
getElemto ()
- bDCarta
- elementoCarta
«interfaz»
CartaClientDao
buscaElemento ()
eliminaElemento ()
agregaElemento ()
linkBD ()
TxtFactoryClientCartaDao
getClient ()
«instantiate»
ElementoCarta
precio : double
ElementoCarta ()
getNombre ()
setNombre ()
getPrecio ()
setPrecio ()
TxtDBClientCartaDao
bd
TxtDBClientCartaDao ()
linkBD ()
buscaElemento ()
eliminaElemento ()
agregaElemento ()
Menu
listaProductos : Producto
Menu ()
addProducto ()
eliminarProducto ()
getListaProducto ()
TOElementoCarta
precio : double
TOElementoCarta ()
TOElementoCarta ()
getNombre ()
setNombre ()
getPrecio ()
setPrecio ()
TOTicket
idCajero : int
hora : int
listaElementos : TOElementoCartaExt
TOTicket ()
TOTicket ()
addElemento ()
eliminaElemento ()
getPrecioTotal ()
getLista ()
Producto
Producto ()
Postre
Postre ()
Bebida
Bebida ()
Hamburguesa
Hamburguesa ()
Complemento
Complemento ()

3.3 Servicio y controladores



4 Diagrama de estructura compuesta

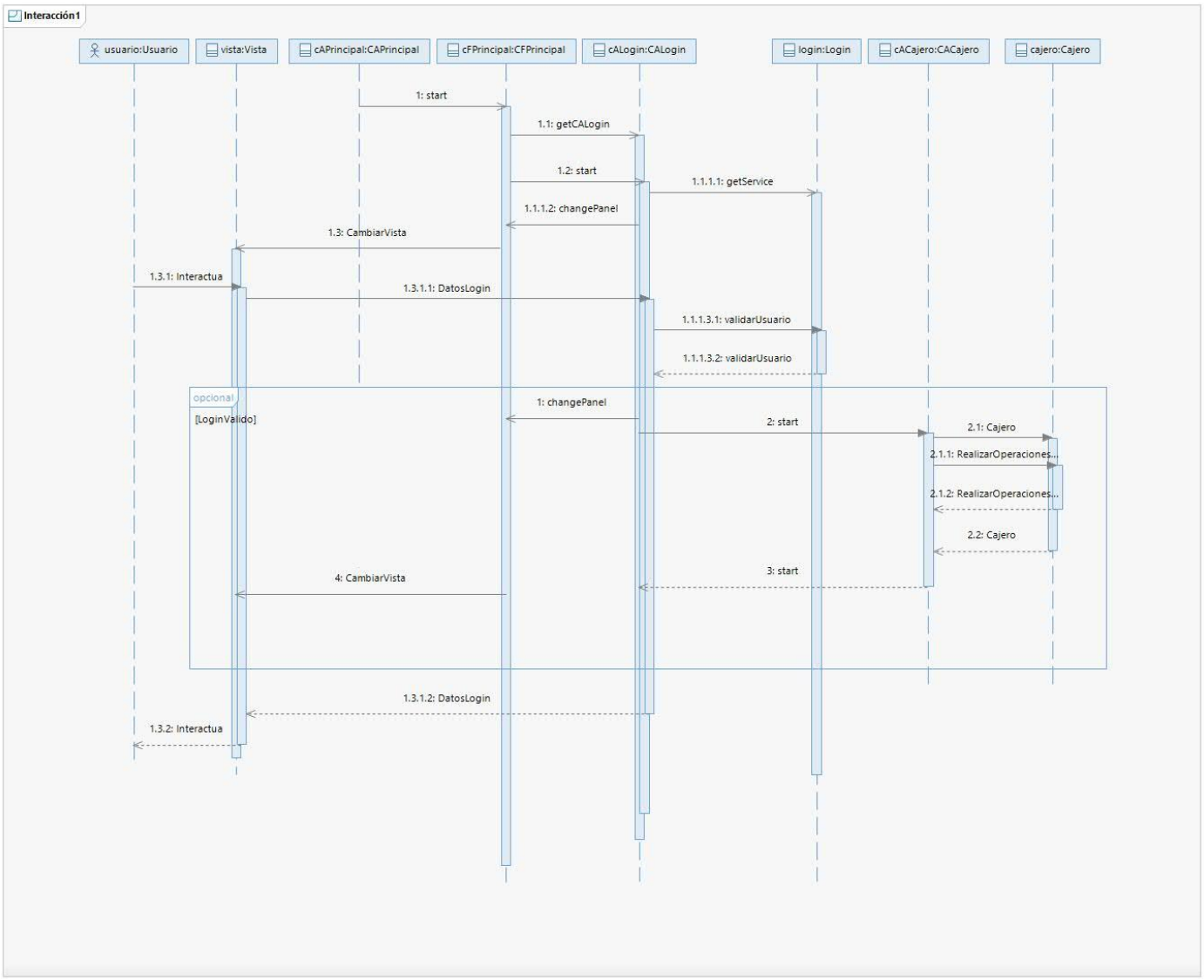


En este diagrama se puede observar la estructura del sistema del administrador, en el que la clase Admin representa al modelo y tiene una colaboración con la base de datos de los usuarios.
Tambien se observa como la clase PantallaAdmin es la vista del modelo, y la clase CAAdmin es su controlador. Tambien se puede ver como las clases PantallaAdmin y CAAdmin se relacionan entre sí.

5 Diagramas de secuencia

A continuación se presentan los diagramas de secuencia elegidos. Hay algunos más incluidos en el proyecto RSA, pero estos son diagramas que tienen que ver con puntos posteriores del documento si se incluyen en los mismos.

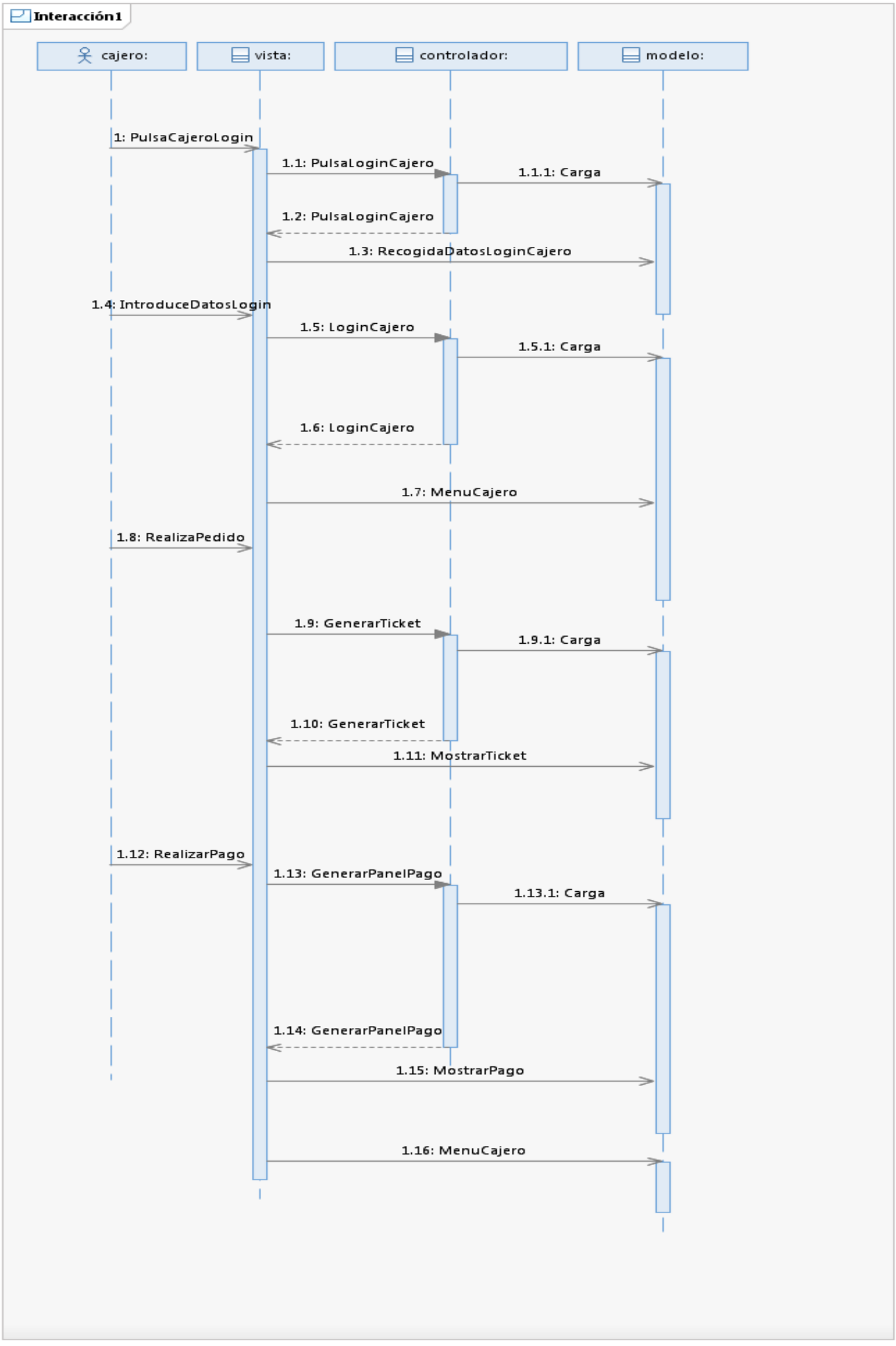
5.1 Login



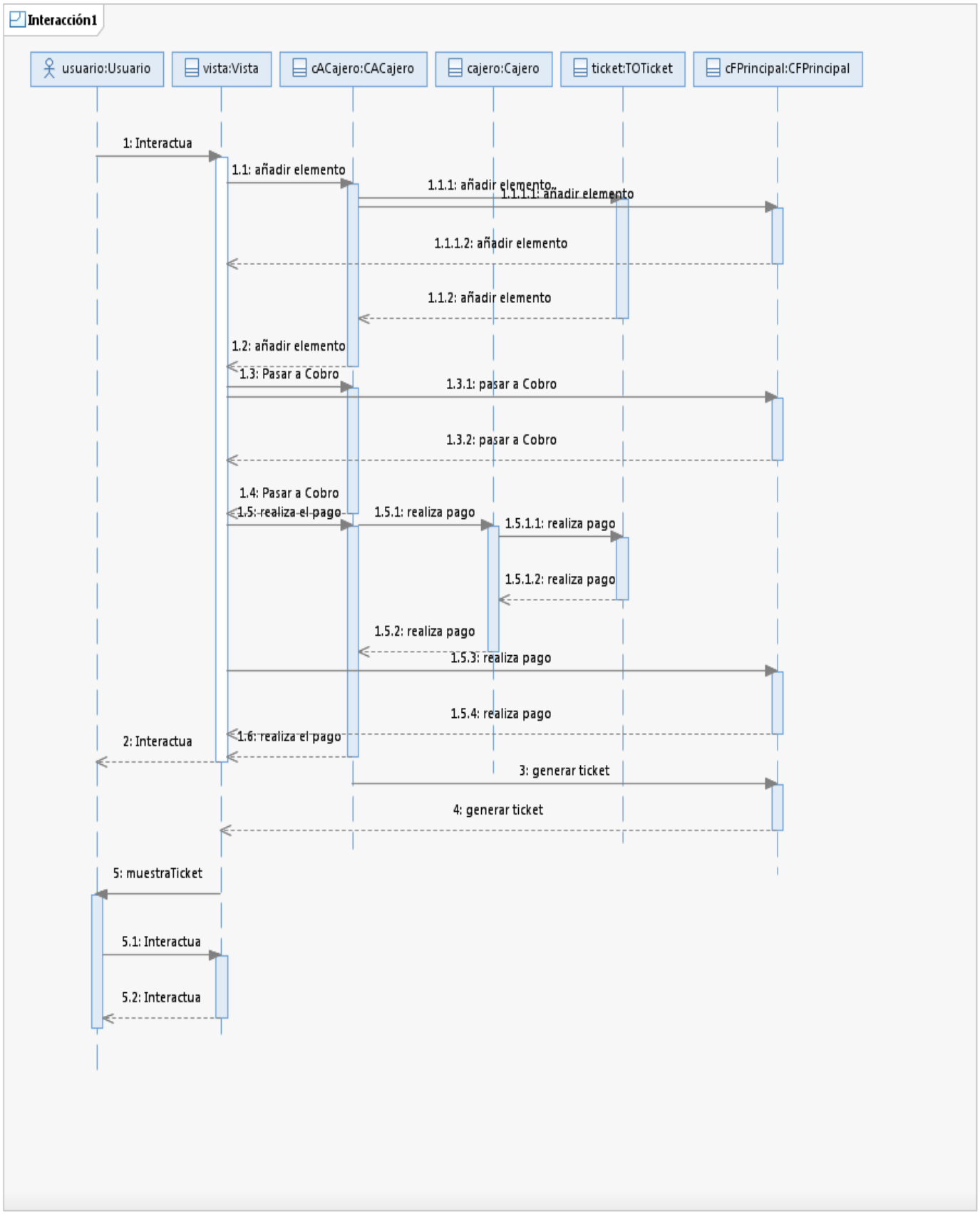
En este diagrama se ve de forma clara la interacción entre clases mediante el proceso de Login, cabe destacar que este diagrama ha sido de los últimos en realizarse para ayudarnos a cambiar del MVC al modelo multicapa.

5.1 Pago y pedido

En este diagrama basado en MVC tenemos una representación de cómo sería la interacción entre clases desde el logueo hasta el momento de terminar el pago.

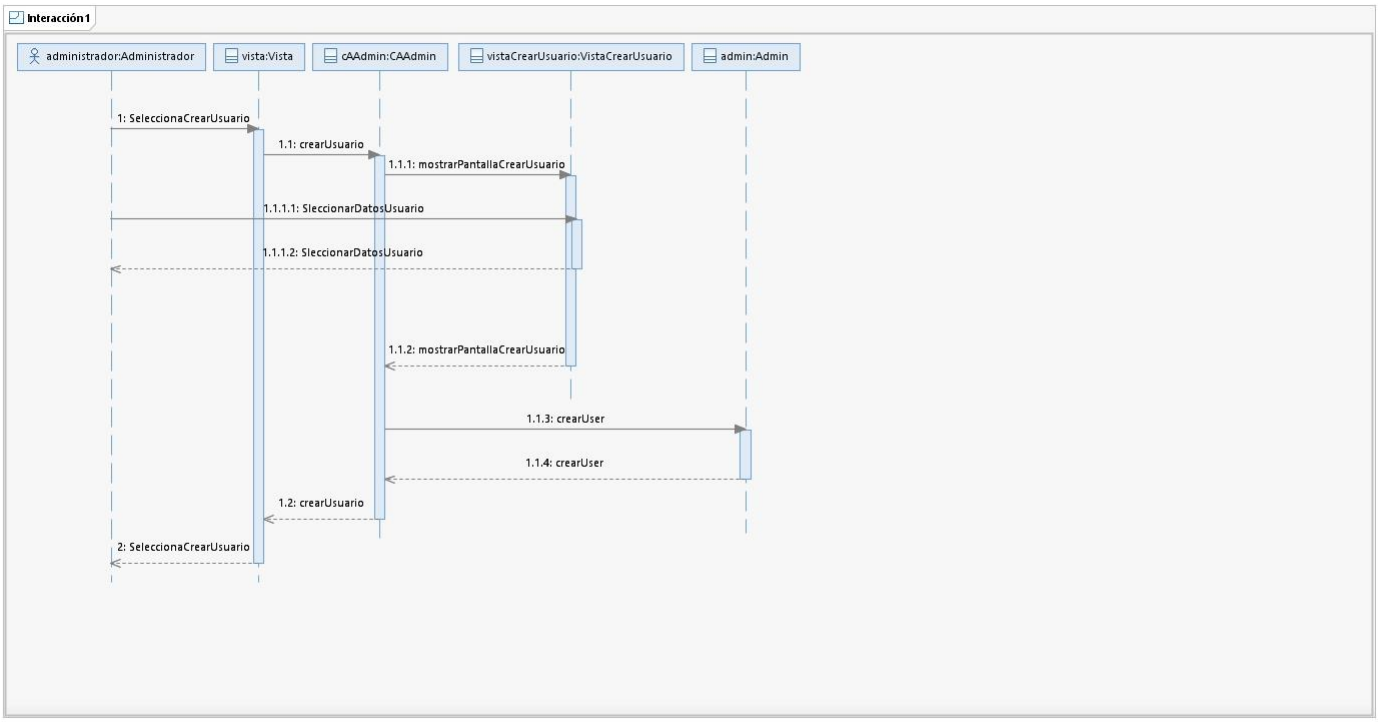

Interacción1
cajero:
vista:
controlador:
modelo:
1: PulsaCajeroLogin
1.1: PulsaLoginCajero
1.1.1: Carga
1.2: PulsaLoginCajero
1.3: RecogidaDatosLoginCajero
1.4: IntroduceDatosLogin
1.5: LoginCajero
1.5.1: Carga
1.6: LoginCajero
1.7: MenuCajero
1.8: RealizaPedido
1.9: GenerarTicket
1.9.1: Carga
1.10: GenerarTicket
1.11: MostrarTicket
1.12: RealizarPago
1.13: GenerarPanelPago
1.13.1: Carga
1.14: GenerarPanelPago
1.15: MostrarPago
1.16: MenuCajero

Después de contemplar el modelo de tres capas, la acción del pago pasa por una serie de controladores que ayudan a la gestión del modelo multicapa como se puede ver a continuación, esto nos permite gestionar mejor la aplicación y como principal virtud cabe destacar que el actualizar el programa con esta nueva disposición es mucho más sencillo.

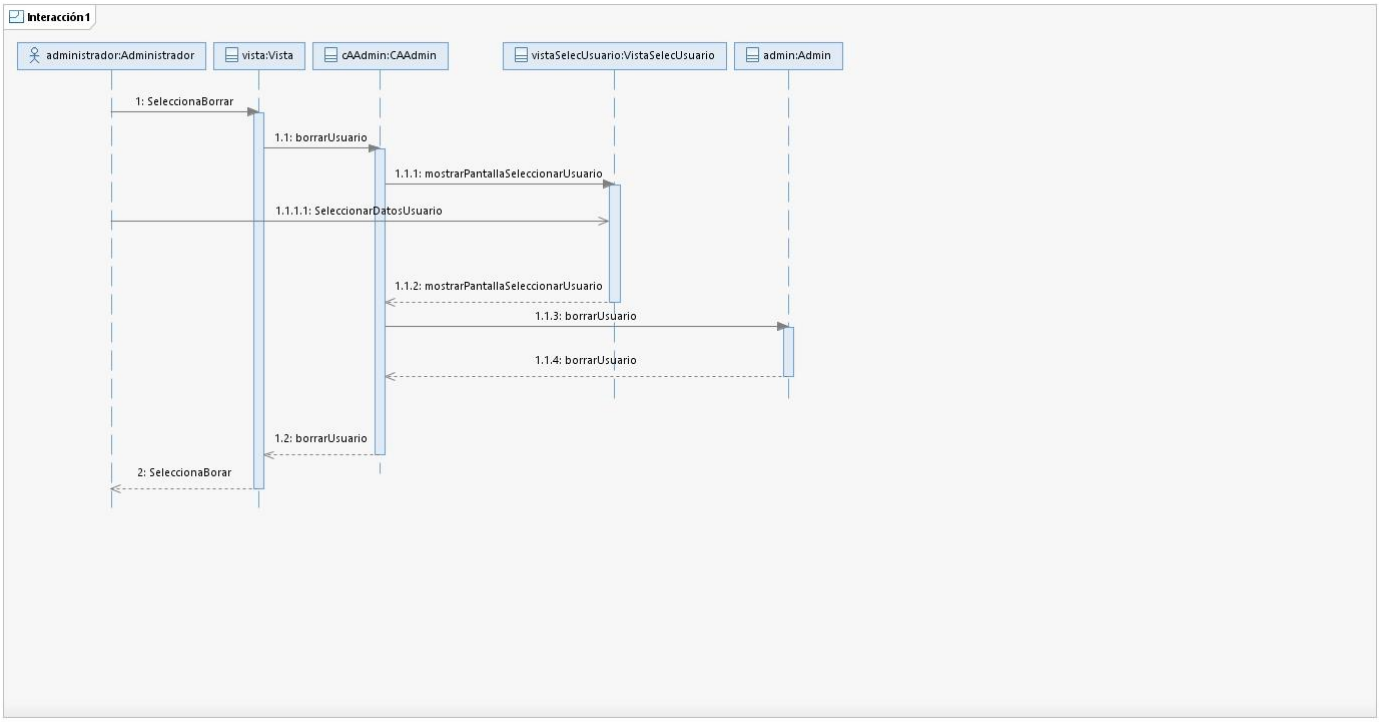


5.2 Administrador

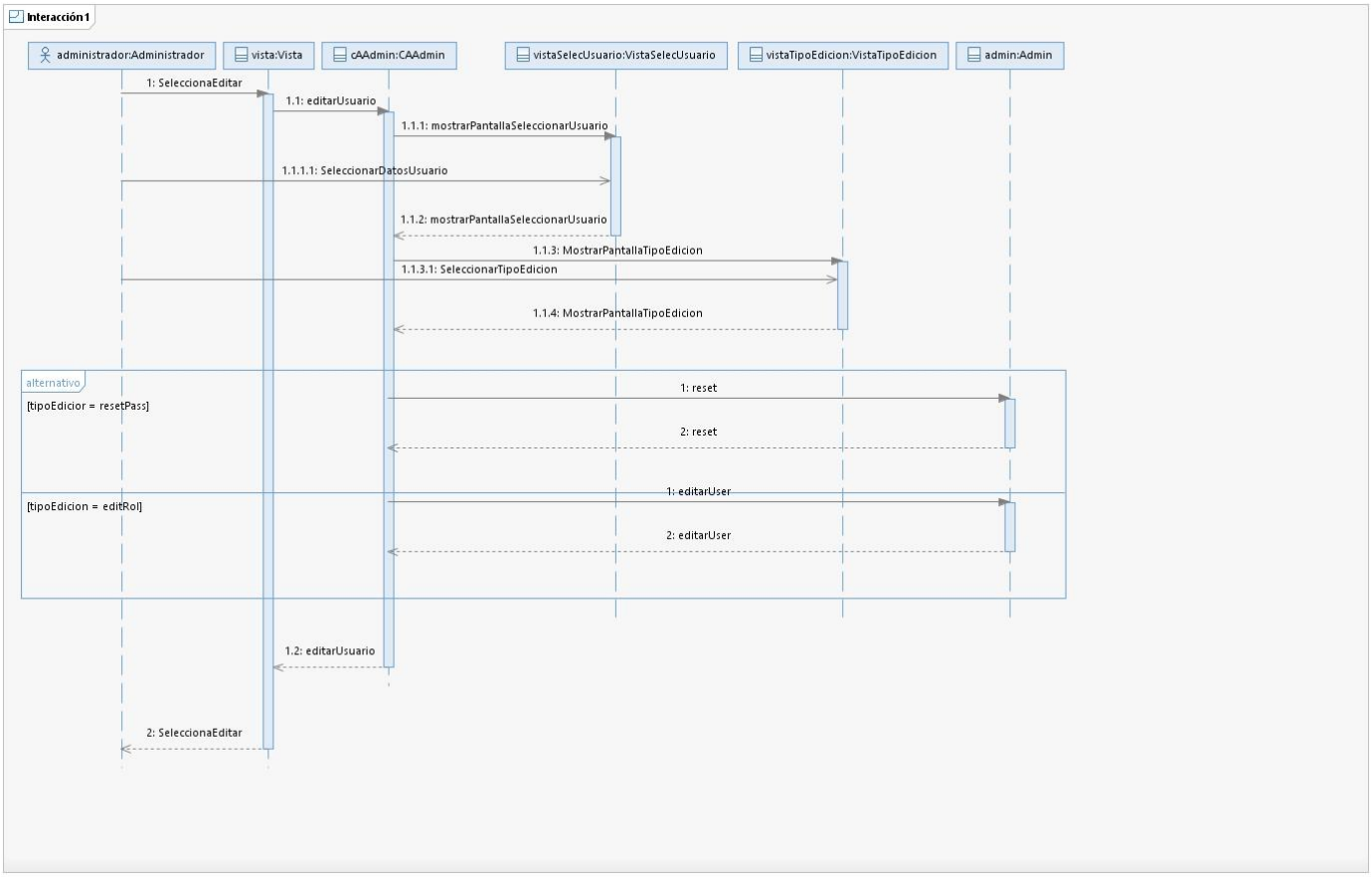
A continuación se encuentran los diagramas de secuencia asociados al administrador. Se han separado para aumentar su legibilidad.



Este primer caso es el de la creación de un usuario, el administrados pulsa en el botón de crear usuario y le saldrá una ventana emergente pidiéndole el identificador del usuario y su rol. Una vez seleccionados el controlador le manda la información al modelo que se encarga de crear el usuario.

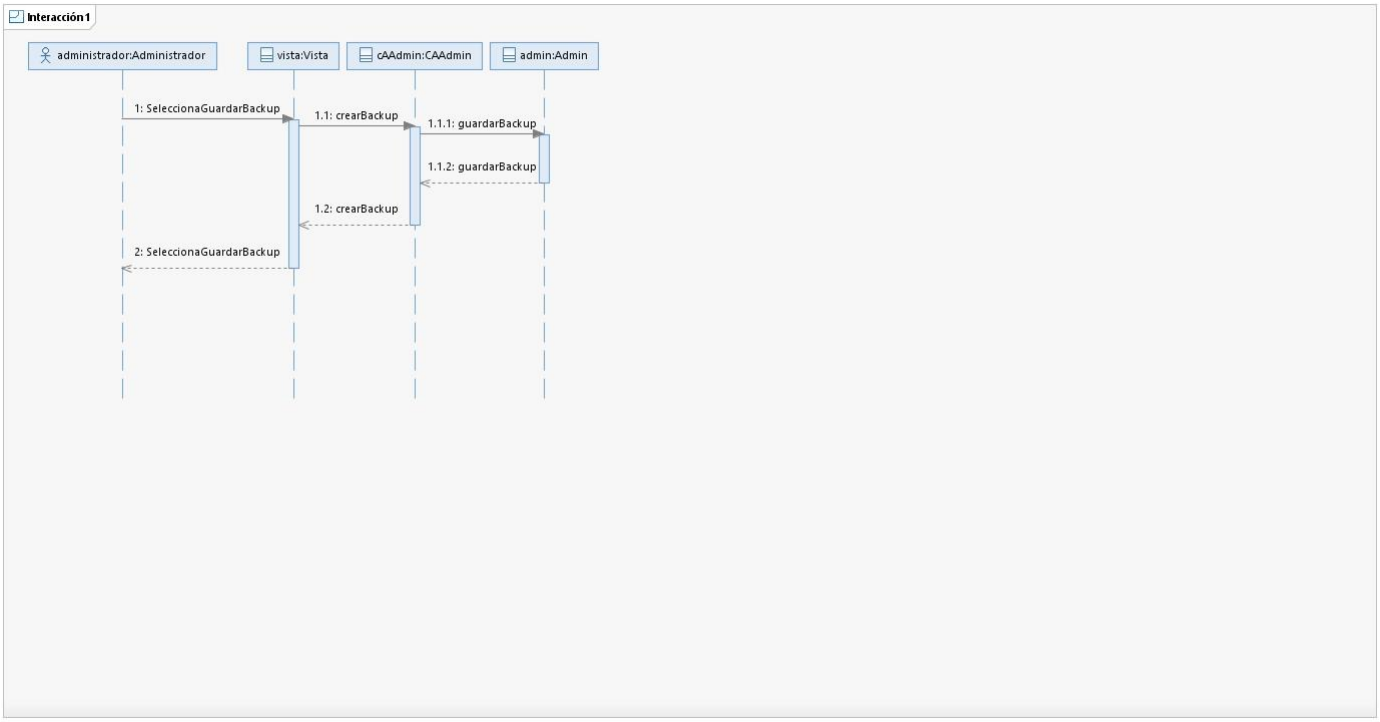


El anterior diagrama se corresponde con la eliminación de un usuario, el administrador pulsa sobre la opción de borrar usuario, e introduce el identificador en la ventana emergente. Posteriormente el controlador le pasa la información al modelo que se encarga de eliminarlo.

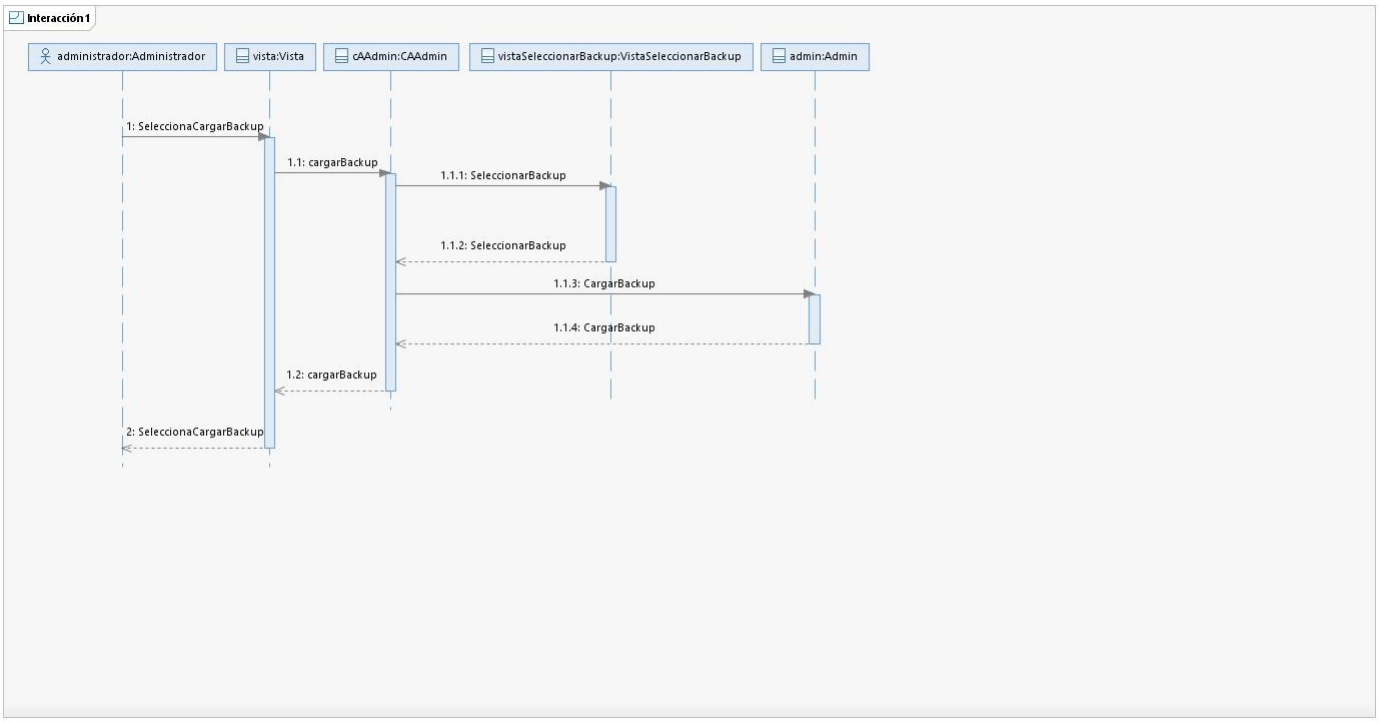


Este diagrama representa la opción de editar un usuario, tras elegirla el administrador tendrá que introducir el identificador del empleado que quiera modificar. Posteriormente elegirá si quiere resetear la contraseña o modificar el rol del usuario. Una vez realizada la elección el controlador se la enviará al modelo que hará la tarea correspondiente.

El siguiente diagrama se corresponde con el guardado de un backup de los usuarios del sistema, el administrador tras elegir esta opción creara un respaldo cuyo nombre será la fecha y hora actuales del sistema.

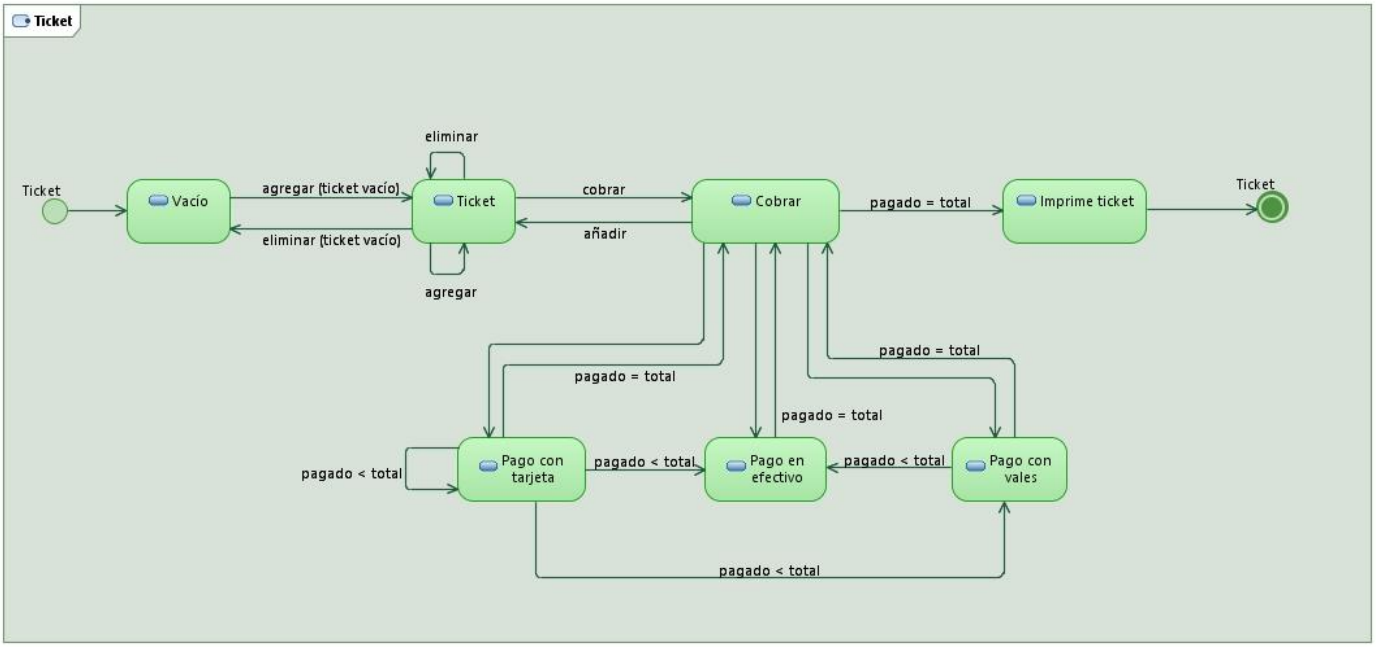


Este último diagrama del parte del administrador representa la carga de un backup. Tras pulsar el botón correspondiente saldrá una ventana emergente que le permitirá elegir el fichero correspondiente. Tras esto el controlador le pasará la información al modelo que procederá a la carga del fichero elegido.



6 Diagrama de máquina de estados

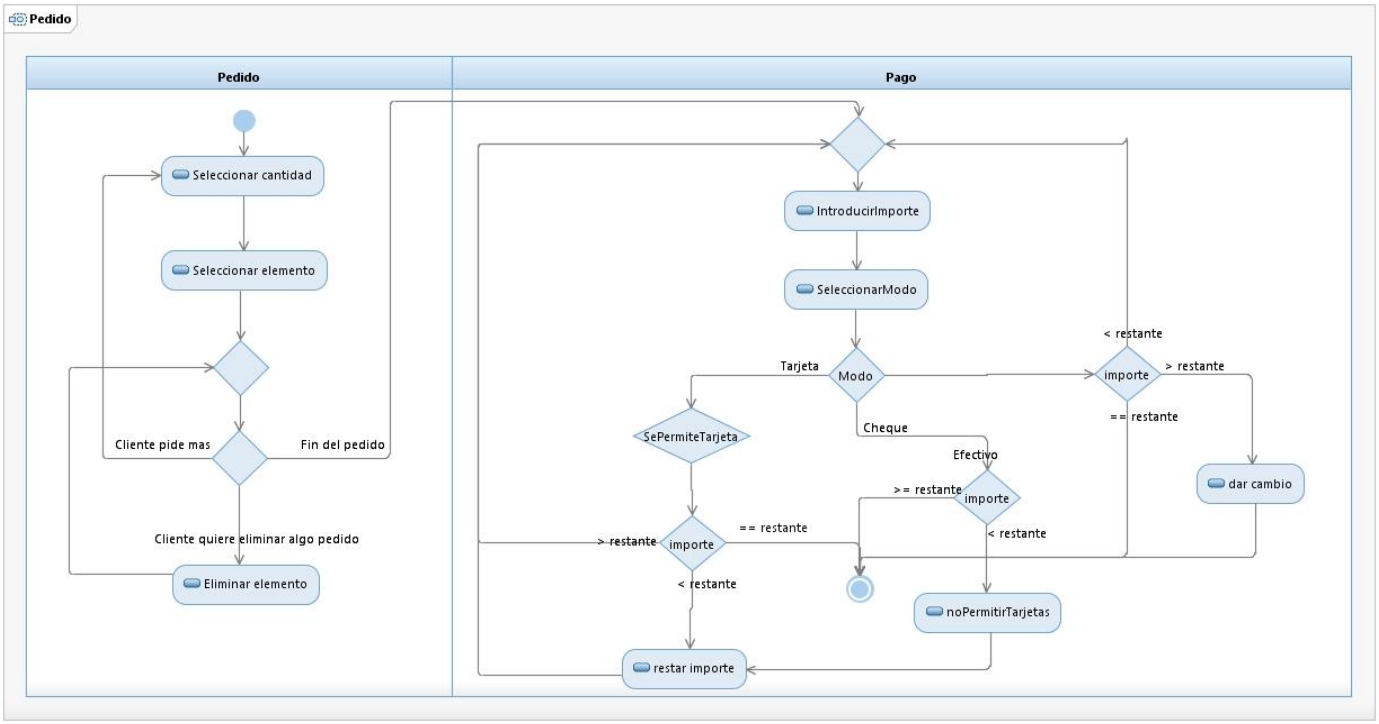
A continuación se muestra el diagrama de máquina de estados elegido. Ha sido sobre el objeto "Ticket", ya que va pasando por distintos estados. Además, al ser el pago una de las partes más complejas de nuestro programa, consideramos necesario tener diagramas que aclaren su funcionamiento.



7 Diagrama de Actividades

A continuación se muestra el diagrama de actividades elegido. Nuevamente tiene que ver con el pago, como la acción pago-pedido se lleva de principio a fin, es adecuado hacer un diagrama de actividades.

Inicialmente, el diagrama comienza en el pedido (partición de la izquierda)



8 Comentario sobre las clases del Proyecto

A continuación se explican las clases que forman parte del proyecto. Cabe destacar que al usar el modelo de tres capas, el diagrama de clases, así como las clases del mismo se vieron modificados, se dedicará el primer apartado de este punto a las clases antiguas que han sido modificadas, y el segundo a las nuevas, que forman parte del modelo de tres capas.

8.1 Clases antiguas

TPV: básicamente era la clase principal, el punto de entrada al programa. Ahora se centra principalmente en el *CAPrincipal* y el *CFrontal*, descritos en el siguiente punto.

UserList: clase que mantenía la Base de Datos de los usuarios, ahora esta clase se llama *BDUsuarios***.**

User: clase que representaba a un usuario, de esta heredaban *Admin, Cajero, Supervisor y Gerente.*Anteriormente implementaban diversas, funciones, pero con el nuevo modelo esto ha dejado de ser así y han pasado a ser, simplemente, contenedores de información de la *BDUsuarios*. En el nuevo modelo se llaman igual.

Carta: similar al caso de UserList, ahora se mantiene una *BDCarta.*

Producto: como con user, solo que ahora ha cambiado levemente la estructura (ver clases nuevas).

Sesion: clase que formaba parte del patrón *strategy* del modelo antiguo. De la misma heredaban los cuatro menús distintos de los empleados, ahora toda esta funcionalidad se divide en las nuevas clases de servicio de aplicación y controladores de aplicación asociadas a cada usuario.

Ticket: el ticket anteriormente implementaba funciones que actualmente se encuentran entre *CAClogin* y *Login*

8.2 Clases nuevas

A continuación se describen las clases del programa, aplicando el modelo de tres capas.

CAPrincipal: controlador principal de la aplicación, que se encarga de inicializar todo lo necesario para su funcionamiento.

CFPrincipal: controlador frontal de la aplicación, su función es la de mantener un correcto cambio entre las diversas pantallas.

CALogin: controlador de la pantalla de login, su ocupación es comprobar que los accesos de los usuarios sean correctos, y relaciona la vista del login con su modelo.

Login: esta clase se corresponde con el modelo del login, que permite hacer inicios de sesión.

CAPassPanel: controlador de la ventana emergente en la que se introduce el usuario y la contraseña, implementa la interfaz ActionTecladoPassListener.

ActionTecladoPassListener: interfaz cuya función es reaccionar ante los botones del teclado en pantalla.

CAAdmin: controlador de la aplicación que se encarga de unir la vista de la pantalla del administrador con sus funciones propias, implementa la interfaz ActionAdminListener.

ActionAdminListener: interfaz que permite responder a las acciones realizadas en la pantalla del administrador.

Admin: modelo de la aplicación referente al administrador, permite la creación, edición y borrado de usuarios, junto con la carga y guarda de los backups de estos.

User: es el modelo de los usuarios del sistema, de esta clase heredan las siguientes:

- **Cajero**
- **Supervisor**
- **Gerente**

Estas tres clases van en jerarquía, cada una tiene todas las funciones de las anteriores.

Admin: hereda también de usuario pero aparte, no consigue las funciones de los demás usuarios.

BDUusarios: es la representación de la base de datos que contiene a los usuarios del sistema.

UserClientDao: DAO que encapsula la información de un usuario.

FactoryUserDao: genera la factoría de DAOs de usuarios según la base datos elegida.

TxtFactoyClienteUserDao: clase creada por FactoryUserDao se en carga de devolver el TxtBDUserDAO.

TxtBDUserDao: el DAO que encapsula la base de datos de usuarios.

TOUsuario: objeto de transferencia de los usuarios del programa.

CACajero: controlador de la aplicación que relaciona la vista del cajero con sus funciones propias, implementa las siguientes interfaces:

ActionPagoListener

ActionPedidoListener

Estas dos interfaces permiten reaccionar ante los botones de pedido y pago respectivamente.

ActionTecladoPedidoListener:

ActionTecladoPagoListener:

Estas otras dos reaccionan al teclado en los momentos del pedido y del pago.

Cajero: modelo de la aplicación que representa a un cajero, que permite tomar un pedido y cobrarlo.

ElementoCarta: esta clase representa a los elementos que se pueden incluir en la carta, de esta heredan las siguientes dos clases.

Producto: representa los elementos individuales de la carta, abarca las siguientes clases que heredan de ella:

- **Hamburguesa**
- **Complemento**
- **Bebida**
- **Postre**

Menu: esta clase se encarga de unir varios productos en un solo elemento de la carta.

BDCarta: representación de la base de datos que contiene la carta de la hamburguesería.

FactoryCartaDao: genera la factoría de DAOs de la carta según la base datos elegida.

TxtFactoyClientCartaDao: clase creada por FactoryCartaDao se en carga de devolver el TxtBDClientCartaDAO.

TxtBDClientCartaDao: el DAO que encapsula la base de datos de la carta.

TOElementoCarta: objeto de transferencia de los elementos de la carta.

CartaClientDao: DAO que representa la carta de la hamburguesería.

9 Patrones implementados

Para el desarrollo de un Software de calidad, así como para facilitar su mantenimiento y mejora constante, se han usado diversos patrones de diseño en esta aplicación, no sólo con el fin de facilitar a los programadores las tareas anteriores, sino también de abaratar costes.

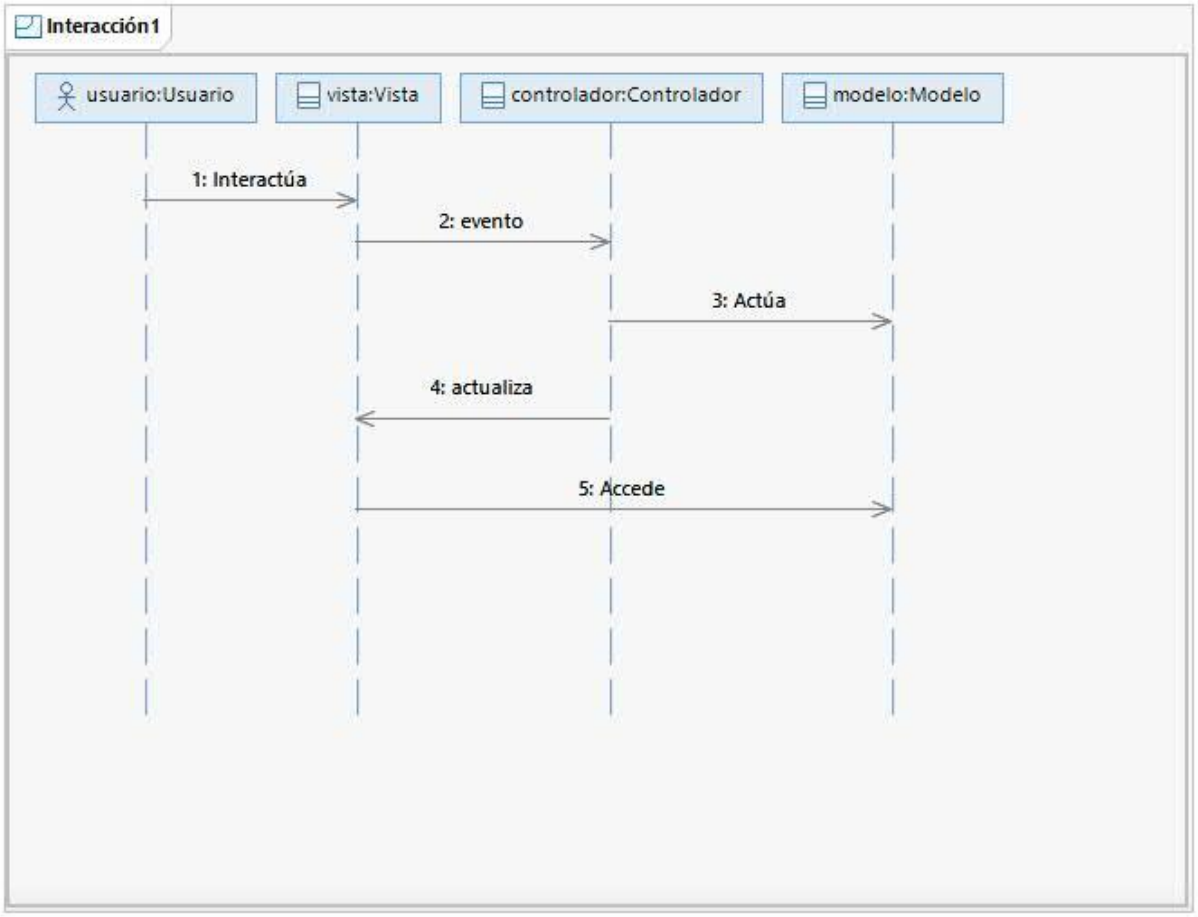
En el apartado del patrón arquitectónico, nos hemos decantado por un patrón multicapa, por las ventajas que conlleva:

- Soporte para múltiples clientes, pues necesitamos varios trabajadores utilizando la aplicación.
- Encapsulación y particionamiento del programa, pues tiene funciones diferentes que deben mantenerse separadas para facilitar su desarrollo y mantenimiento.
- Se facilita la modificación de capas independientes, tales como la visual o la lógica del programa, como se comenta en el punto anterior.
- Permite desarrollar más rápido al poder programar los diferentes componentes por separado.
- Permite una alta integración, por ejemplo, con diferentes bases de datos.

9.1 Patrones de diseño básicos ligados a multicapa

Una vez elegida una arquitectura multicapa, se utilizaron los siguientes patrones, por razones similares a las anteriores:

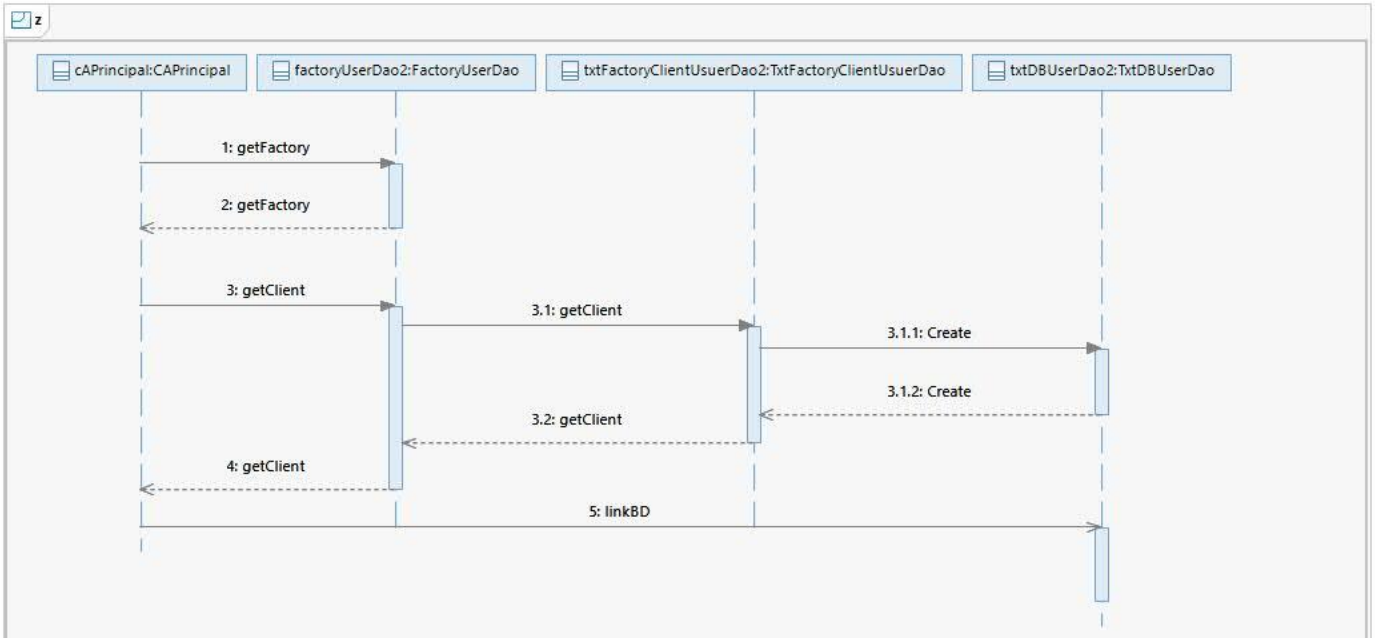
- **Modelo-Vista-Controlador.** Al ser un patrón con el que los implementadores tienen experiencia, se utilizó, en conjunto con la arquitectura de tres capas (ver más abajo). La idea de utilizar este patrón es la de encapsular y separar aún más el código, separando la interfaz gráfica y la lógica del programa utilizando un controlador. En nuestra aplicación, hemos utilizado el MVC pasivo.

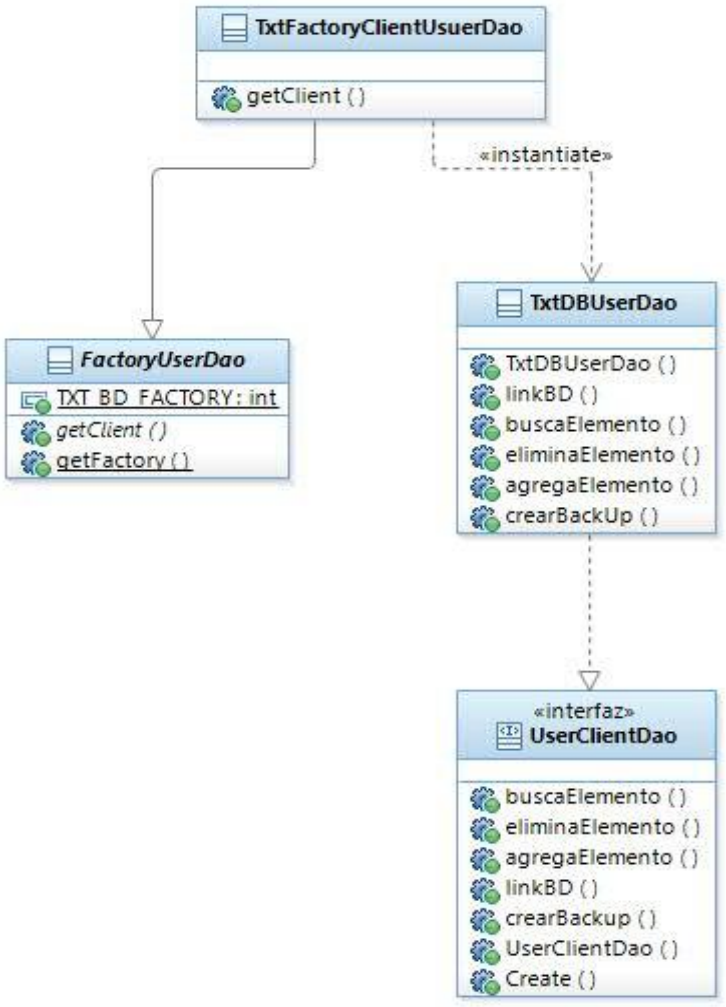


En el diagrama de clases, las clases ControladorFrontal y Controlador de aplicación forman parte del controlador del MVC, y las clases Login, Admin y Cajero del modelo del MVC, pues contienen la lógica del sistema. El resto de clases tiene más sentido comentarlas en el apartado de los patrones del modelo de tres capas, más abajo.

- **Singleton.** En nuestra aplicación hay múltiples clases que por sus características sólo serán instanciadas una vez, por lo tanto se optó por la utilización de este patrón. Casos del uso de este patrón son el Login (y las clases asociadas, como su controlador), el ControladorFrontal, o las factorías del punto siguiente (se puede observar que getFactory es estático).

- **Factoría / factoría abstracta.** Este patrón se ha utilizado en los DAO, para facilitar el uso (y adaptación) de diferentes bases de datos, así como de modificar las actuales y de incluir otras nuevas. Cómo ejemplo con el DAOUsuarios(análogo para la carta):

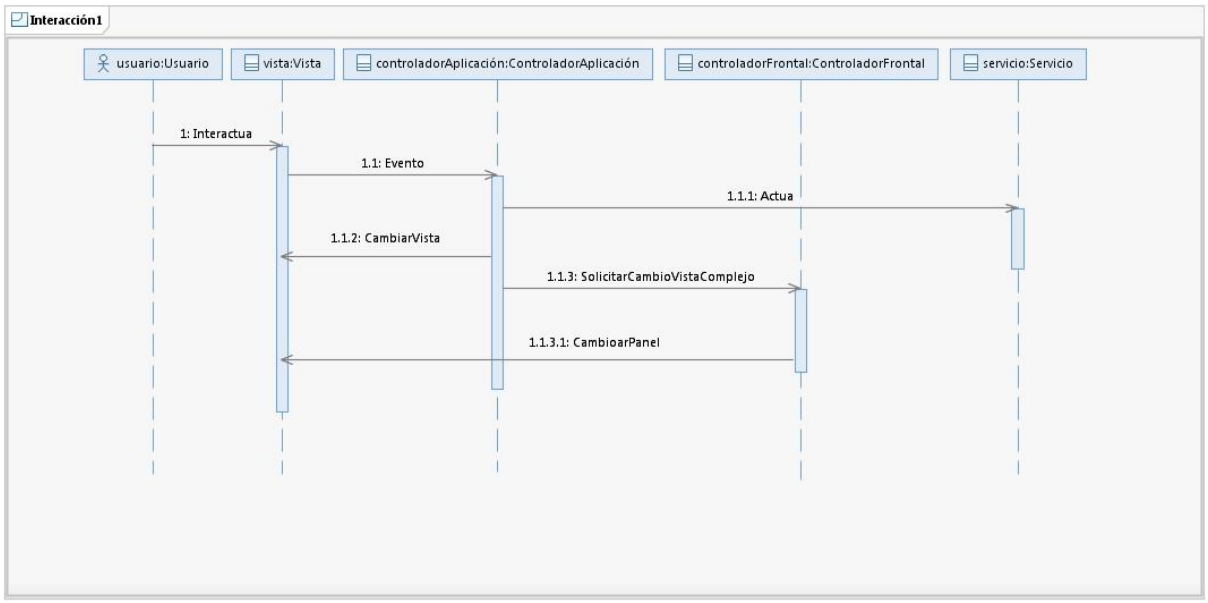




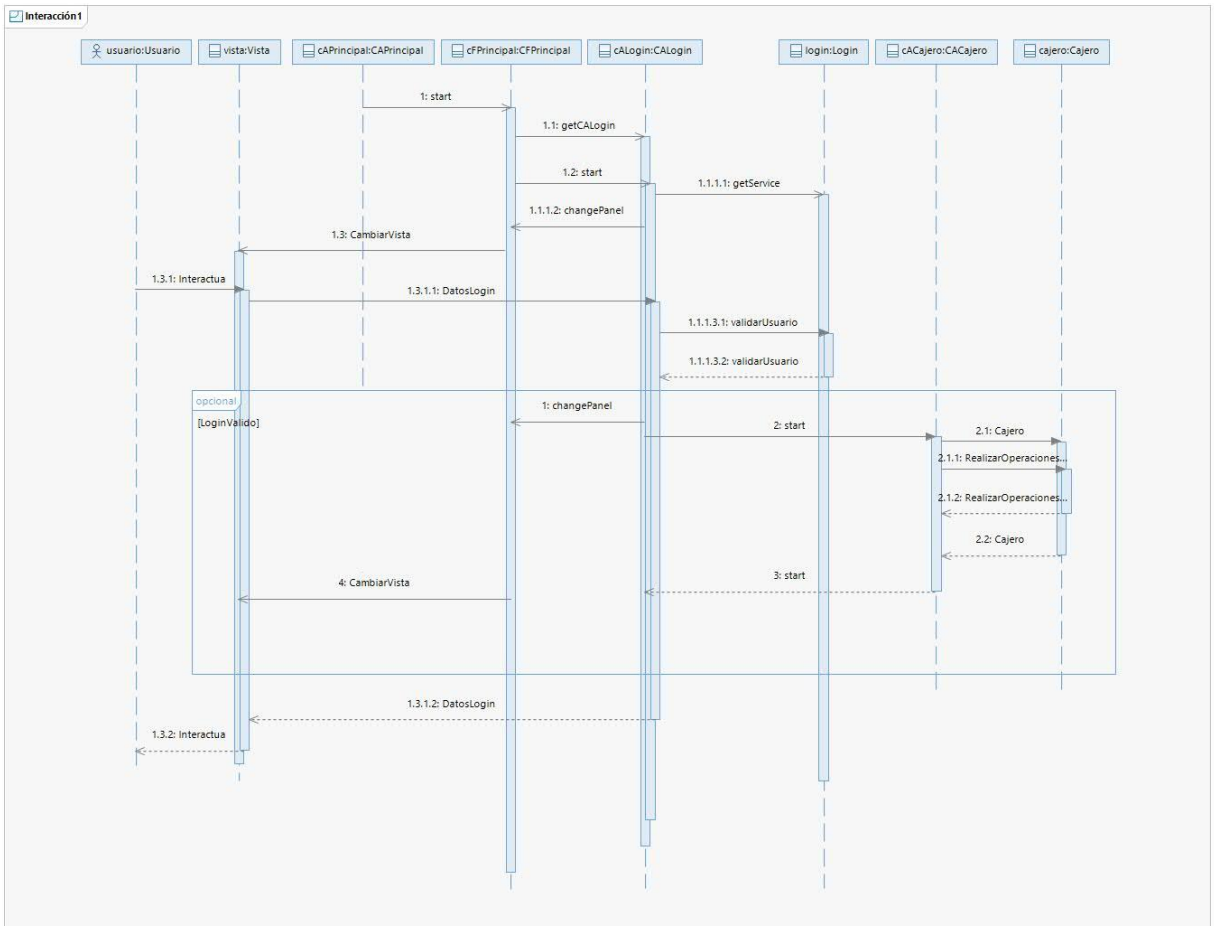
9.2 Arquitectura y patrones arquitectónicos

Como se menciona previamente, se utiliza una arquitectura de tres capas, la capa de negocio, encargada de la lógica del sistema, la capa de presentación, encargada de la interfaz visual y de su comunicación con la capa de negocio, y la capa de integración, encargad de facilitar y encapsular la comunicación de la capa de negocio con el almacén de datos, consiguiendo un funcionamiento transparente de la base de datos.

El funcionamiento se muestra en el siguiente diagrama, y se profundiza en los puntos sucesivos del documento:



Como complemento se adjunta un ejemplo sobre el proyecto:

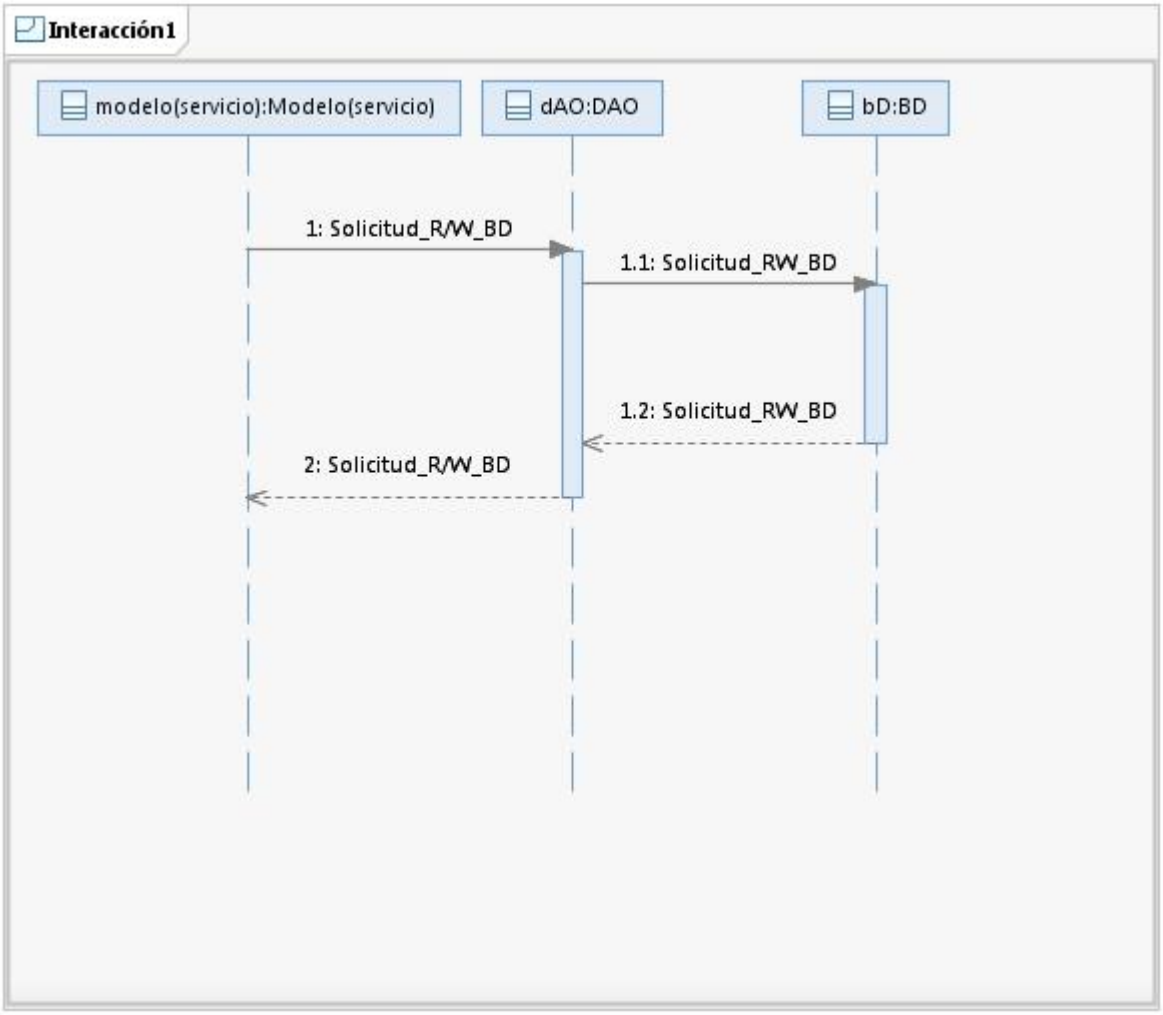


9.2.1 Capa de presentación

En la capa de presentación se han usado los patrones de "Controlador de Aplicación" y "Controlador frontal". Sin embargo, al no ser una aplicación muy grande, se ha modificado el patrón, pues supondría bastante repetición de código. De esta manera, Se mantiene un Controlador Frontal, denominado CFPrincipal, que es el encargado de realizar grandes cambios en la vista, como puede ser cambiar un panel entero. También está implementado un "Controlador de Aplicación", CAPrincipal, que es el punto de entrada al programa, se encarga de iniciar los DAO con la base de datos, instanciar el CFPrincipal y pasar a la vista de login. Además de esos controladores, cada sesión tiene al menos un Controlador de aplicación, concretamente el Login tiene CALogin, y CAPass, y el admin CAAdmin, que realizan las labores de interacción vista-modelo (Controlador frontal) y con otros modelos y controladores (Controlador de aplicación). Los diagramas anteriores muestran estas interacciones.

9.2.2 Capa de Integración

En la capa de integración se ha usado el patrón DAO para una mejor integración de las distintas bases de datos. Al fusionarlo con el patrón "Factoría abstracta" (ver más arriba), se consigue una flexibilidad que posibilita la integración rápida de nuevas bases de datos, así como de mantener bases de datos de diferentes características a la vez.



El servicio realiza una petición al dao, que actúa en consecuencia según la BD. Los diagramas en el punto "Factoría abstracta" definen la jerarquía y creación de los DAO.

9.2.3 Capa de Negocio

En la capa de negocio se utilizan dos patrones.

Por un lado, está el patrón de **Transferencia**, que se encarga de encapsular la estructura de datos de las BD; haciendo así que su uso sea transparente al modelo y, junto con los DAO; se consigue independizar la estructura y funcionamiento de la BD a la lógica de programa. Su uso es simple, el servicio utiliza los TO ("transfer object") pertinentes, esperando que los DAO devuelvan elementos de este tipo, o bien pasando a los DAO los TO; por otro lado los DAO traducen los TO a/desde la estructura de datos de la bd pertinente.

Por otro, se utiliza el patrón **Servicio de aplicación**, que se encarga de mantener la lógica del programa agrupada. En nuestro proyecto se mantienen las clases Login, Admin y Cajero, que implementan la lógica del programa asociada, independientemente del resto de la aplicación.

9.3 Otros patrones

Además de los patrones anteriores, se utilizan los siguientes;

- **Iterador.** En la aplicación se utilizan elementos iterables de java.collection, y como en determinados casos han de recorrerse, se ha utilizado el patrón iterador por su facilidad de uso, su eficiencia computacional, todo ello aparte de venir ya implementado en el SDK utilizado.
- **Composición y decorador.** Principalmente utilizado en la vista

10 Casos de prueba

La aplicación se encuentra totalmente funcional en las sesiones de Cajero(exceptuando el loginInterno de Supervisor), y el Admin. Para poder probar el sistema, se encuentra integrado en el código los siguientes usuarios predefinidos:

- ID:0, Pass: 1234, rol: Admin
- ID:1, Pass: 1234, rol: Cajero
- ID:2, Pass: no definida, rol: Cajero
- ID:3, Pass: 1234, rol: Supervisor
- ID:4, Pass: 1234, rol: Gerente

Al estar implementado el sistema de administración, es posible crear, modificar y borrar usuarios. Estos cambios quedarán reflejados en la sesión actual hasta que se cierre el programa, sin embargo, cualquier eliminación realizada sobre dichos usuariosserá revertida con los valores anterioresal volver a ejecutar la aplicación; sin embargo, creaciones y modificaciones sí que se guardarán en la base de datos.

10.1 Prueba de login

Seleccione Cajero. Cómo se especifica en el srs, sólo un Cajero, Supervisor ó Gerente pueden loguear.

- Introduzca ID: 0 pass 1234, dará error porque el usuario 0 es un admin.
- Introduzca ID: 1 pass 1111, dará error porque la contraseña no es correcta.
- Introduzca ID: 1 pass 1234, el login es correcto.
- Introduzca ID: 2 con cualquier contraseña, el login es correcto ya que el usuario no tenía contraseña definida.
- Introduzca ID: 2 con una contraseña diferente a la anterior, dará error.

10.2 Prueba Admin

Seleccione admin y loguee con ID 0 pass 1234

- Seleccione Crear Backup, se creará un backup con nombre la fecha y hora actuales.
- Seleccione Nuevo usuario, con ID:1, dará error porque el usuario ya existe.
- Seleccione Eliminar usuario, con ID:1, se eliminará el usuario.
- Seleccione Modificar usuario, con ID:1, dará error porque el usuario no existe.
- Seleccione CargarBackup, seleccione el backup anterior.
- Seleccione Editar usuario, con ID:1, se le dará a elegir entre cambiar rol y resetear password.
- Haga clic en “Salir” para cerrar la sesión como administrador.

10.3 Prueba Cajero

Seleccione cajero en la pantalla de login y loguee con un cajero.

- Introducir pedidos:
    - Haga clic en “bebida”. Aparecerá en el panel superior derecho.
    - Haga clic en “Cantidad”. A continuación, en el 2. Aparecerá un 2 en el panel de cantidad.
    - Haga clic en “Menu”. Aparecerán dos menús en el panel superior derecho.
    - Haga clic en “Postre”. Aparecerá en el panel superior derecho.
    - Haga clic en “Complemento” Aparecerá en el panel superior derecho.
    - Haga clic en la “X” roja del panel superior derecho de complemento en el panel superior derecho. Se eliminará el complemento.
- Haga clic en “Intro” para proceder al pago.
    - Observe que en esta pantalla no se pueden eliminar elementos del panel superior.
- Haga clic en “Atrás”, elimine “bebida” y haga clic en intro. La bebida se eliminará y se recalculará el coste.
- Introduzca el importe “15.50” con el teclado numérico.
- Haga clic en “Tarjeta”. Se mostrará un error porque no se permite un importe superior con tarjeta al total.
- Haga clic en “C” del teclado numérico, introduzca como importe “5.00” y haga clic en tarjeta.
    - Se descontará el importe del total. Observe que ya no se permite volver a atrás.
- Introduzca como importe “5.00” y haga clic en “Cheque”.
    - Se descontará el importe del total. Observe que ya no se pueden pasar más tarjetas.
- Introduzca como importe “6.50” y haga clic en “Efectivo”
    - Se mostrará el cambio correspondiente a la operación, se desactivarán los botones del pago y se activará el botón “Intro” para efectuar un nuevo pedido.
- Haga clic en “intro”. En la pantalla siguiente, introduzca una hamburguesa y pase a la pantalla de pago volviendo a pulsar “intro”.
- Introduzca como importe “2.00” y seleccione efectivo.
    - Dará error ya que no se permite menos importe en efectivo que el total.
- Pulse “C” en el teclado numérico. Introduzca como importe “6.00” y haga clic en “Cheque”
    - Se finalizará el pago y se permitirá pasar a un nuevo pedido.
- Pulse “Cerrar” para cerrar sesión.

11 Gestión del proyecto

La planificación de este segundo cuatrimestre ha estado marcada por varios elementos clave que vale la pena explicar, exponer y profundizar en ellos.

- Las entregas de seguimiento cada 2 semanas, estas entregas han sido una guía y una pauta de qué teníamos que hacer para ir en el proyecto, ha sido de ayuda de cara a saber que hay que hacer, pero también ha supuesto un problema dado que sólo sabíamos que hacer para las próximas 2 semanas, sin poder avanzar más en caso de ir con tiempo para tener un margen de error en caso de sufrir algún riesgo.
- La corrección de la entrega del primer cuatrimestre ha supuesto un contratiempo por dos factores clave, el primero que no se ha tenido en cuenta en esas entregas semanales de seguimiento marcadas de forma externa, y segundo que por un error en la notificación de la misma por cual está no se ha podido realizar de forma completa hasta la última etapa del primer cuatrimestre.
- El uso y aprendizaje prácticamente autodidacta de la herramienta IBM Rational Software Architect for WebSphere Software ha supuesto un reto enorme desde el primer momento, a pesar de haber tenido a nuestra disposición un pequeño apéndice sobre la aplicación, este ha resultado ser insuficiente para describir como trabajar con la herramienta, esto ha supuesto reajustes en la planificación para tener en cuenta y solucionar los problemas surgidos por el uso de esta nueva herramienta.

  En concreto cabe destacar algunas de las soluciones que han llevado a cabo para poder mejorar el uso de la aplicación. En primer lugar dado los problemas para tener la aplicación esta se subió a un servidor en la nube para su fácil descarga junto con un pequeño tutorial para la instalación básica de la aplicación, además se han realizado varias reuniones en los laboratorios de la facultad con el objetivo de resolver dudas y buscar soluciones a los problemas del uso de esta aplicación.

- La implementación del prototipo con dos sesiones completas y la gestión de los casos de prueba ha supuesto un reto en esta última etapa del proyecto, dado que ha coincidido en el tiempo con las correcciones del primer cuatrimestre, sólo 4 miembros del grupo tienen los conocimientos para programar y además en este mes de mayo no son pocos los proyectos ajenos al proyecto TheGodBurger en los cuales los miembros de equipo están implicados, por ello se han tomado una serie de medidas, las dos más importantes han sido la creación de dos equipos dentro del grupo para poder gestionarnos mejor y la segunda el aumento de reuniones para gestionar más de cerca el proyecto y no tener que trabajar dos veces por culpa de un error de comunicación.

En cuanto a los riesgos sufridos durante este segundo cuatrimestre hay que destacar que casi todosellos ya habían sido previstos y por ello no han supuesto un problema grave en la planificación del proyecto, a continuación se detallan los riesgos sufridos, consecuencias y soluciones:

- Riesgo de contrato: nuestro contrato el srs ha sufrido una serie de correcciones, por suerte, por suerte y gracias a una gestión proactiva de los riesgos esto no ha supuesto un problema grave para la finalización del proyecto, aunque si ha supuesto una pequeña carga de trabajo extra que se ha subsanado dedicándole alguna hora extra.
- Riesgo de desarrollo: desarrollo de la aplicación ha supuesto un reto enorme por parte del grupo dada la falta de conocimientos de programación de la mitad del equipo, esto se ha solucionado de forma natural puesto que ya está previsto desde el primer día, dando menos carga de trabajo no relacionado con la aplicación al equipo de programación para que este pudiera acabar la aplicación a tiempo para la entrega.
- Riesgo planificación: en este segundo cuatrimestre no se han dado grandes cambios en la planificación, el único destacable ha sido la corrección del primer cuatrimestre que no se tuvo en cuenta en un primer momento, pero este no ha supuesto un problema grave dado que en este a última etapa del proyecto la parte del grupo que no estaba programando disponía de tiempo suficiente para realizar la corrección de forma efectiva.
- Riesgo de falta de presupuesto: en cuanto al coste en horas del proyecto en este segundo cuatrimestre cabe destacar que no ha habido que recortar el proyecto para llegar a tiempo, sin embargo si se han descartado ideas en la implementación del aplicación, todos ellas relacionadas con la vista y la parte del proyecto ya que se ha decidido dejar la aplicación con una vista más primitiva consiguiendo con esto poder aplicar algún patrón más y mejorar y corregir en algunos casos los diagramas de este cuatrimestre .
- Riesgo de abandono: gracias a la buena gestión de grupo se ha conseguido llegar al final del año con todos los miembros en el equipo, el mérito de esto es de todos y de la política de transparencia de opiniones y críticas dentro del grupo, por norma se ha promovido el buen ambiente en el grupo y siempre se ha escuchado y dado voz a todos los miembros del grupo, en general cabe destacar las dinámicas llevadas a cabo en la reuniones , donde los primeros minutos siempre se dedican a decir lo que cada uno piensa y a valorar el esfuerzo del todos a pesar de tener un jefe de proyecto que ha tenido siempre la última palabra.
- Riesgo de comunicación en cuanto a la comunicación entre los miembros del grupo no ha habido grandes cambios, semana a semana todos los viernes nos hemos reunido para gestionar el proyecto, por otro lado lo que sí ha cambiado ha sido el incremento de más reuniones por parte de grupos reducidos dentro del grupo general, donde se han tratado los temas correspondientes en cada caso, por ejemplo los jueves de las tres últimas semanas de mayo se ha reunido el equipo de programación para gestionar las creación de la aplicación.

Por último y para concluir la coordinación del grupo se ha llevado a cabo de manera vertical y muy jerarquizada para evitar que algún miembro del grupo no tuviera nada que hacer, de modo que el jefe de proyecto tras la reunión dada instrucciones claras y concisas de lo que hacer, bien a los miembros de forma individual o bien a unos de los dos equipos formados dentro del grupo, el equipo de programación y el que no programaba y han sido ellos los que han gestionado como realizar su parte del trabajo.