Ingeniería del Software 2007-2008

Gestión de una Red de Kioscos

Grupo E - 4ºC

Rodrigo Denís Cepeda Mateos
Alfonso García Pérez
Felipe Gutiérrez Lébedev
Javier Lázaro Requejo
Cristina Marco de Francisco
David Mirón García
David Rodríguez Pérez
Javier Santiago Medina

Distribución de Tareas

Especificación	Todos
Diagrama actividades	Felipe Gutiérrez, Cristina Marco
Diagrama de clases	Alfonso García, Javier Lázaro
Diagrama componentes	Javier Lázaro, Javier Santiago
Diagrama estados	Denís Cepeda, David Rodríguez
Diagrama despliegue	Felipe Gutiérrez, David Rodríguez
Diagrama secuencias	David Mirón, Cristina Marco
Interfaz gráfica	Denís Cepeda, David Rodríguez
Pruebas	Javier Santiago
Restricciones OCL	Alfonso García
Tablas de la base de datos	Javier Lázaro

Índice General

Introducción

La aplicación asiste en la logística de un almacén y una red de kioscos.
El encargado del almacén y los empleados de los kioscos utilizan la aplicación para gestionar y automatizar el proceso de recepción de pedidos y distribución de artículos.
Se dispone de terminales en los kioscos y en el almacén, que están conectados al servidor central.
Una base de datos gestiona toda la información del sistema, como el catálogo, el inventario, etc.

Especificación del Proyecto

Para hacer referencia a los distintos diagramas la nomenclatura usada es :

Diagrama de clases → DC
Diagrama de actividades → DA
Diagrama de secuencias → DS
Digramas de casos de usos → DCU
Diagrama de clases de la interfaz gráfica → DCIG
Interfaz gráfica → IG
Invariantes → INV

KIOSCO	**Logueo**	K1.Antes de poder utilizar la aplicación el kiosquero debe identificarse con su nombre de usuario y contraseña.	DA1, IG01, IG02
		K2.El kiosquero puede cambiar su contraseña.	DA2, IG03
	Saldo	K3.Todo kiosco tendrá un saldo. Este saldo se utilizará para comprar artículos al almacén.	DS3, DS7, IG07
		K4.Dicho saldo aumentará con las ventas realizadas.	DS1, DS2
		K5.Cada kiosco podrá visualizar su saldo y gasto previsto en pedidos.	IG07
		K6.El saldo de un kiosco no puede ser negativo.	DS3, INV11
		K7.Todo kiosco tendrá un inventario que listará todos los artículos en su posesión, y podrá consultarlo.	DC, IG08
	Pedidos	K8.Deberá poder realizar pedidos al almacén de artículos existentes en el catálogo, indicando los artículos y la cantidad correspondiente de cada uno. También deberá indicar la fecha de entrega del pedido, es decir, la fecha en que el almacén repartirá el pedido al kiosco. Sólo habrá un pedido para cada dia.	DS3 INV4, IG10
		K9.Los pedidos pueden ser modificados mientras no haya prescrito la hora de registro de los mismos, es decir, las 23:00 del día anterior a la fecha de entrega.	DS3, DS4a, DS4b
		K10.Una vez registrados, los pedidos pueden ser consultados pero no modificados.	DS3, DS4a, DS4b
		K11.El valor total de los pedidos por atender debe ser menor o igual al saldo del kiosco. Esto se comprobará cada vez que el kiosquero intente añadir un nuevo artículo a su pedido o modificar uno antiguo.	DS3, DS4a, DS4b
		K12.Los pedidos podrán ser anulados antes del momento de registro.	DS5, IG10
	Albarán	K13.Una vez atendido y satisfecho el pedido por parte del almacén, se generará un albarán que listará todos los artículos concedidos. Solo se generará un albarán por pedido.	DS6a, DS6b, INV6, IG11
		K14.En este albarán el kiosquero deberá confirmar la recepción / no recepción de cada uno de los artículos.	DS7, DS8, IG12
		K15.Si se confirma la llegada de un artículo, éste se extrae del inventario del almacén para añadirse al inventario del kiosco.	DS7
		K16.Si se confirma la no llegada del artículo, éste desaparecerá del inventario del almacén, pero no será incluido en el del kiosco. Se guardará constancia de esta acción (artículo extraviado).	DS8
		K17.Cada albarán se confirmará una única vez.	DS7, DS8
		K18.Un kiosco puede consultar todos sus albaranes.	DCU

ARTÍCULOS	Ar1.Existen dos conceptos de artículo. El tipo de artículo y el artículo en sí.	DC
	Ar2.El *tipo de artículo*, o artículo del catálogo, representa las características comunes de un conjunto de artículos iguales.	DC
	Ar3.El *artículo* en sí, o artículo de inventario, representa un elemento concreto.	DC
	Ar4.Se dispondrá de un único catálogo en el que figurarán las características de los tipos de artículo con los que se comercia (características descritas en el siguiente apartado blanco).	DC,
	Ar5.Dicho catálogo será editable desde el almacén y visible desde el almacén y desde cualquier kiosco.	DC, DA10, IG09, IG17
	Ar6.Los artículos del catálogo constan de las siguientes características: *nombre*, *PCA* (precio al que compra el almacén) y *PVP* (precio al que vende el kiosco).	DC
	Ar7.Un artículo del catálogo debe pertenecer a una, y solo una, de las siguientes categorías: *publicaciones*, *chucherías* o *genéricos*.	DC
	Ar8.Si un artículo del catálogo pertenece a la categoría de *publicaciones* necesitará indicar su fecha de publicación.	DC
	Ar9.Si un artículo del catálogo pertenece a la categoría de *chucherías*, necesitará almacenar su fecha de caducidad.	
	Ar10.Cada artículo pertenece a un tipo de artículo.	DC
	Ar11.Cada artículo debe estar asociado a un único inventario, ya sea de almacén o de kiosco. Además tendrá un número de referencia único.	DC, INV9
	Ar12.Si un artículo es devuelto por un kiosco, éste deberá volver al inventario del almacén. Hasta que el almacenista no confirme su llegada, el artículo permanecerá en el inventario del kiosco, pero no podrá ser vendido.	DS10, DS11, DS1, DS2, IG21
	Ar13.La aplicación deberá chequear todos los días la fecha de caducidad de los artículos de tipo chucherías. En caso de que un artículo esté caducado se eliminará del inventario en el que esté y se guardará constancia de esta acción.	DS18, INV2
	Ar14.Cuando un artículo del catálogo es descatalogado, todos los artículos asociados a él serán eliminados del inventario del almacén. Se guardará constancia de esta acción.	DS17

ALMACÉN	A1.Existe un único almacén en el sistema.	DC
	A2.Antes de poder utilizar la aplicación el almacenista debe identificarse con su nombre de usuario y contraseña. El nombre de usuario debe ser único.	DA1, INV3, IG01, IG02
	A3.El almacenista puede cambiar su contraseña.	DA2 , IG03
	A4.El almacén dispondrá de un saldo inicializado a 0.	DC
	A5.El almacén podrá incrementar dicho saldo.	DCU, IG13
	A6.El almacén gestionará el catálogo de la aplicación. Esto es, podrá *añadir* nuevos artículos de catálogo. Para añadir nuevos artículos del catálogo, debe indicar todas las características del mismo.	DS19, IG17a
	A7.El almacén podrá modificar los artículos del catálogo. Solo podrá cambiar el PVP y PCA.	DC, DA10, IG14
	A8.El almacén tendrá un inventario de todos los artículos de los que dispone, y podrá ser consultado por él mismo, y por cualquier kiosco.	DCU, IG18
	A9.Se podrán añadir artículos al inventario del almacén restándole el PCA del artículo al saldo.	DS20, IG20
	A10.El almacén podrá descatalogar tipos de artículos, los que queden en los kioscos podrán ser vendidos, pero no devueltos al almacén.	DS17, DS10, IG17
	A11.Cuando un almacén descataloga un tipo de artículo no se aceptarán pedidos de este tipo de artículo.	DS3
	A12.El almacén determinará el porcentaje generador del PVK. El PVK se obtiene incrementando el PCA en dicho porcentaje. Este porcentaje es único para todos los artículos.	DC
	A13.El almacén determinará el porcentaje generador del PDev. El PDev se obtiene incrementando el PCA en dicho porcentaje. Este porcentaje es único para todos los artículos.	DC
	A14.Desde el almacén se podrá *dar de alta* a un kiosco de la cadena.	DS14, DA4, INV1, IG16
	A15.A la hora de dar de alta un kiosco se le asigna un *identificado único*, una *contraseña* genérica, y un saldo inicializado a 0.	
	A16.El almacén podrá incrementar el saldo de cualquier kiosco.	DCU, IG15
	A17.El almacén podrá ver el inventario de cualquier kiosco.	
	A18.El almacén podrá ver los pedidos de cualquier kiosco.	
	A19.El almacén podrá ver los albaranes de cualquier kiosco.	DCU
	A20.El almacén podrá ver su saldo y el de cualquier kiosco.	DCU, IG13, IG15

ASIGNACIÓN DE PEDIDOS	As1.El sistema deberá gestionar los pedidos que son realizados al almacén por todos los kioscos.	DS6a, DS6b
	As2.Se crearán los albaranes de los pedidos automáticamente a las 23:00 h del día anterior a la fecha de entrega del pedido.	
	As3.En el caso de que no haya suficientes artículos para atender todos los pedidos, dichos artículos se asignarán proporcionalmente con respecto a los artículos pedidos.	

Diagrama de Casos de Uso

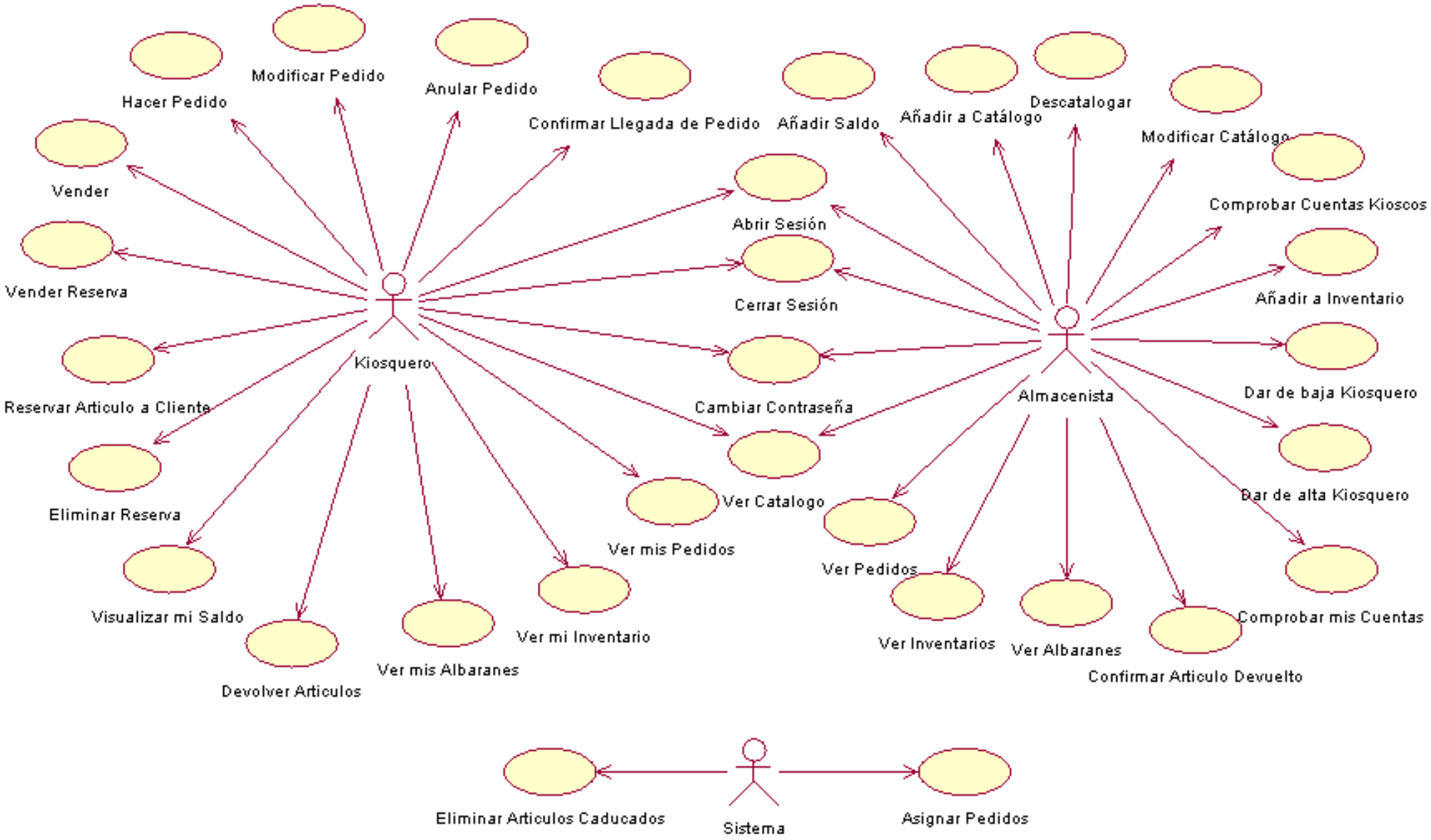
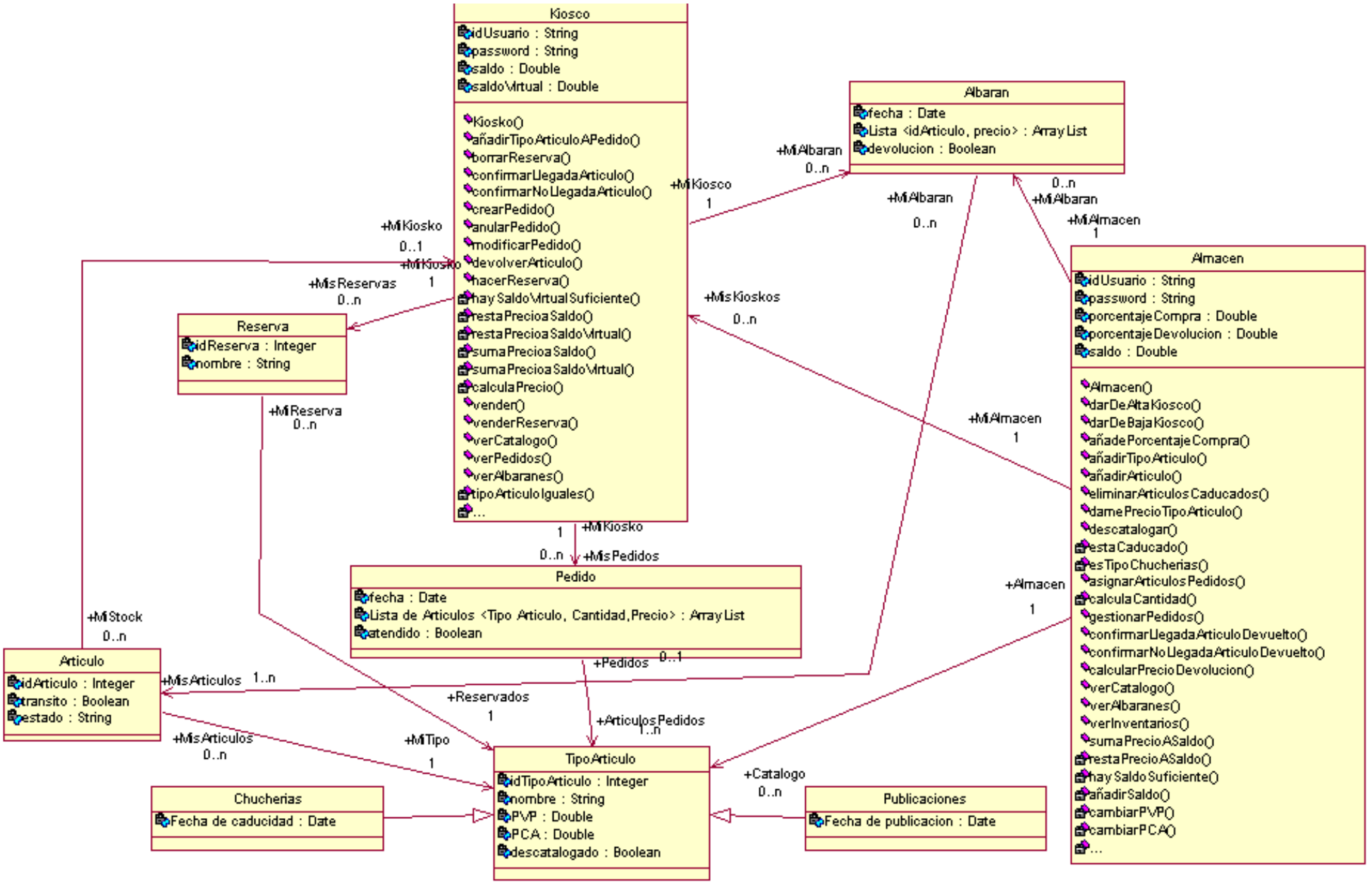
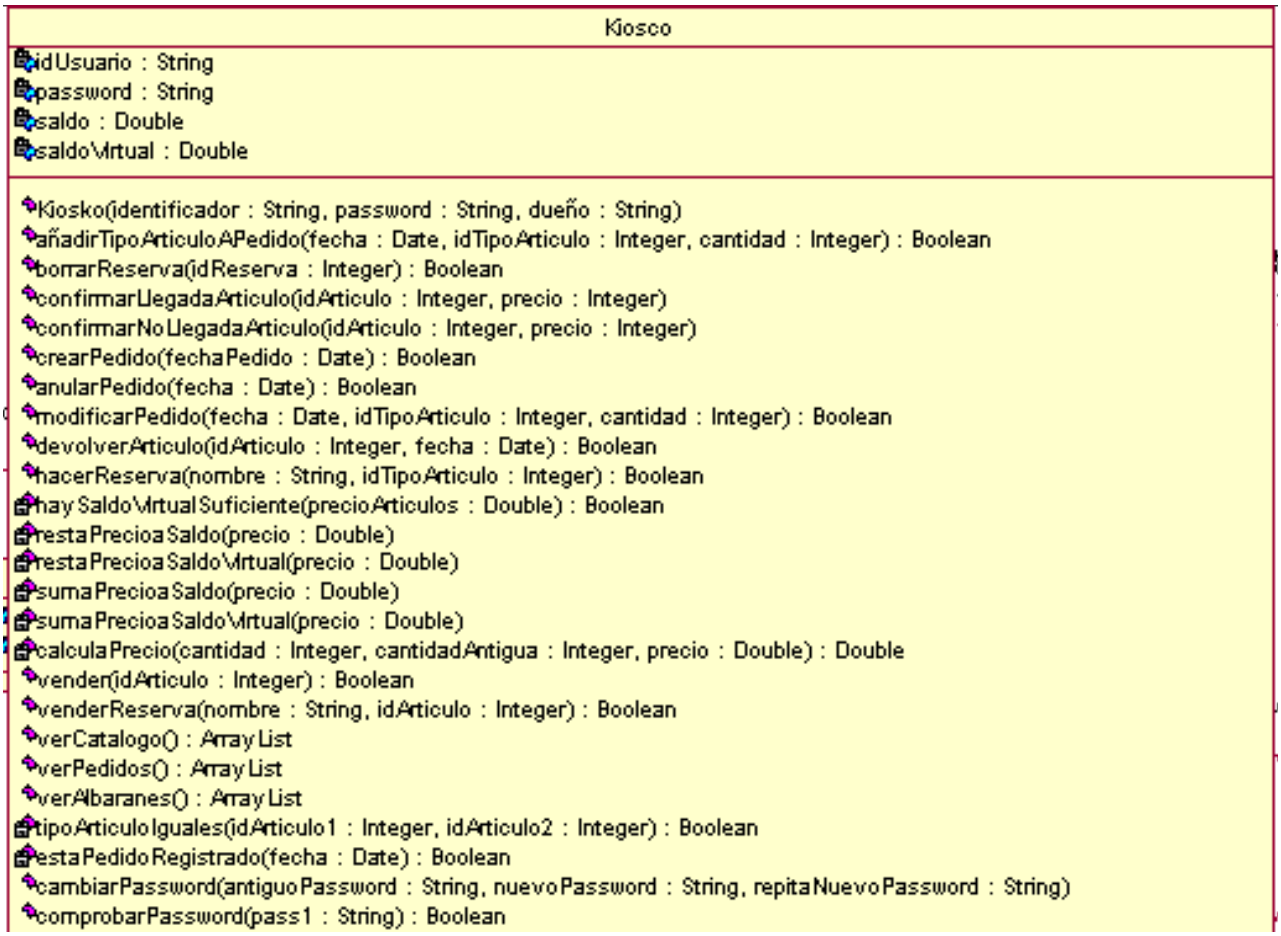


Diagrama de Clases



Clase Kiosco



Clase Almacén

Restricciones OCL

- ✓ INV1 - No puede haber dos kioscos con el mismo identificador.
  context Kiosco:
  Kiosco.allInstances -> forAll (k1: Kiosco | (self: Kiosco <> (k1: Kiosco) implies (k1: Kiosco.Identificador <> (self: Kiosco.Identificador))))

- ✓ INV2 - No hay artículos caducados en los inventarios.
  context Kiosco:
  self.miStock -> forAll (k2: Chucheria | k2.fecha > fechaActual)

- ✓ INV3 - El identificador del almacén es único.
  context Almacen:
  Kiosco.allInstances -> forAll (k1: Kiosco | (k1: Kiosco.Identificador <> (self: Almacen.Identificador)))

- ✓ INV4 - Un kiosco sólo tiene un albarán por día.
  context Kiosco:
  self.miAlbaran -> forAll (p1, p2: Albaran | (p2.fecha <> p1.fecha)

- ✓ INV5 - Un kiosco sólo tiene un pedido por día.
  context Kiosco:
  self.misPedidos -> forAll (p1, p2: Pedido | (p2.fecha <> p1.fecha)

- ✓ INV6 - A cada pedido atendido le corresponde un albarán.
  context Kiosco:
  self.misPedidos -> forAll (p1: Pedido | p1.fecha <= fechaActual implies (self.miAlbaran -> select (a1: Albaran | a1.fecha = p1.fecha) ->size () = 1))

- ✓ INV7 - Dos tipo artículo no pueden tener el mismo identificador.
  context Almacen:
  self.Catalogo -> forAll (t1, t2: Tipo-Articulo | (t1: Tipo-Articulo.Nombre = (t2: Tipo-Articulo.Nombre) implies (t1: Tipo-Articulo = (t2: Tipo-Articulo)))

- ✓ INV8 - Si dos publicaciones tienen el mismo nombre, son de fechas distintas.
  context Almacen:
  self.Catalogo -> forAll (p1, p2: Publicaciones | (p1: Publicaciones.Nombre = (p2: Publicaciones.Nombre) implies (p1: Publicaciones.FechaPublicacion <> (p2: Publicaciones.FechaPublicacion))))

- ✓ INV9 - La referencia de un artículo es única.
  Articulo.allInstances -> forAll (a1: Articulo | (a1: Articulo <> (a1: Articulo) implies (a1: Articulo.Referencia <> (self: Articulo.Referencia))))

- ✓ INV10 - Un artículo no puede aparecer en dos albaranes del mismo tipo en la misma fecha.
  context Albaran:
  Albaran.Allinstances -> forAll (alb1: Albaran | self.devolucion = alb1.devolucion and (self.fecha = alb1.fecha and (self.miKiosco <> alb1.miKiosco)) implies not (self.ArticulosAsignados -> exist (a1:Articulo | alb1.ArticulosAsignados -> exist (a2: Articulo | a1 = a2)))

- ✓ INV11 - Ningún kiosco puede tener saldo negativo.
  context Kiosco:
  Kiosco.Allinstances -> forAll (k1: Kiosko | k1.saldo >= 0)

Base de Datos

Tabla de la Base de Datos

Usuario (idUsuario, password, kioscoOAlmacen, saldo, saldoVirtual)

En esta tabla se almacenan todos los usuarios del sistema con su identificador único, su contraseña de acceso, el tipo de usuario es decir, si es kiosquero, o almacenista, el saldo real, y el saldo virtual que sólo se utiliza para los kiosqueros para saber si pueden pagar todos los pedidos que realizan; en almacenista será null.

Articulo (idArticulo, transito, idTipoArticulo, idUsuario)

En esta tabla se almacenan todos los artículos de los inventarios del almacén y de los kioscos, tienen un identificador único, que es el idArticulo; y el atributo en tránsito indica si el artículo esta siendo enviado a un kiosco, o al almacén; el idTipoArticulo indica a que tipo de artículo pertenece dicho artículo, y el idUsuario indica quien tiene el artículo; en caso de que en tránsito esté a true y el usuario sea el almacén significa que se está enviando desde el almacén hacia un kiosco; si el usuario fuese un kiosquero, entonces se estará devolviendo el artículo al almacén.

ArticuloVendido (idArticulo, idUsuario, PVP, fecha)

En esta tabla se almacenan los artículos que han sido vendidos por los kioscos, con el precio de venta al público y la fecha correspondiente a la venta.

ArticuloDescatalogado (idArticulo, fecha)

En esta tabla se almacenan los artículos que han sido descatalogados por el almacenista, la fecha en la que han sido retirados del catálogo. Los artículos caducados se consideran descatalogados.

ArticuloExtraviado (idArticulo, idUsuario, fecha)

En esta tabla se almacenan los artículos que han sido extraviados, es decir que no está en ninguna de las tablas anteriores.

TipoArticulo (idTipoArticulo, nombre, PCA, PVP, descatalogado, tipo, fecha)

En esta tabla se almacenan todos los tipos de artículo, es decir los correspondientes al catálogo del almacén; el PCA corresponde al Precio de Compra del Almacén de los artículos, mientras que el PVP es el Precio de Venta al Público; también se indica si dicho tipo de artículo está descatalogado o no, así como el tipo al que corresponde (chucherías, publicaciones o genéricos) y la fecha correspondiente a dicho tipo (caducidad o publicación) en caso de que sea necesario.

Reserva (idReserva, idUsuario, nombre, idTipoArticulo)

En esta tabla aparecen todas las reservas realizadas por los clientes a los kioscos.

Pedido (fecha, idUsuario, idTipoArticulo, cantidad, precio)

En esta tabla se almacenan todos los pedidos realizados por los kioscos al almacén; indicando la cantidad que desean y el precio por unidad de en el momento que se añadió el tipo de artículo al pedido. PCA+porcentajeCompra. Y la fecha nos indica cuando se recibirá el pedido.

Albaran (fecha, idUsuario, idArticulo, confirmado, precio)

En esta tabla se almacenan los albaranes generados por el almacén para abastecer los pedidos de los kioscos, la fecha indica cuando se va a recibir el pedido y el booleano confirmado está a false hasta que el kiosquero confirme la llegada de los artículos. El precio es del tipoArticulo al que pertenece el artículo, el cual se obtiene del pedido al que pertenezca el albarán.

AlbaranDevolucion (fecha, idUsuario, idArticulo, confirmado)

En esta tabla se almacenan los albaranes generados por el kiosco cuando quiere devolver un artículo al almacén. La fecha corresponde al día en el que un kiosco devuelve los artículos correspondientes al almacén mientras que el booleano confirmado indica si el almacén ha recibido dichos artículos y ha aceptado la devolución.

Funciones de la Base de Datos

hayMenosReservasQueArticulos (idTA, idU) -> Comprobaríamos que el número de filas de la tabla Reserva con el campo idTipoArticulo = idTA e idUsuario = idU , es menor que el número de filas de la tabla Artículo con el campo idTipoArticulo = idTA y idUsuario = idU. Si esto se verifica, devolvemos "true", en caso contrario, "false".

body:

- b:boolean
- b:= (Reserva.filas -> select (f1: fila | f1.idTipoArticulo = idTA and f1.idUsuario = idU) -> size () < Articulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA and f1.idUsuario = idU) -> size ())

damePVP (idTA) -> Consulta en la tabla TipoArticulo cuál es el PVP de un tipoArticulo con idTipoArticulo = idTA.

body:

TipoArticulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA) -> forAll (f1: fila | f1.PVP)

añadirArticuloVendido (idA, idU, PVP, fechaDeHoy) -> Inserta una fila en la tabla ArticuloVendido, con los parámetros de entrada de la función.

pre:

- not (ArticuloVendido.filas -> exists (f1: fila | f1.idArticulo = idA and f1.idUsuario = idU))
- Usuario.filas -> exists (f1: fila | f1.idUsuario = idU)

post:

ArticuloVendido.filas -> exists (f1: fila | f1.idArticulo = idA and f1.idUsuario = idU)

eliminarArticulo (idA) -> Elimina de la tabla Articulo la fila con idArticulo = idA.

pre:

```
Articulo.filas -> exists (f1: fila | f1.idArticulo = idA)
```

post:

```
not (Articulo.filas -> exists (f1: fila | f1.idArticulo = idA))
```

dameTipoArticulo (idA) -> Devuelve el campo idTipoArticulo, al cual corresponde el artículo con IdArticulo = idA de la tabla Artículo.

body:

```
Articulo.filas -> select (f1: fila | f1.idArticulo = idA) -> first ().idTipoArticulo
```

estaCatalogado (idTA) -> Devuelve el campo catalogado de la fila con idTipoArticulo = idTA de la tabla TipoArticulos.

pre:

```
TipoArticulo.filas -> exists (f1: fila | f1.idTipoArticulo = idTA)
```

body:

```
TipoArticulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA)->first().descatalogado
```

ponerADescatalogado(idTA)

body:

```
TipoAriculo.filas -> select (f1: fila | f1.idTipoArticulo = idTA) ->
first().descatalogado:= true
```

damePrecioCompraAlmacen (idTA) -> Devuelve el campo PCA de la fila con idTipoArticulo = idTA de la tabla TipoArticulos.

body:

```
TipoArticulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA) -> first ().PCA
```

añadirTipoArticuloAPedido (fech, idU, idTA, cant, prec) -> Añade una fila a la tabla Pedido, con campos: fecha = fech, idUsuario = idU idTipoArticulo = idTA, cantidad = cant y precio = prec

pre:

- not (Pedido.filas -> exists (f1: fila | f1.idUsuario = idU and f1.fecha = fech and f1.cantidad = cant and f1.precio = prec))
- Usuario.filas -> exists (f1: fila | f1.idUsuario = idU)
- TipoArticulo.filas -> exists (f1: fila | f1.idTipoArticulo = idTA)
- cant > 0
- n: integer
- n:= Pedido.filas -> size ()

post:

- Pedido.filas -> exists (f1: fila | f1.idUsuario = idU and f1.fecha = fech and f1.cantidad = cant and f1.precio = prec)
- Pedido.filas -> size () = n+1

dameCantidadPedido (fech, idU, idTA) -> Consulta en la tabla Pedido el campo cantidad de la fila con parámetros fecha = fech, idUsuario = idU e idTipoArticulo = idTA

body:

Pedido.filas -> select (f1: fila | f1.idUsuario = idU and f1.idTipoArticulo = idTA and f1.fecha = fech) -> first ().cantidad

damePrecioPedido (fech, idU, idTA) -> Consulta en la tabla Pedido el campo precio de la fila con parámetros fecha = fech, idUsuario = idU e idTipoArticulo = idTA

body:

Pedido.filas -> select (f1: fila | f1.idUsuario = idU and f1.idTipoArticulo = idTA and f1.fecha = fech) -> first ().precio

actualizarCantidadPedido (fech, idU, idTA, cant) -> Modifica en la tabla Pedido el campo cantidad de la fila con parámetros fecha=fech, idUsuario=idU e idTipoArticulo=idTA.

pre:

- Pedido.filas -> exists (f1: fila| f1.idUsuario = idU and f1.fecha = fech and f1.idTipoArticulo = idTA))
- cant >= 0

post:

Pedido.filas -> select (f1: fila | f1.idUsuario = idU and f1.fecha = fech and f1.precio = prec) -> first ().cantidad = cant

borraTipoArticuloAPedido (fecha, IdU, idTA) -> Borra en la tabla Pedido la fila con parámetros fecha = fech, idUsuario = idU e idTipoArticulo = idTA.

pre:

- Pedido.filas -> exists (f1: fila | f1.idTipoArticulo = idTA and f1.fecha = fecha and f1.idUsuario = IdU)
- n: integer
- n:= Pedido.filas -> size ()

post:

- not (Pedido.filas -> exists (f1: fila | f1.idTipoArticulo = idTA and fi.fecha = fecha and f1.idUsuario = IdU))
- Pedido.filas -> size () = n – 1

damePedidosDelDia () -> Devuelve una lista de tuplas de tres elementos: *<idTipoArticulo, SumCantidad, <IdUsuario, Cantidad, Precio>>* donde:

- *idTipoArticulo* es la referencia del artículo solicitado
- *SumCantidad* es la suma total de las cantidades pedidas por todos los kioscos de un TipoArticulo concreto
- una lista *(<IdUsuario, Cantidad, Precio>)*, con los kioscos que pidieron ese tipo de artículo y la cantidad pedida por cada uno de ellos:
  - *idUsuario* es el identificador del kiosco,
  - *Cantidad* es el número de tipoArticulos que solicitado.
  - *Precio:* es el PCA + porcentaje de compra del Tipo artículo

Para ello simplemente ha de consultar en la tabla Pedidos.

body:

Pedido.filas -> select (f1: fila | f1.fecha = fechaDehoy) -> forAll (f2: fila | f2.idTipoArticulo) -> forAll (idTA: idTipoArticulo | SumCantidad:= Pedido.filas -> select (f3: fila | f3.fecha = fechaDehoy and f3.idTipoArticulo = idTA) -> forAll (f4: fila | f4.cantidad) -> sum ())

dameInventario (L) -> Recorre la tabla Articulo y devuelve tuplas <idTipoArticulo, Cantidad> de todas aquellas filas con idUsuario igual al identificador del kiosco, idTipoArticulo igual a alguno de los distintos idTipoArticulo de la lista L (aquellos que han sido pedidos por los kioscos), y transito = false.

dameIdArticulo (idTA) -> Recorre la tabla Articulo y devuelve el IdArticulo de la primera fila cuyo campo idTipoArticulo = TA, transito = false y idUsuario = al idUsuario del almacén.

body:

Articulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA and f1.transito = false and f1.idUsuario = idAlmacen) -> first ().idArticulo

añadirEntradaAlbaran (IdKiosco, fech, id, prec) -> Añade en la tabla Albaran una fila con parámetros: fecha = fech, idUsuario = idKiosco, idArticulo = id, confirmado = false, precio = prec

pre:

- not (Albaran.filas -> exists (f1: fila | f1.idArticulo = id and f1.fecha = fecha and f1.idUsuario = IdKiosco and f1.precio = prec))
- Usuario.filas -> exists (f1: fila | f1.idUsuario = idKiosco)
- n: integer
- n:= Albaran.filas -> size ()

post:

- Albaran.filas -> exists (f1: fila | f1.idArticulo = id and f1.fecha = fecha and f1.idUsuario = IdKiosco and f1.precio = prec)
- Albaran.filas -> size () = n + 1

cambiarTransito (idA) -> En la tabla Articulo, cambia el campo transito de la fila con idArticulo = idA.

pre:

- x: boolean
- x:= Articulo.filas -> select (f1: fila | f1.idArticulo = idA) -> first ().transito

post:

not (x)

cambiarUsuarioArticulo (idA,idU) -> En la tabla Articulo, cambia el campo idUsuario a idU, de la fila con idArticulo = idA.

body:

Articulo.filas -> select (f1: fila | f1.idArticulo = idA) -> first ().idUsuario:= idU

confirmarArticulo (idA) -> En la tabla Albarán, pone el campo confirmado, de la fila con idArticulo = idA y confirmado =”false”, a “true”

pre:

Albaran.filas -> exists (f1: fila | f1.idArticulo = idA and f1.confirmado = false)

body:

Albaran.filas -> select (f1: fila | f1.idArticulo = idA) -> first ().confirmado:= true

añadirArticuloExtraviado (idA, idU, fechaDeHoy) -> Añade en la tabla ArticuloExtraviado una fila con los campos idArticulo = idA, idUsuario = idU, y fecha = fechaDeHoy

pre:

- Articulo.filas -> exists (f1: fila | f1.idArticulo = idA)
- Usuario.filas -> exists (f1: fila | f1.idUsuario = idKiosco)
  not (ArticuloExtraviado.filas -> exists (f1: fila | f1.idArticulo = idA and f1.idUsuario = idU and f1.fecha = fechaDeHoy))
- n:integer
- n:=Articulo.filas -> size ()

post:

- ArticuloExtraviado.filas -> exists (f1: fila | f1.idArticulo = idA and f1.idUsuario = idU and f1.fecha = fechaDeHoy)
- Articulo.filas -> size() = n + 1

añadirReserva (idU, nombre, idTA) -> Consulta cuál es el idReserva más alto de la tabla Reserva. Inserta una nueva fila con campos: idReserva = max (idReserva) + 1, idUsuario = idU, nombre = nombr, idTipoArticulo = idTA

pre:

- x: integer
- x:= Reserva.filas -> select (f1: fila | f1.idU = idU and f1.nombre = nombre and f1.idTipoArticulo = idTA) -> size ()

post:

Reserva.filas -> select (f1: fila | f1.idU = idU and f1.nombre = nombre and f1.idTipoArticulo = idTA) -> size () = x + 1

añadirArticuloAlbaranDevolucion (fecha, idU, idA) -> Añade en la tabla AlbaranDevolución una fila con parámetros: fecha = fech, idUsuario = idU, idArticulo = idA, confirmado = false.

pre:

- not (AlbaranDevolucion.filas -> exists (f1: fila | f1.idUsuario = idU and f1.idArticulo = idArticulo and f1.fecha = fecha))
- n: integer
- n:= AlbaranDevolucion.filas -> size ()

post:

- AlbaranDevolucion.filas -> exists (f1: fila | f1.idUsuario = idU and f1.idArticulo = idArticulo and f1.fecha = fecha)
- AlbaranDevolucion.filas -> size () = n + 1

damePrecioArticulo (idA) -> Busca en la tabla AlbaranDevolución cual es el campo idUsuario de la fila con idarticulo = idA, y el campo "confirmado" = false.

body:

- x: idUsuario
- x:= AlbaranDevolucion.filas -> select (f1: fila | f1.idArticulo = idA and f1.confirmado = false) -> first ().idUsuario
- Albaran.filas -> select (f1: fila | f1.idArticulo = idA and f1.idUsuario = x) -> last().precio

A continuación, busca en la tabla Albaran la fila con campos idArticulo=idA, idUsuario= al dato hallado anteriormente, y, la fecha la última que haya (de entre las filas que cumplan los dos primeros requisitos). Devuelve el campo precio de dicha fila.

dameUsuario (idA) -> Busca en la tabla AlbaranDevolución cual es el campo idUsuario de la fila con idarticulo = idA, y el campo "confirmado" = false.

body:

AlbaranDevolucion.filas -> select (f1: fila | f1.idArticulo = idA and f1.confirmado = false) -> first ().idUsuario

confirmarLlegadaArticulo (idA) -> En la tabla AlbaranDevolucion, pone el campo confirmado, de la fila con idArticulo = idA y confirmado = "false" , a "true"

pre:

Albaran.filas -> exists (f1: fila | f1.idArticulo = idA and f1.confirmado = false)

body:

Albaran.filas -> select (f1: fila | f1.idArticulo = idA) -> first ().confirmado:= trae

eliminarArticulos(idTA) -> En la tabla Articulos, elimina todos los artículos que pertenezcan a idTipoArticulo.

post:
Articulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA) -> isEmpty ()

eliminarPedido(fecha,idUsuario) -> Elimina todas las filas de la tabla Pedidos, correspondientes a ese usuario y con la fecha señalada

post:
Pedido.filas –> select (f1: fila | f1.fecha = fecha and f1.idUsuario = idU) -> isEmpty()

usuarioValido(idU)-> Comprueba en la tabla Usuario, que no exista ninguna fila cuyo campo idUsuario sea igual a idU. Devuelve false en caso de que si exista.

pre:
not (Usuario.filas -> exists (f1: fila | f1.idUsuario = idU))

anadeUsuario(idUsuario,idUsuario,kiosco,0,0)-> Inserta en la tabla Usuario una fila con los parámetros de entrada de la función.

post:
Usuario.filas -> exists (f1: fila | f1.idUsuario = idU and f1.saldo = 0 and f1.saldoVirtual = 0 and f1.contraseña = idU

damePedidos(idTipoArticulo)-> Devuelve una lista con todas las filas de la tabla Pedidos que contienen el campo idTipoArticulo.

body:
Pedido.filas -> select (f1: fila | f1.idTipoArticulo = ID and f1.fecha >= fechaDeHoy)

borrarTipoArticuloAPedido(fecha,idUsuario,idTipoArticulo) -> Eliminamos todas las filas de la tabla Pedidos que coincidan con los campos dados.

post:
not (Pedido.filas -> exists (f1:fila | f1.fecha = fecha and f1.idUsuario = idU and f1.idTipoArticulo = idTA))

añadirArticuloDescatalogado(idTA,fechaDeHoy) -> Inserta en la tabla ArticuloDescatalogado todos los idArticulos de la tabla Artículos cuyo campo idTipoArticulo=idTA, incluyendo la fecha de hoy.

post: ArticuloDescatalogado.filas -> exists (f1:fila | f1.fecha = fechaDeHoy and f1.idTipoArticulo = idTA))

eliminarArticulosDescatalogados(idTA,idAlmacen) -> Elimina de la tabla Articulo todos las filas con campo idTipoArticulo=idTA , idUsuario=idAlmacen y transito = false. También elimina todas las filas con campo idTipoArticulo=idTA, idUsuario <> idAlmacen y transito = true. Además elimina estos últimos artículos del albarán devolución.

post:
not (Articulo.filas -> exists (f1: fila | (f1.idTipoArticulo = idTA and f1.idUsuario = idAlmacen and transito = false) or (f1.idTipoArticulo = idTA and f1.idUsuario <> idAlmacen and transito = true and (not (AlbaranDevolucion.filas -> exists (f2: fila | (f2.idArticulo = f1.idArticulo and f2.idUsuario = f1.idUsuario and f2.confirmado = false)))))))

eliminarArticulosCaducados(idTA) -> Elimina de la tabla Articulo todos las filas con campo idTipoArticulo=idTA. Si tránsito = true e idUsuario = idAlmacen se elimina del albarán y se le añade el precio del articulo al saldoVirtual del Usuario que va a recibirlo. En caso de que idUsuario <> idAlmacen solo se elimina del albarán.

post:
Articulo.filas -> select (f1: fila | f1.idTipoArticulo = idTA) -> isEmpty () and (Albaran.filas -> select (f2: fila | f2.idArticulo.idTipoArticulo = idTA and f2.confirmado = false) -> isEmpty () or AlbaranDevolucion.filas -> select (f3: fila | f3.idArticulo.idTipoArticulo = idTA and f3.confirmado = false) -> isEmpty ())

existeTipoArticulo(nombre,tipo,fecha) -> Comprueba que no exista ninguna fila en la tabla TipoArticulo con los parámetros de entrada seleccionados.

body:
TipoArticulo.filas -> exists (f: fila | f.idTipoArticulo = nombre and f.fecha = fecha and f.tipo = tipo)

añadirTipoArticulo(nombre, PCA, PVP, tipo, fecha) -> Inserta una fila en la tabla TipoArticulo con los parámetros insertados

pre:
not (existeTipoArticulo (idTA, tipo, fecha))

post:
TipoArticulo.filas -> exists (f: fila | f.idTipoArticulo = idTA and f.fecha = fecha and f.tipo = tipo and f.PVP = pvp and f.PCA = pca)

Descripción de Métodos

Diagramas de Actividades

Diagrama de actividades de Abrir Sesión – DA1

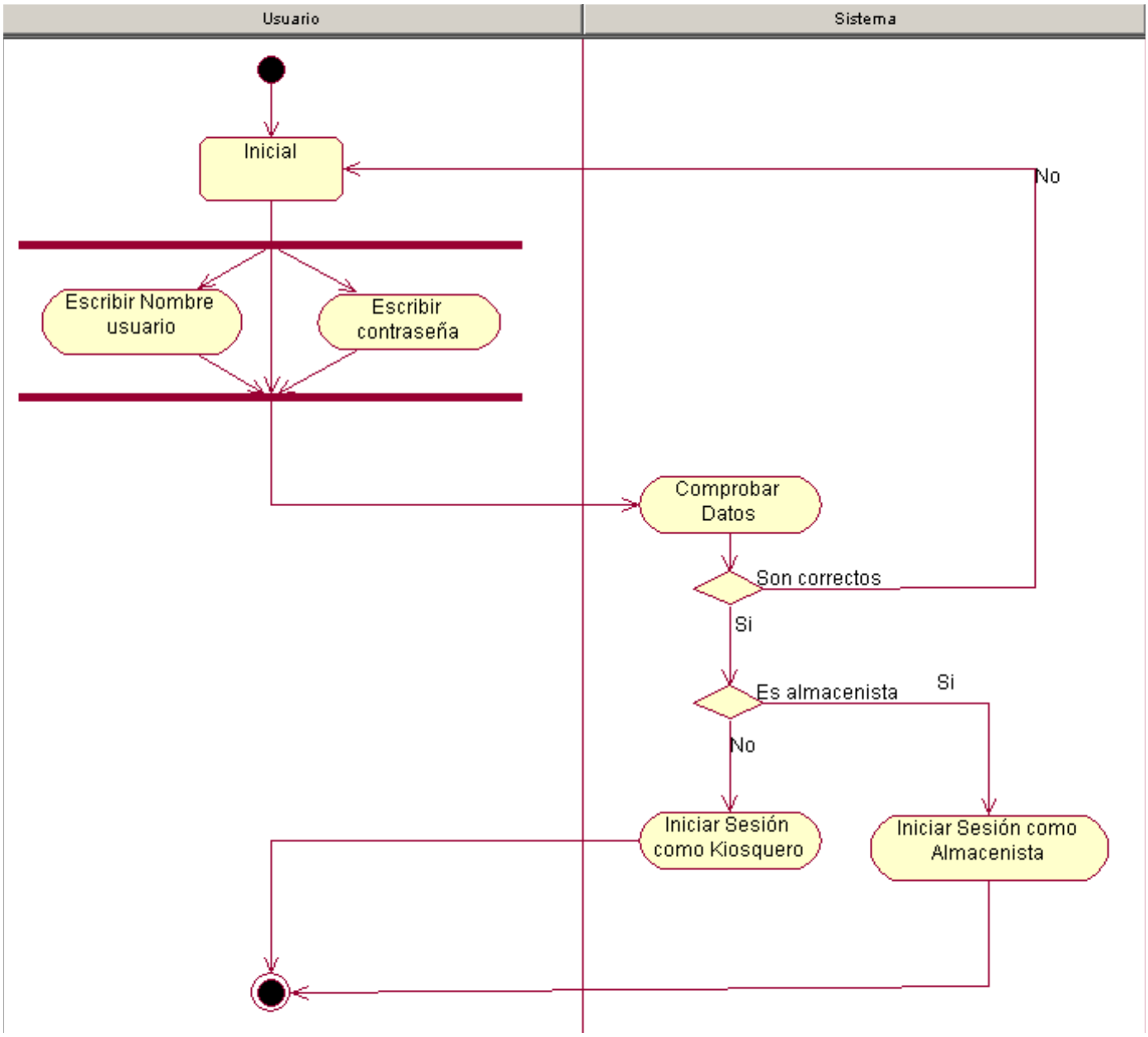


Diagrama de actividades de Cambiar Contraseña – DA2

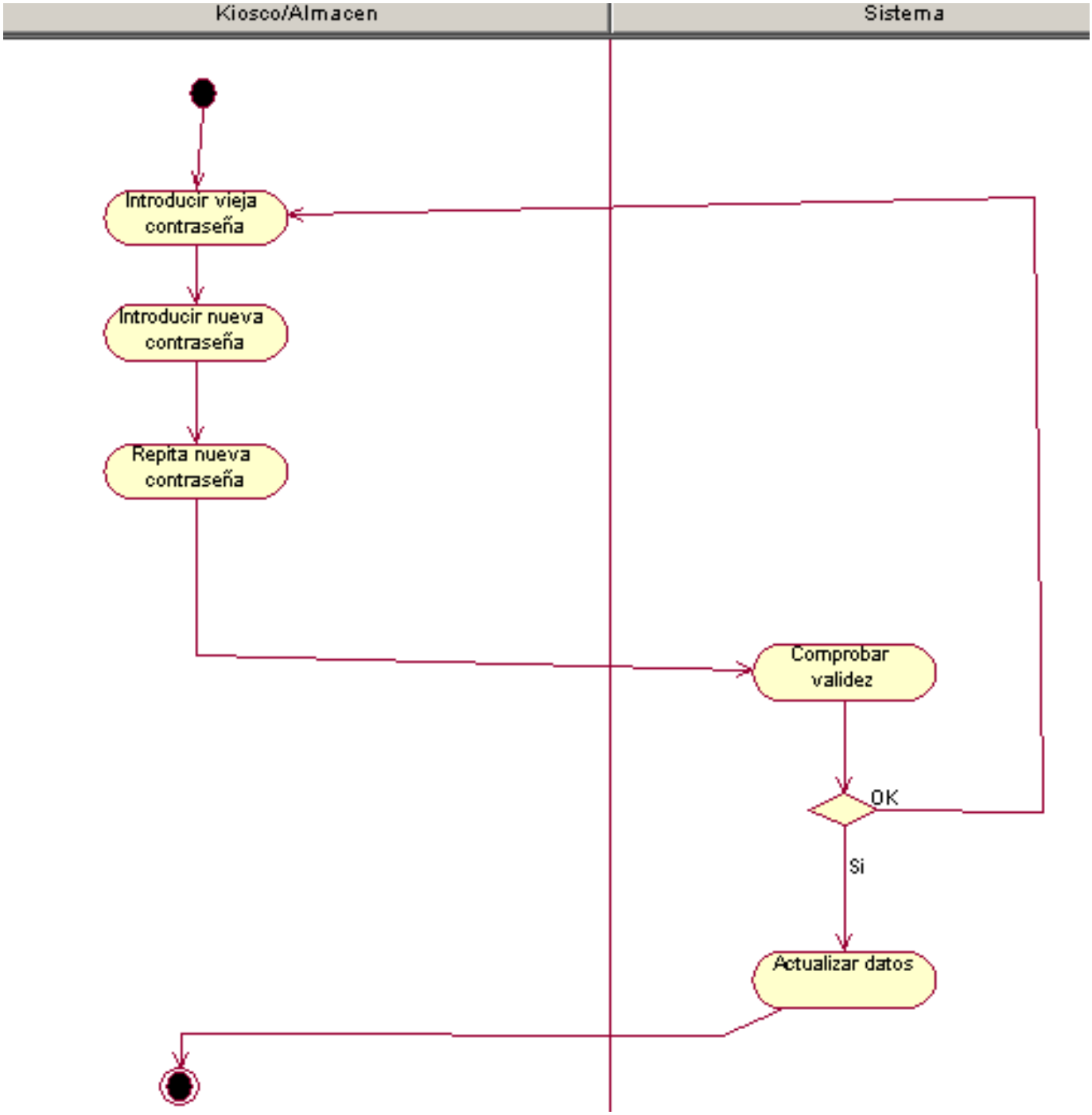


Diagrama de actividades de Cerrar Sesión – DA3

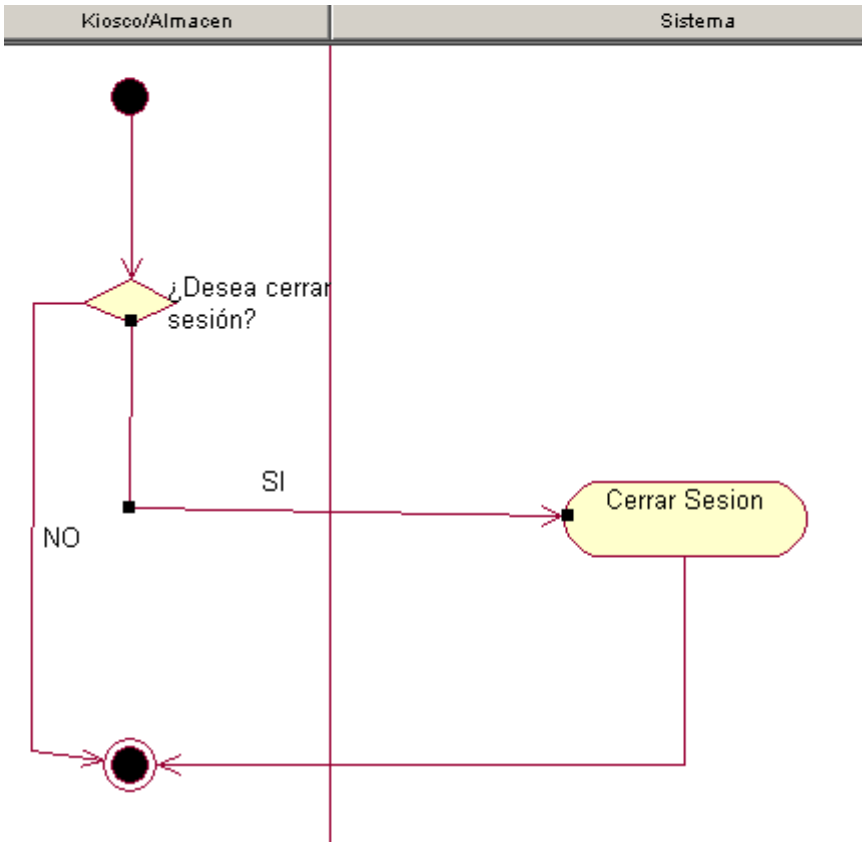


Diagrama de actividades de Dar de alta Kiosquero – DA4

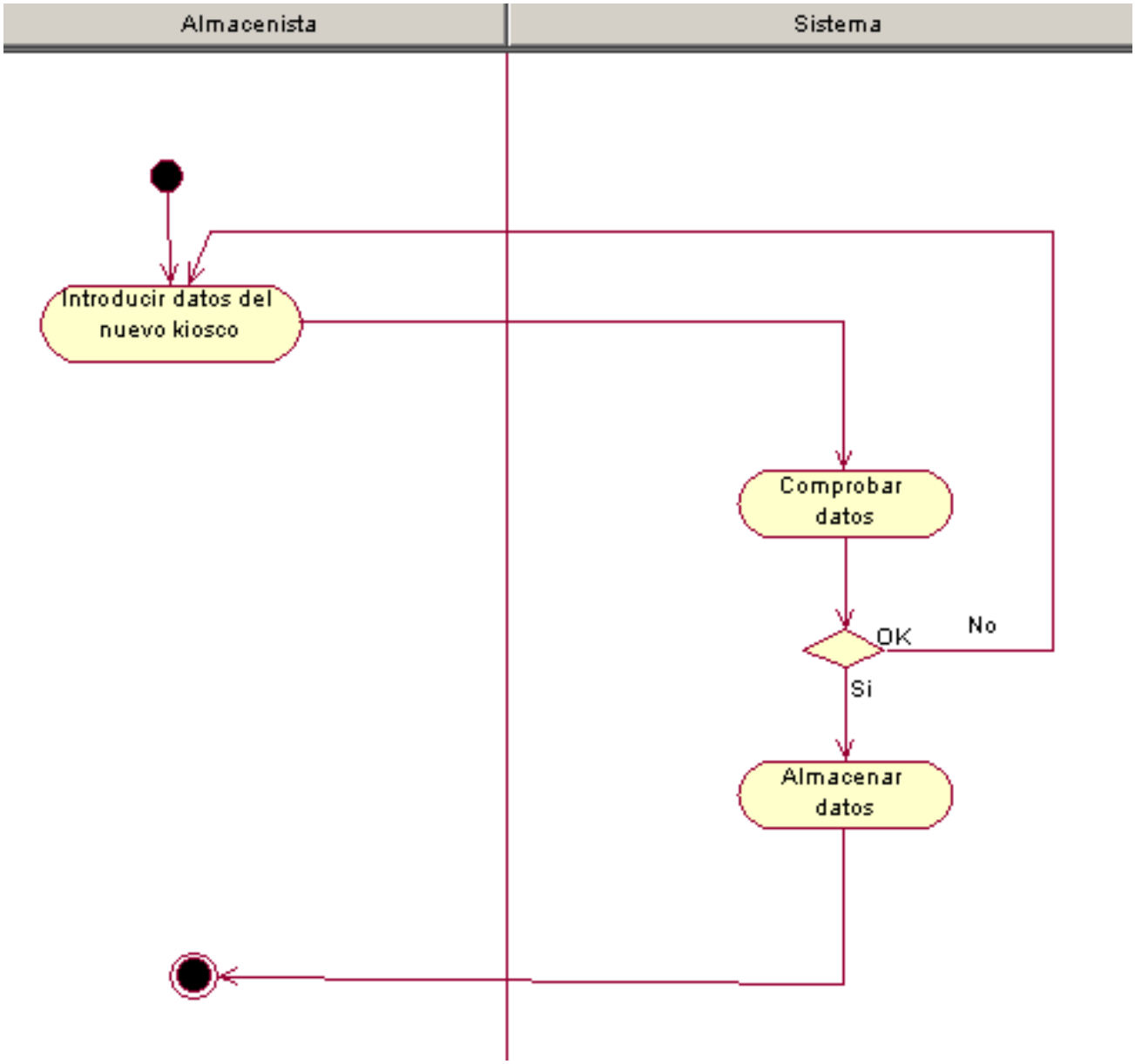


Diagrama de actividades de Dar de baja Kiosquero – DA5

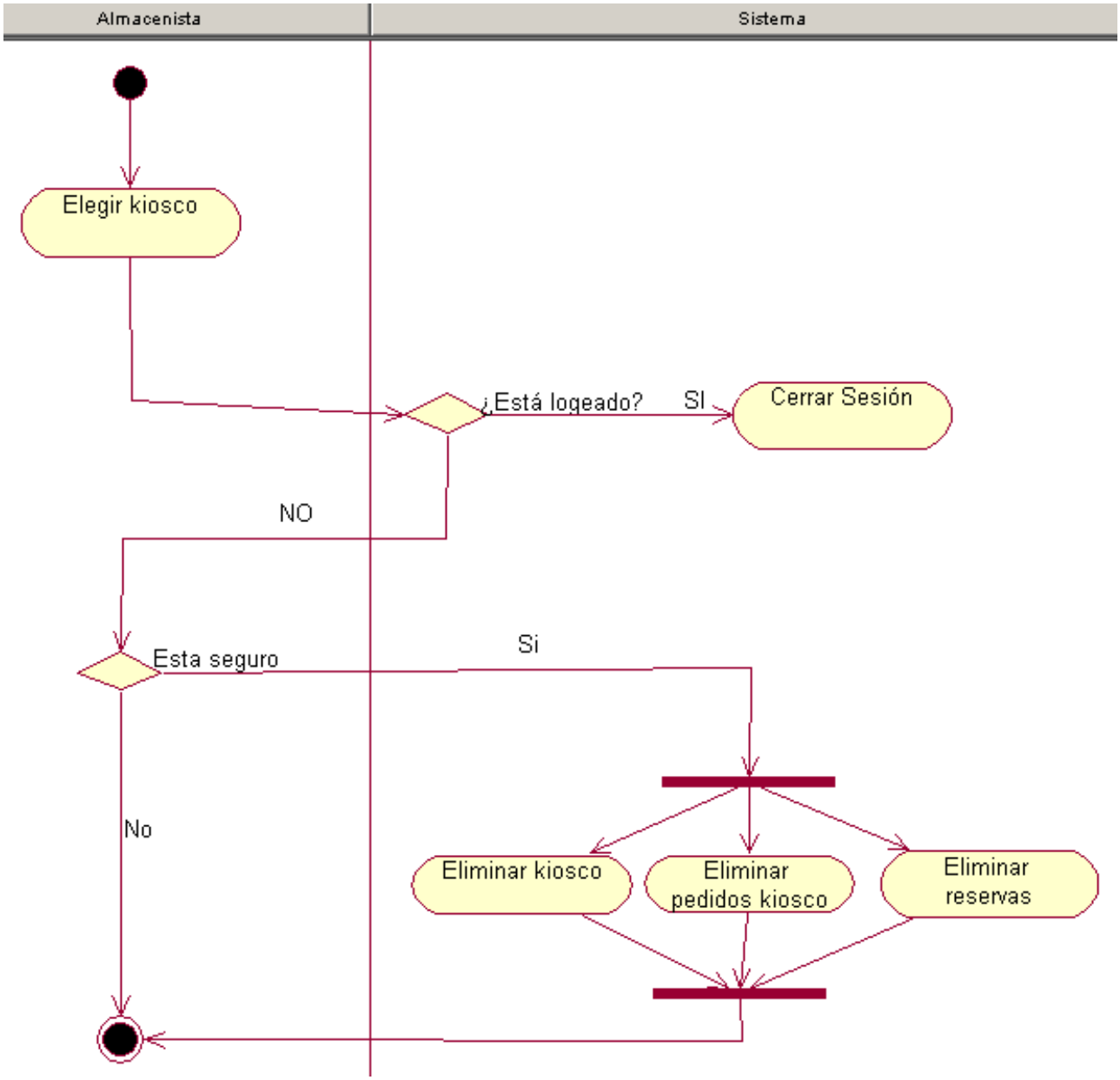


Diagrama de actividades de Reservar Articulo a Cliente – DA6

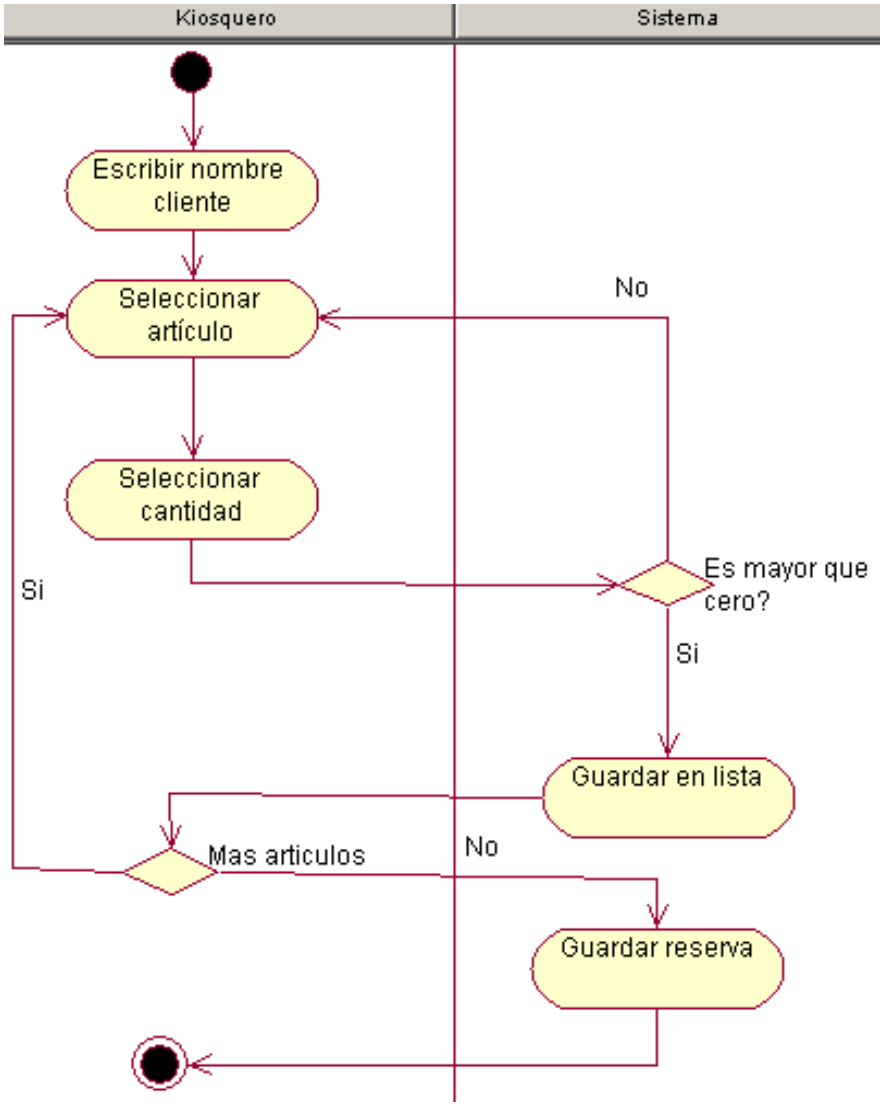


Diagrama de actividades de Vender – DA7

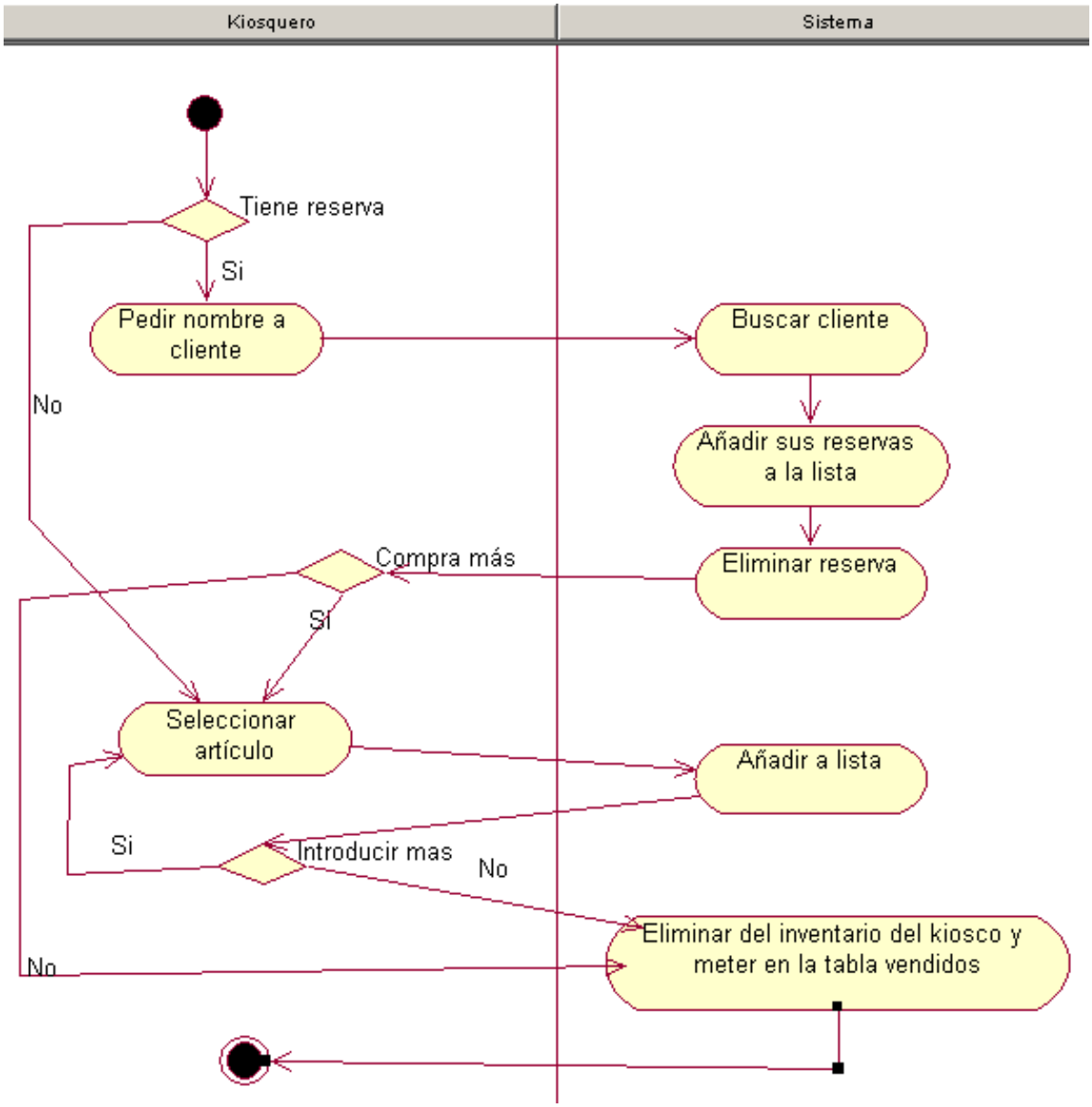


Diagrama de actividades de Hacer Pedido – DA9

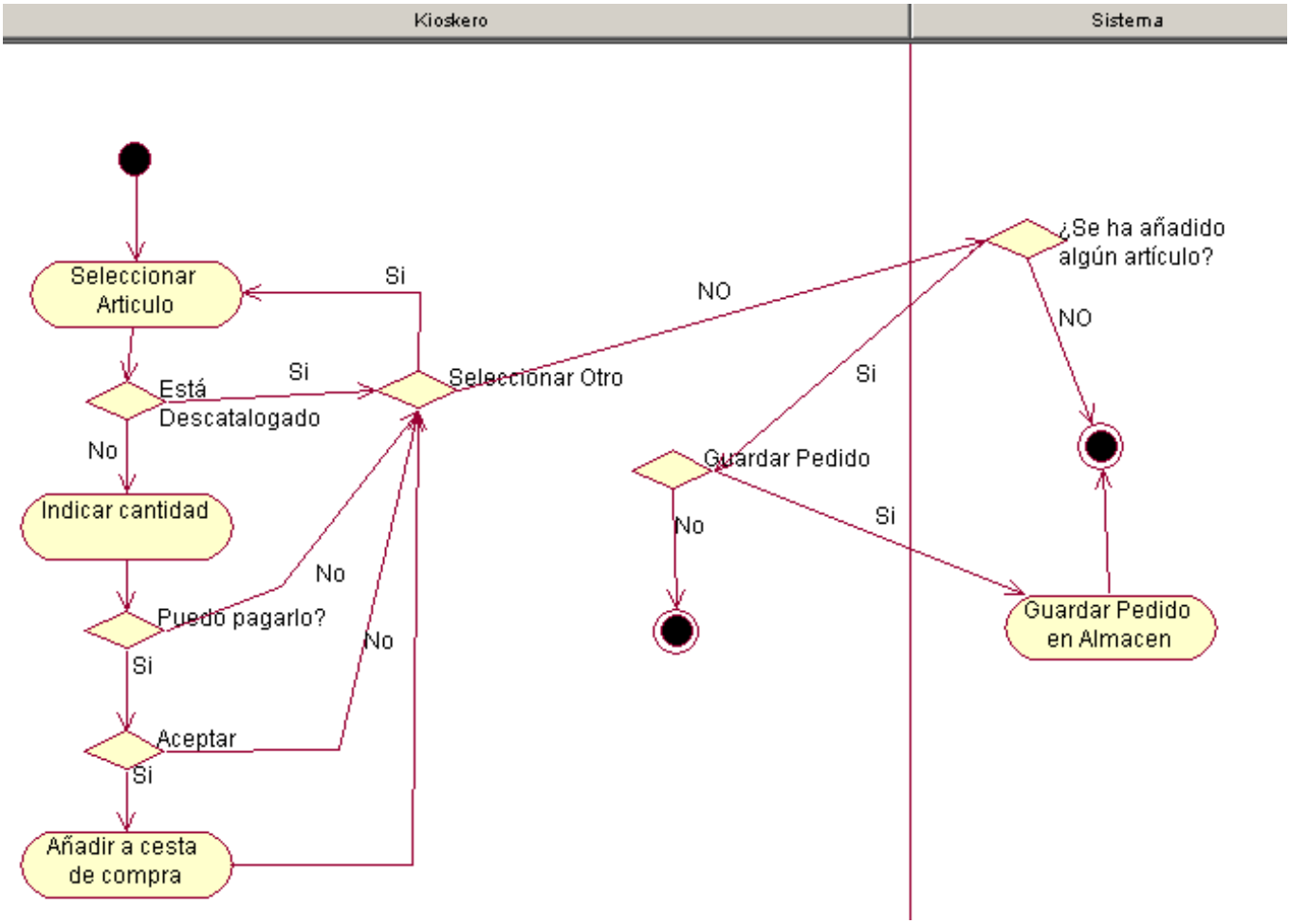


Diagrama de actividades de Modificar Catálogo – DA10

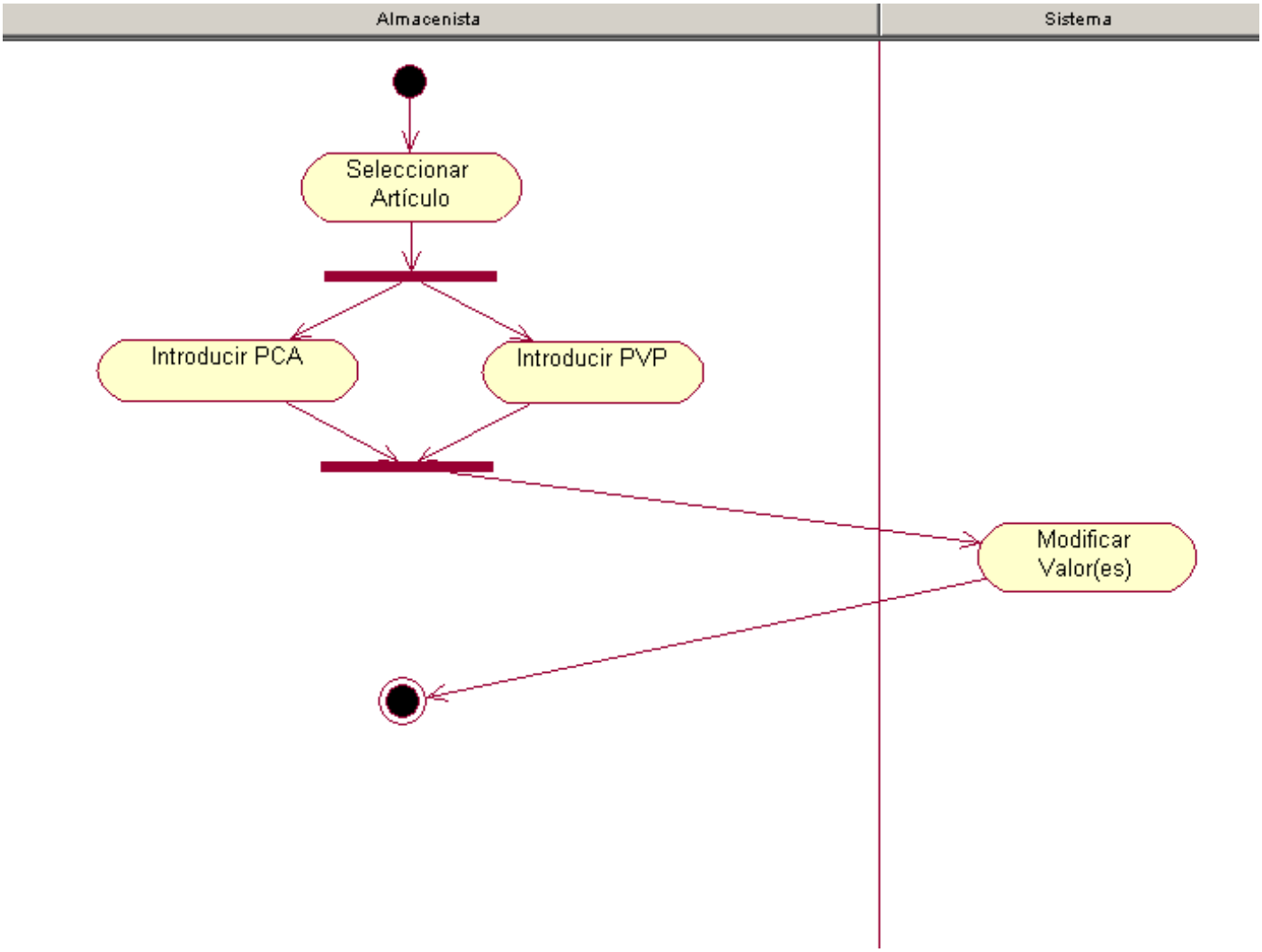
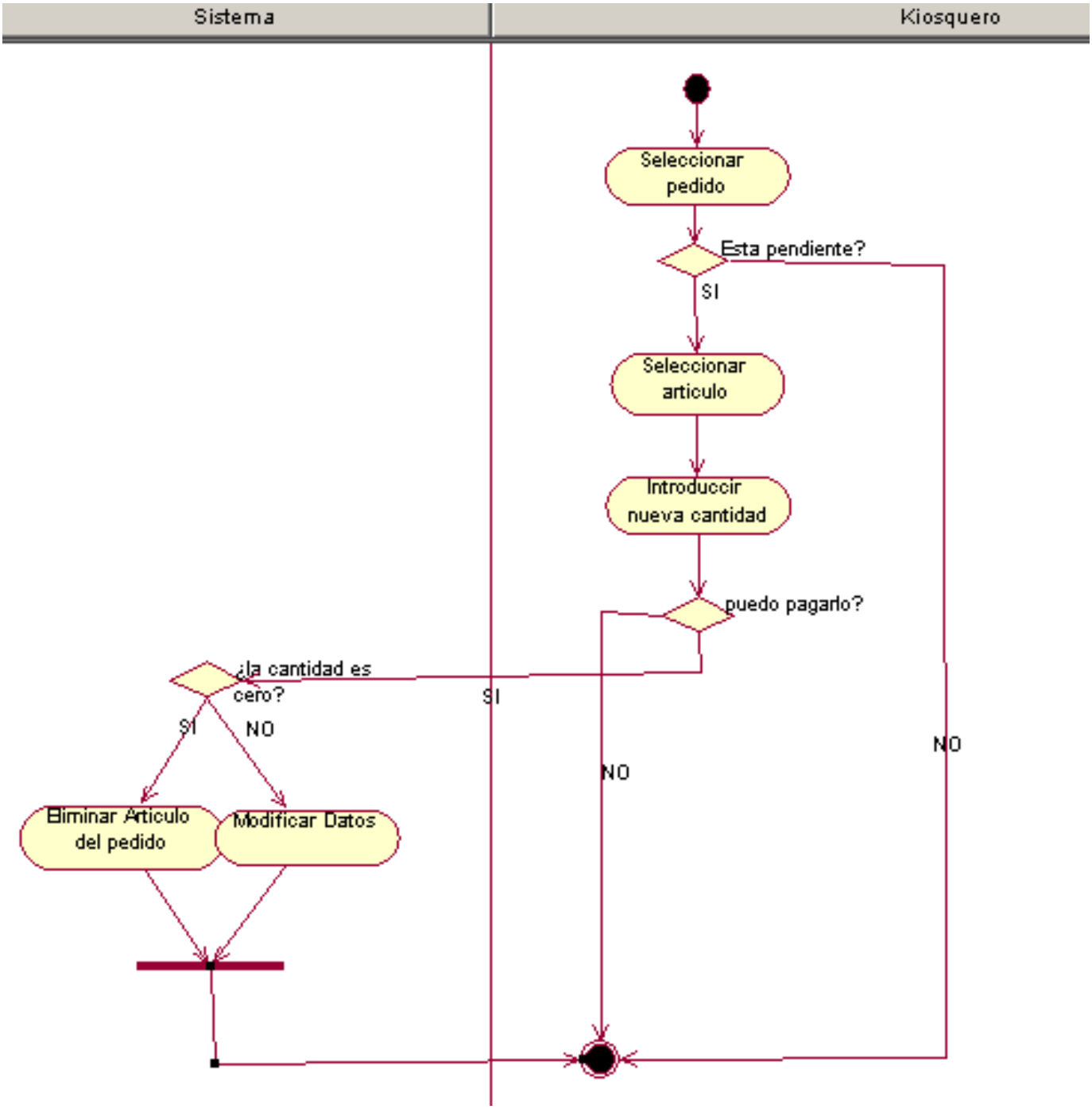


Diagrama de actividades de Modificar Pedido – DA11



Diagramas de Secuencias

Diagrama de Secuencias de vender() – DS1

El método vender se llama desde la interfaz gráfica del kiosco. El atributo idArticulo corresponde a un artículo existente en el inventario del kiosco y que esté en transito a false. IdTipoArticulo es el identificador su tipo artículo.

Para que se pueda vender un artículo es necesario que el nº de reservas de su tipo artículo sea inferior al nº de artículos del inventario del kiosco para asegurarse que ningún cliente que haya reservado un artículo (en realidad se reserva un tipo artículo) se quede sin él. En caso de que no fuera así, el método vender devolvería false y no se realizaría la venta.

Para realizar la venta se consulta el PVP del artículo y se suma al saldo y saldo virtual del kiosco. También se añade en la tabla ArticuloVendido constancia de esa acción.

Finalmente se elimina el artículo del inventario.

Nota: sumaPrecioaSaldo y sumaPrecioaSaldoVirtual modifican sus atributos del objeto kiosco y de la tabla Usuario en la BBDD.

Restricciones OCL:

Context Kiosco :: vender (referencia idArtículo, ID idTipoArtículo)

pre:

- self.miStock -> select (a1: Articulo | a1.idArticulo = referencia and (a1.transito = false)) -> size () = 1
- self.miStock -> select (a1: Articulo | a1.idArticulo = referencia) -> first ().miTipo = ID

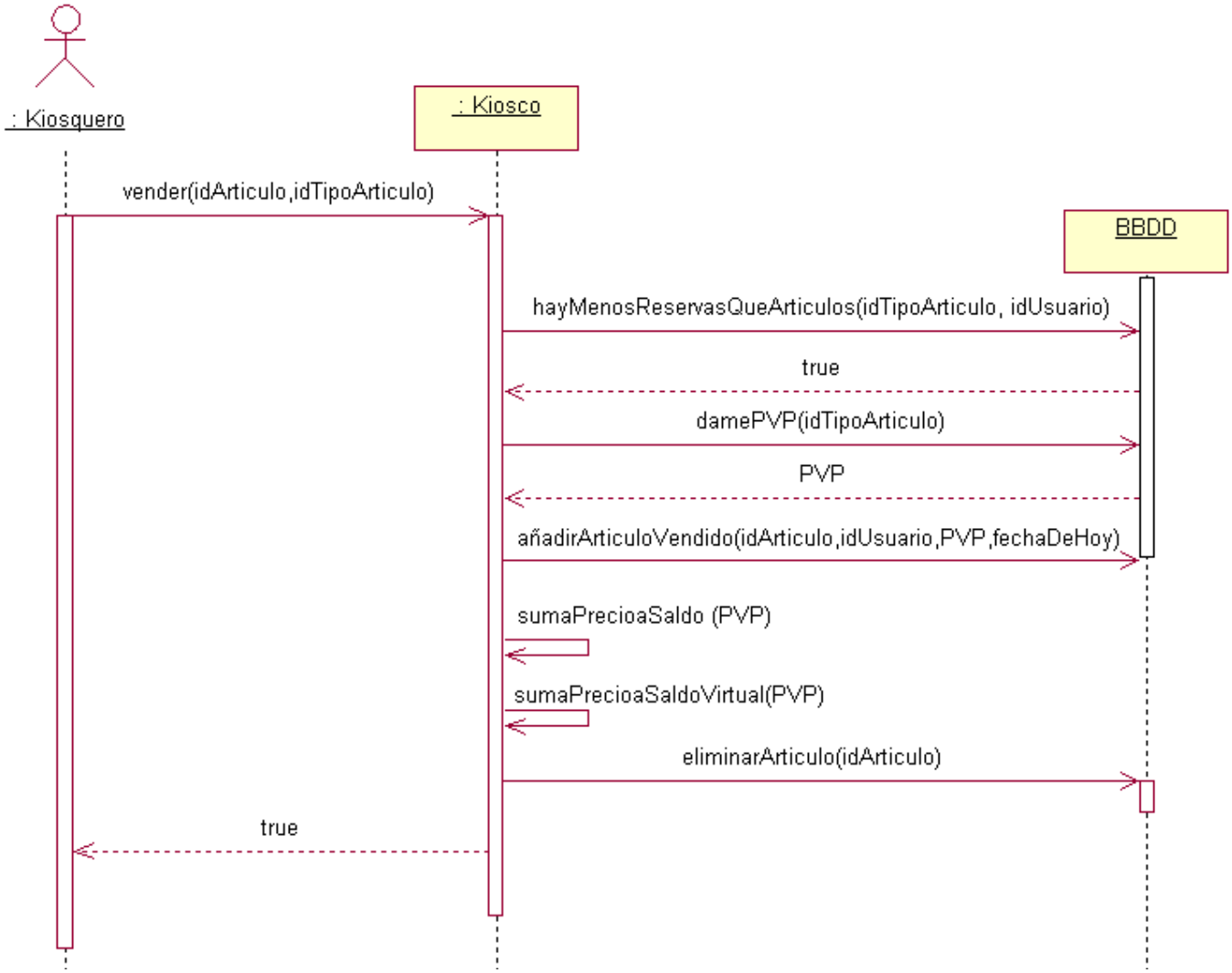

: Kiosquero
: Kiosco
BBDD
vender(idArticulo,idTipoArticulo)
hayMenosReservasQueArticulos(idTipoArticulo, idUsuario)
true
damePVP(idTipoArticulo)
PVP
añadirArticuloVendido(idArticulo,idUsuario,PVP,fechaDeHoy)
sumaPrecioaSaldo (PVP)
sumaPrecioaSaldoVirtual(PVP)
eliminarArticulo(idArticulo)
true

Diagrama de Secuencias de venderReserva() – DS2

El método venderReserva se llama desde la interfaz gráfica del kiosco. El atributo idReserva corresponde a un identificador de una reserva del kiosco. IdTipoArticulo es el identificador del tipo artículo de la reserva. IdArticulo es el identificador del artículo que se va a vender.

Para que se pueda vender una reserva es necesario que el tipo del artículo sea igual que el de la reserva. En caso de que no fuera así el método venderReserva devolvería false y no se realizaría la venta.

Después de realizar la comprobación hay que borrar la reserva para asegurarse de que haya menos reservas que artículos. Y finalmente se llama al método vender para realizar la venta en cuestión.

Restricciones OCL:

Context Kiosco :: borrarReserva (ID idReserva)

pre:

```
self.misReservas -> exists (r1: Reserva | r1.idReserva = ID)
```

post:

```
not (self.misReservas -> exists (r1: Reserva | r1.idReserva = ID))
```

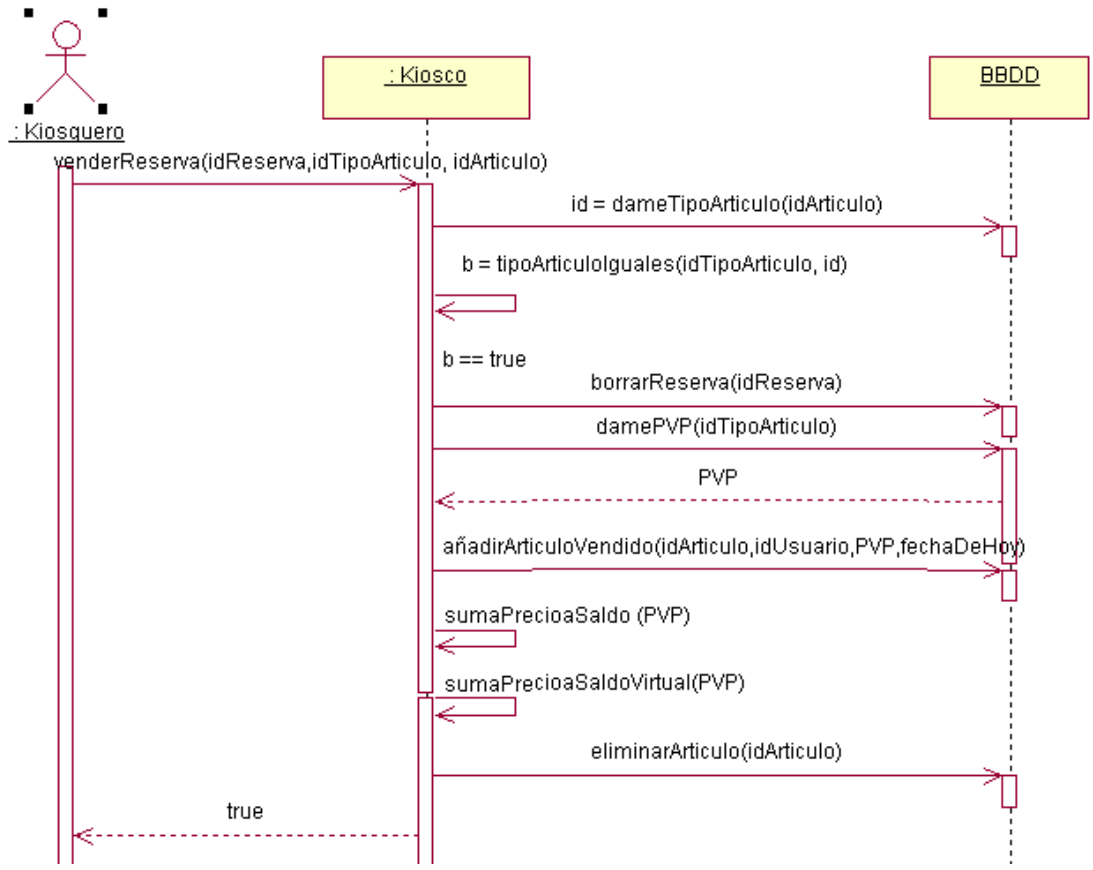


Diagrama de Secuencias de añadirTipoArtículoAPedido() – DS3

El método añadirTipoArticuloAPedido se llama desde la interfaz gráfica del kiosco cuando se va a crear un pedido o añadir un nuevo tipo artículo.

El atributo fecha indica la fecha de llegada del pedido al kiosco. El atributo idTipoArticulo es el identificador del tipo articulo a añadir y cantidad el número de artículos que desea recibir el kiosquero.

Antes de poder añadir cualquier tipo artículo hay que asegurarse que no está registrado el pedido, es decir, que no hayan pasado las 23:00 del día anterior a la fecha de entrega.

Si estuviese registrado el pedido, el método devolvería false y no realizaría la operación.

Después comprueba que el tipo artículo está catalogado. En realidad, la interfaz nunca dejaría seleccionar un tipo que no esté catalogado, pero puede darse el caso que después de seleccionarlo el almacenista lo descatalogue, por lo que es necesario volverlo a comprobar.

Si no estuviera catalogado el método devolvería false y no realizaría la operación.

Una vez realizadas estas comprobaciones, se solicita a la BBDD el PCA del tipo artículo y se verifica si hay saldo virtual suficiente. Con esto nos aseguramos que el saldo nunca va a ser negativo.

En caso de que no fuera así, el método devolvería false y no realizaría la operación.

Por último se añade al pedido en la BBDD y se resta del saldo virtual el precio correspondiente.

Nota: restaPrecioaSaldoVirtual modifica saldoVirtual del objeto kiosco y de la BBDD.

Restricciones OCL:

Context Kiosco :: anadirTipoArticuloAPedido (fecha Fecha, ID idTipoArticulo, cantidad Natural)

pre:

- self.misPedidos -> exists (p1: Pedido | p1.fecha = fecha)
  fecha -> getFechaActual ()
- not (self.misPedidos -> exists (p1: Pedido | p1.fecha = fecha and p1.articulosPedidos ->
  exists (t1: tipoArticulo | t1.idTipoArticulo = ID))

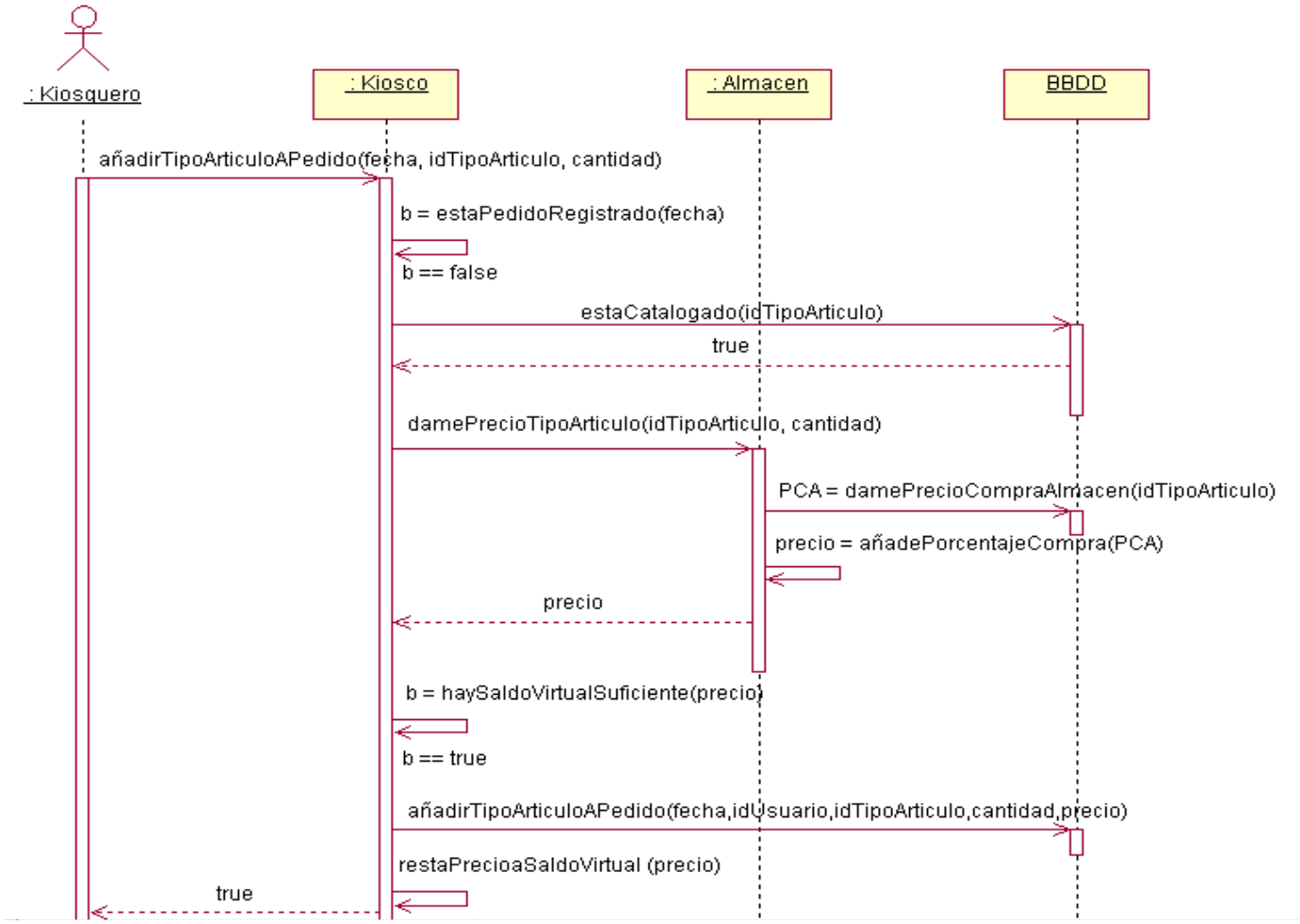

: Kiosquero
: Kiosco
: Almacen
BBDD
añadirTipoArticuloAPedido(fecha, idTipoArticulo, cantidad)
b = estaPedidoRegistrado(fecha)
b == false
estaCatalogado(idTipoArticulo)
true
damePrecioTipoArticulo(idTipoArticulo, cantidad)
PCA = damePrecioCompraAlmacen(idTipoArticulo)
precio = añadePorcentajeCompra(PCA)
precio
b = haySaldoVirtualSuficiente(precio)
b == true
añadirTipoArticuloAPedido(fecha,idUsuario,idTipoArticulo,cantidad,precio)
restaPrecioaSaldoVirtual (precio)
true

Diagrama de Secuencias de modificarPedido() (cantidad > 0) – DS4a

El método modificarPedido se llama desde la interfaz gráfica cuando se quiere cambiar la cantidad de un tipo artículo de un pedido.

El atributo fecha corresponde a un pedido del kiosco, idTipoArticulo es el identificador del tipo artículo y cantidad es el nº de artículos que desea recibir el kiosco. En este diagrama consideramos el caso en que cantidad sea mayor que cero.

Primero verificamos que el tipo artículo esta catalogado y que el pedido no está registrado, en caso contrario el método devolvería false y no se realizaría la acción.

Después solicitamos la cantidad antigua del tipo artículo y el precio. A continuación calculamos la diferencia de precio entre la nueva y la antigua, esta cantidad puede ser positiva o negativa en función de si la cantidad es mayor o menor que la antigua. Después comprobamos si tenemos saldo virtual suficiente y en caso positivo se le resta.

Si no hubiese suficiente saldo virtual modificar pedido devolvería false y no realizaría la operación.

Restricciones OCL:

Context Kiosco :: modificarPedido (fecha Fecha, ID idTipoArticulo, cantidad Natural)

pre:

- fecha -> getFechaActual () and (23:00 > getHoraActual ())
- self.misPedidos -> exists (p1: Pedido | p1.fecha = fecha and p1.articulosPedidos -> exists (t1: TipoArticulo | t1.idTipoArticulo = ID))

post:

- self.ArticulosPedidos -> forAll (t1:TipoArticulo | t1.cantidad > 0)
- self.ArticulosPedidos -> notEmpty ()

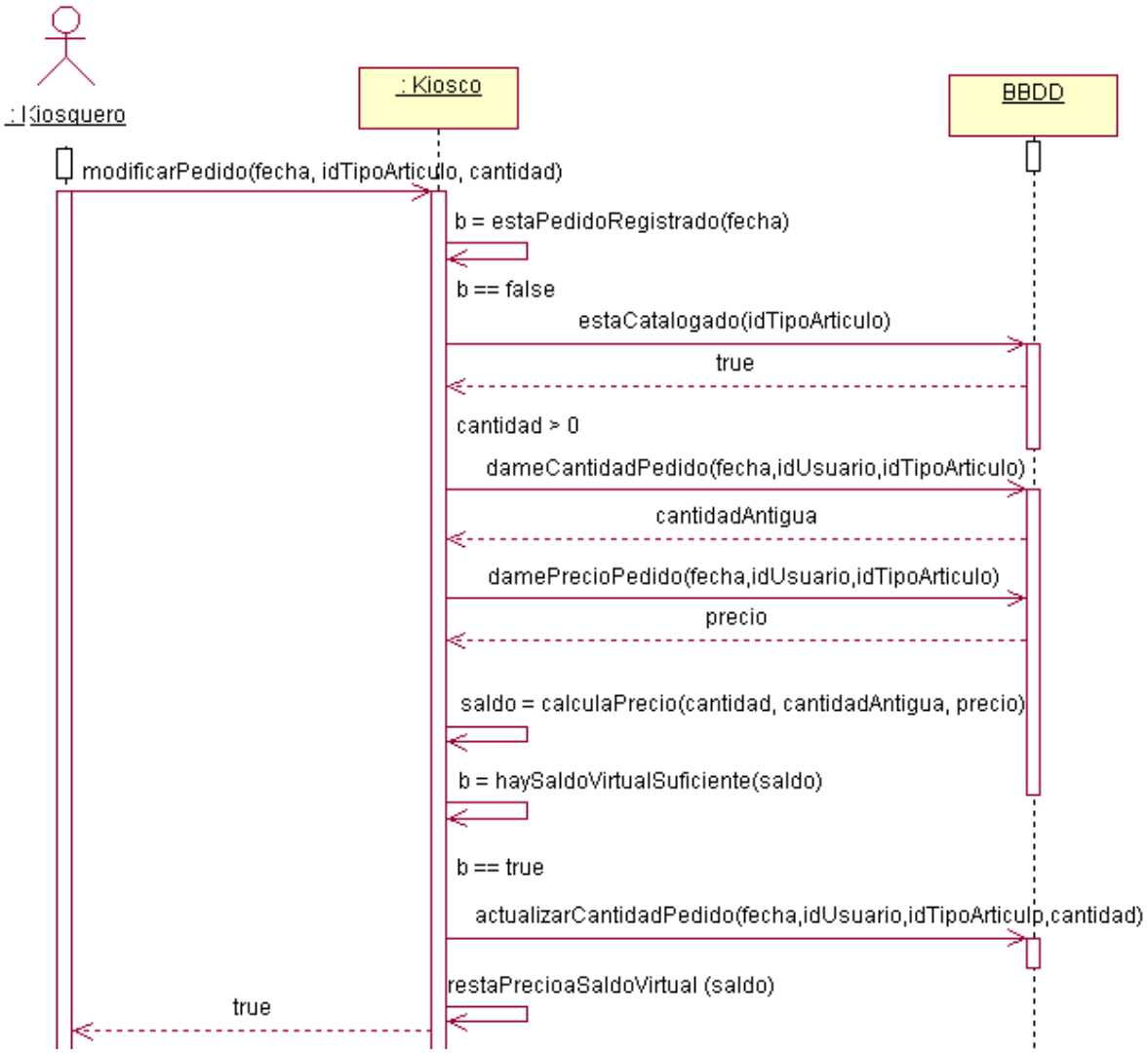

: Kiosquero
: Kiosco
BBDD
modificarPedido(fecha, idTipoArticulo, cantidad)
b = estaPedidoRegistrado(fecha)
b == false
estaCatalogado(idTipoArticulo)
true
cantidad > 0
dameCantidadPedido(fecha,idUsuario,idTipoArticulo)
cantidadAntigua
damePrecioPedido(fecha,idUsuario,idTipoArticulo)
precio
saldo = calculaPrecio(cantidad, cantidadAntigua, precio)
b = haySaldoVirtualSuficiente(saldo)
b == true
actualizarCantidadPedido(fecha,idUsuario,idTipoArticulo,cantidad)
restaPrecioaSaldoVirtual (saldo)
true

Diagrama de Secuencias de modificarPedido() (cantidad == 0) – DS4b

El método modificarPedido se llama desde la interfaz gráfica cuando se quiere cambiar la cantidad de un tipo artículo de un pedido.

El atributo fecha corresponde a un pedido del kiosco, idTipoArticulo es el identificador del tipo artículo y cantidad es el nº de artículos que desea recibir el kiosco. En este diagrama consideramos el caso en que cantidad es cero, es decir, se pretende eliminar el tipo artículo del pedido.

Primero verificamos que el tipo artículo esta catalogado y que el pedido no está registrado, en caso contrario el método devolvería false y no se realizaría la acción.

Después solicitamos la cantidad antigua del tipo artículo y el precio y se lo sumamos a saldoVirtual.
Por último se borra el tipo artículo del pedido.

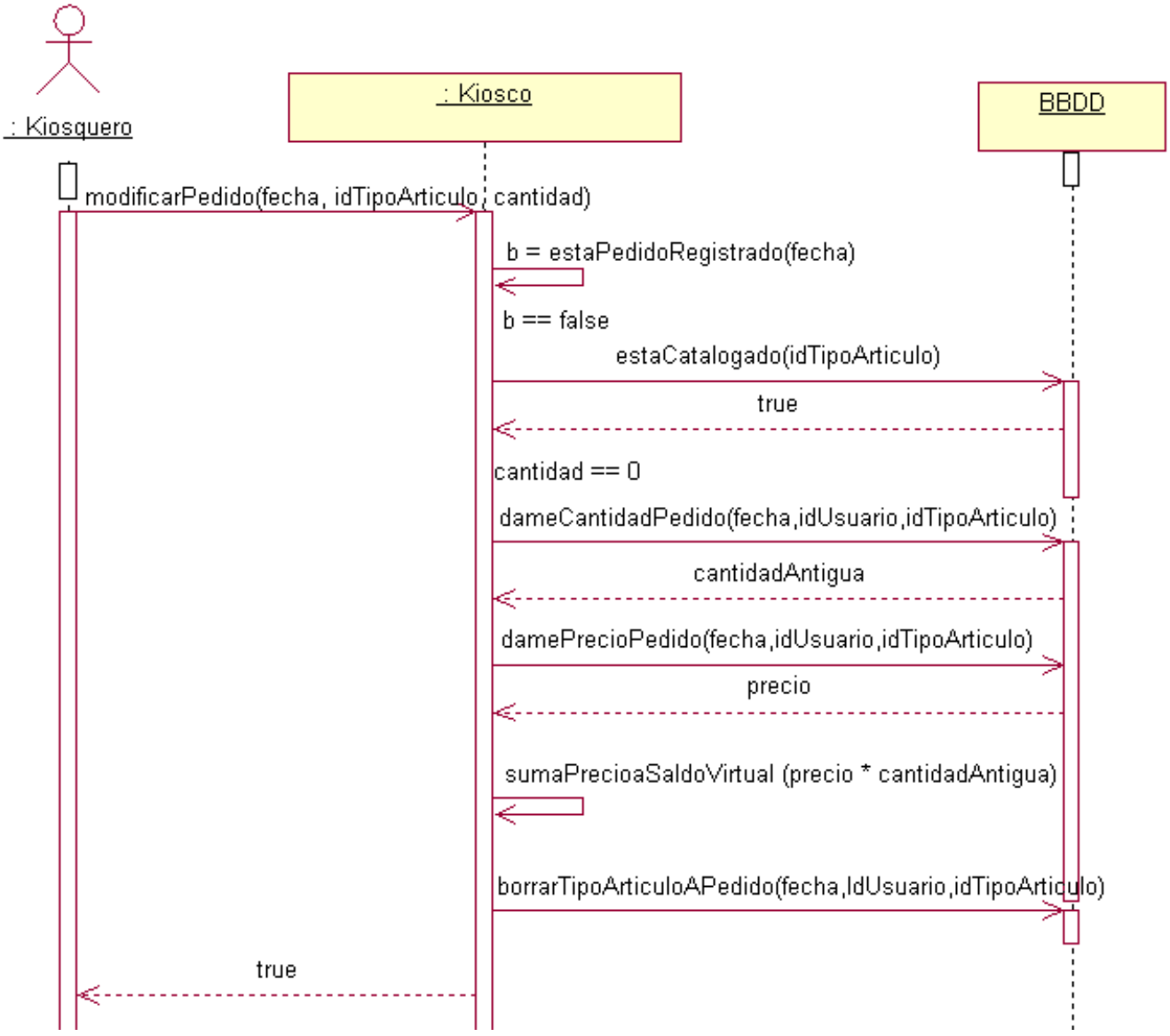


Diagrama de Secuencias de anularPedido() – DS5

Restricciones OCL:

Context Kiosco :: anularPedido (fecha Fecha)

body:

self.misPedidos -> select (p1: Pedido | p1.fecha = fecha) -> first ().misArticulos -> forAll (a1: TipoArticulo | self.modificarPedido (fecha, a1.identificador, 0))

Context Kiosco :: eliminaPedido (fecha Fecha, idU idUsuario)

post:

Pedido.filas –> select (f1: fila | f1.fecha = fecha and f1.idUsuario = idU) -> isEmpty ()

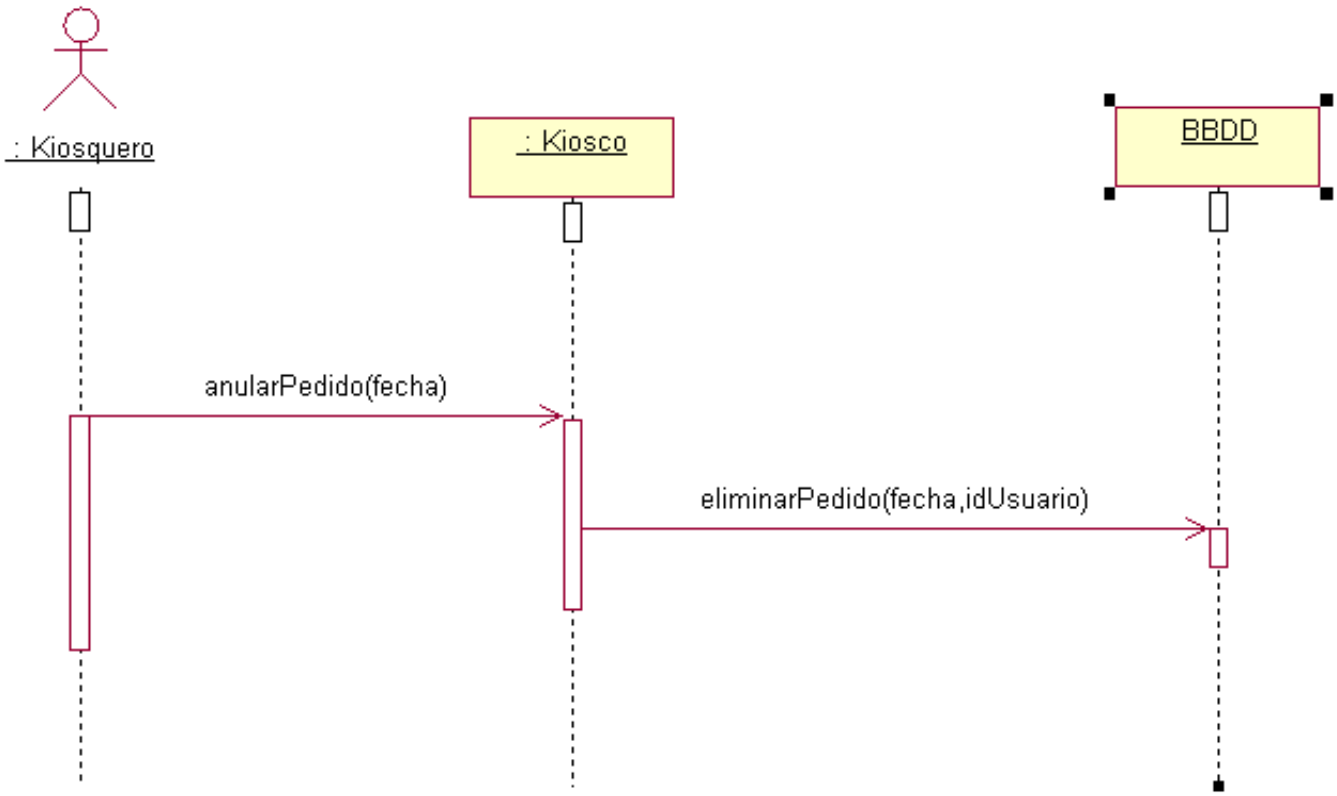


Diagrama de Secuencias de asignarArticulosPedidos() – DS6a

A las 23:00 se recogen todos los pedidos realizados para el día siguiente; para ello se usa el método *damePedidosdelDia* devuelve L que es una lista de tuplas de tres elementos: *<idTipoArticulo, SumCantidad, <IdUsuario, Cantidad, Precio>>* donde:

- *idTipoArticulo* es la referencia del artículo solicitado,
- *SumCantidad* es la suma total de las cantidades pedidas por todos los kioscos de un TipoArticulo concreto,
- una lista (a la que a partir de ahora llamaremos S), con los kioscos que pidieron ese tipo de artículo y la cantidad pedida por cada uno de ellos:
  - *idUsuario* es el identificador del kiosco,
  - *Cantidad* es el número de tipoArticulos que solicitado..
  - *Precio:* es el PCA + porcentaje de compra del tipo artículo.

Después se calcula el número de artículos existentes en el inventario del almacen, pero sólo de los tipos artículo solicitados en L. Para hacer esto se usa el método *dameInventario (L)* que devuelve la lista *: <idTipoArticulo,cantidad>*.

Se recorre la lista L para asignar a todos los kioscos que solicitaron un tipo articulo la cantidad correspondiente.

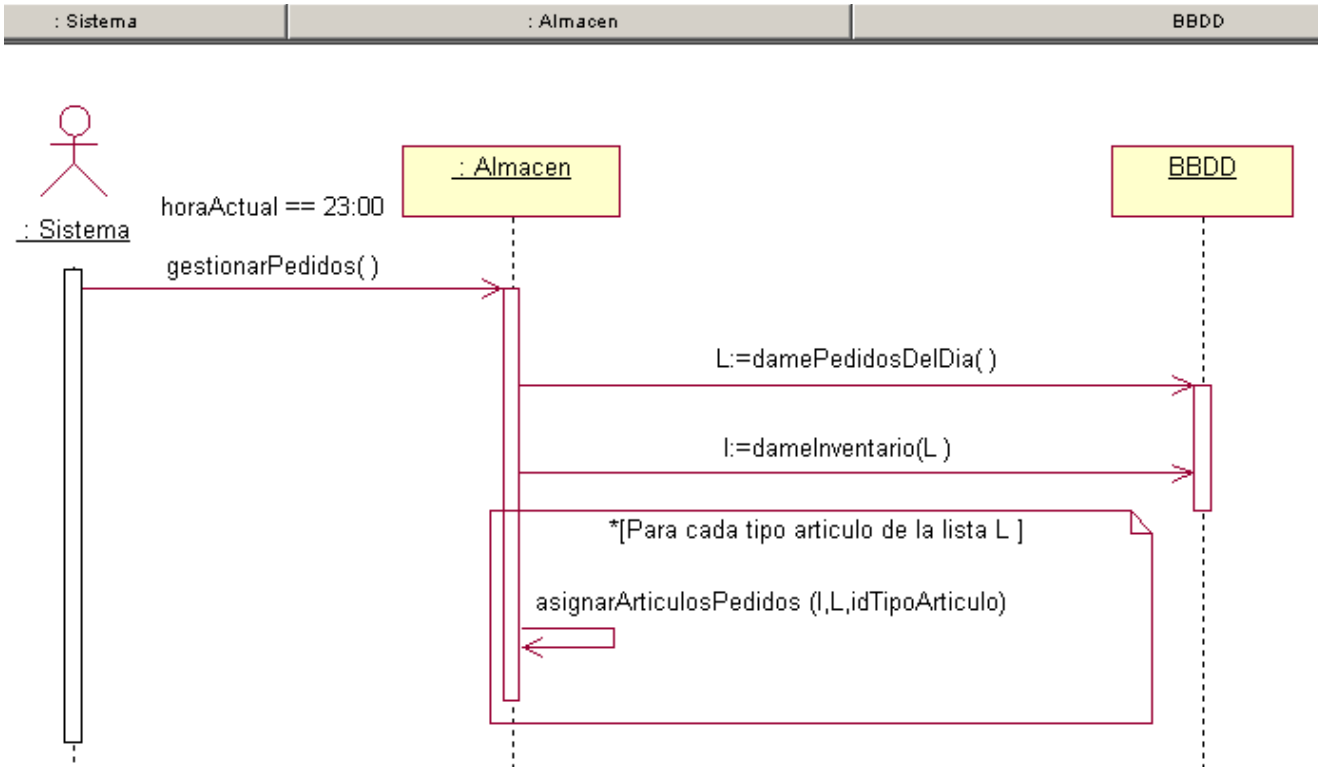


Diagrama de Secuencias de asignarPedido() – DS6b

Este método se llama desde gestionarPedidos () y asigna una cantidad de artículos por cada tipoArtículo solicitado por los kioscos.

Esta cantidad se calcula en función de las existencias en el inventario del almacén en el método calculaCantidad (I, L, K). Si hay suficientes para todos los kioscos se les asigna la cantidad solicitada en el pedido, en caso contrario, se les asigna una parte proporcional a la pedida.

Despúes se busca en la BBDD los identificadores de los artículos a asignar y se añaden en el albarán del kiosco. Por último se pone en tránsito el artículo.

Restricciones OCL:

Context Almacen :: calculaCantidad (listaPedidos Array (Nodo), inventario Array (Nodo), k Kiosco, Id idTipoArticulo)

Nodo = (ID TipoArticulo, cantidad Integer)

body:

- disponibles:= inventario -> select (n: Nodo | n.ID = Id).cantidad
- Pedidos:= listaPedidos -> select (n:Nodo| n.ID = Id).cantidad
- porcentaje:= disponibles / pedidos
- pedidosPorKiosco:= k.misPedidos -> select (p1: Pedido | p1.fecha = fechaAtcual()).lista(Id).cantidad *

```
if (x >= 1) then
    pedidosPorKiosco
else
    trunc (pedidosPorKiosco * porcentaje)
```

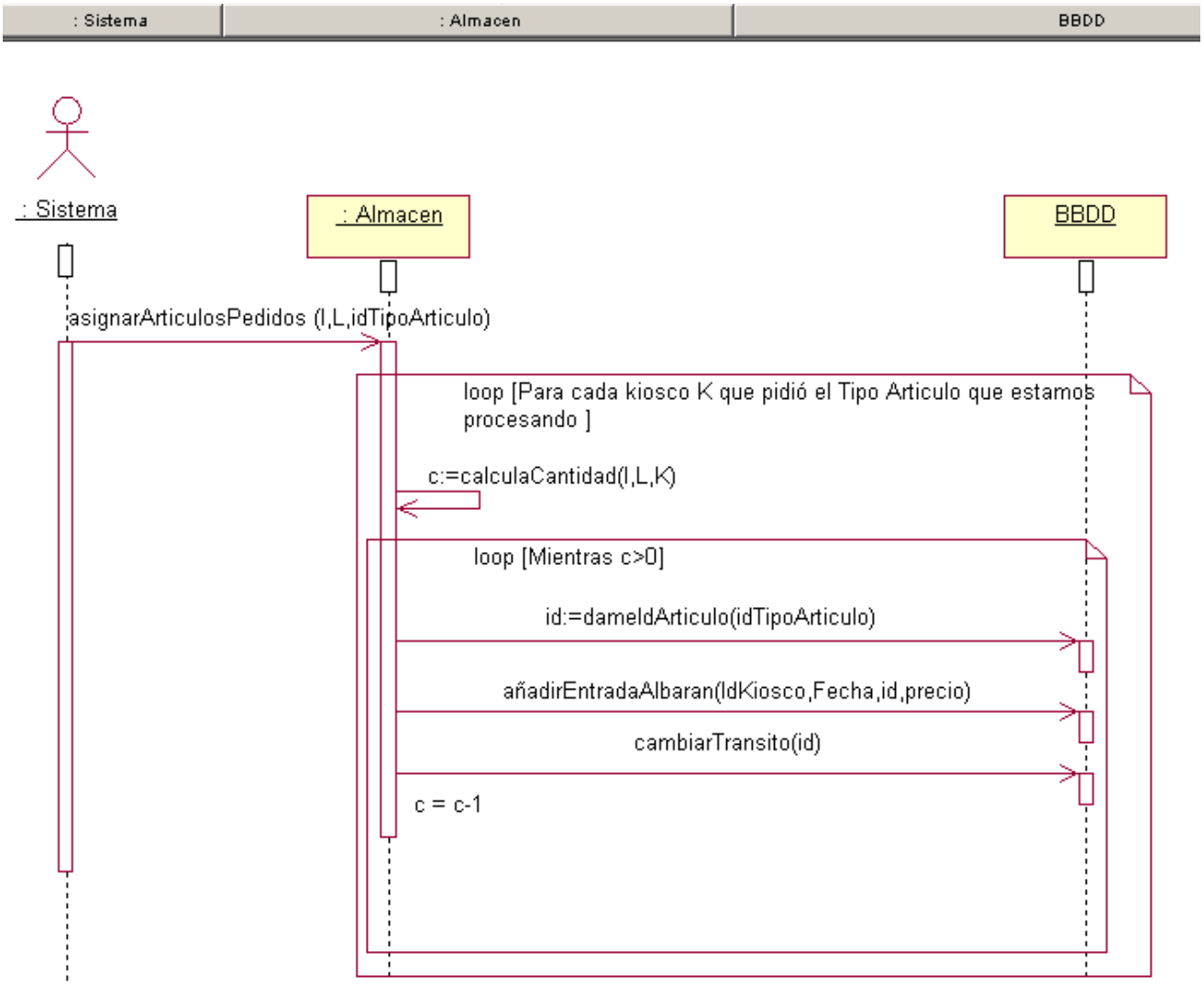

: Sistema
: Almacen
BBDD
: Sistema
: Almacen
BBDD
asignarArticulosPedidos (l,L,idTipoArticulo)
loop [Para cada kiosco K que pidió el Tipo Articulo que estamos procesando]
c:=calculaCantidad(l,L,K)
loop [Mientras c>0]
id:=dameIdArticulo(idTipoArticulo)
añadirEntradaAlbaran(IdKiosco,Fecha,id,precio)
cambiarTransito(id)
c = c-1

Diagrama de Secuencias de confirmarLlegadaArtículo() – DS7

Restricciones OCL:

Context Kiosco :: confirmarLlegadaArticulo (referencia idArticulo, precio Double, fecha Fecha)

pre:

- self.miAlbaran -> exists (alb1: Albaran | alb1.devolucion = false and (alb1.fecha = fecha and ((alb1.articulosAsignados -> exists (a1: Articulo | a1.idArticulo = referencia)))))
- not(self.miStock -> exists(a1: Articulo| a1.idArticulo = referencia)

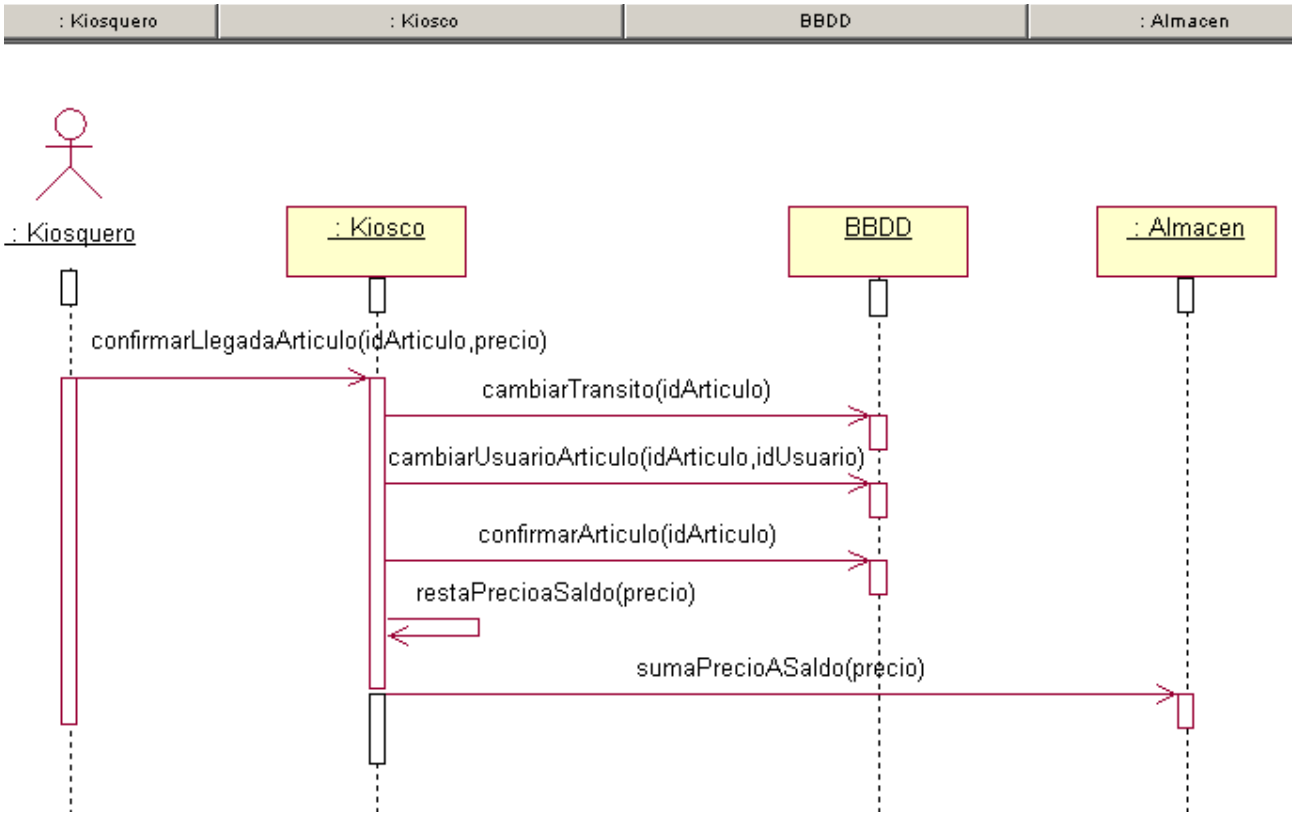


Diagrama de Secuencias de confirmarNoLlegadaArtículo() – DS8

Restricciones OCL:

Context Kiosco :: confirmarNoLlegadaArticulo (referencia idArticulo, precio Double, fecha Fecha)

pre:

- self.miAlbaran -> exists (alb1: Albaran | alb1.devolucion = false and (alb1.fecha = fecha and ((alb1.articulosAsignados -> exists (a1: Articulo| a1.idArticulo = referencia)))))
- not (self.miStock -> exists(a1: Articulo| a1.idArticulo = referencia)

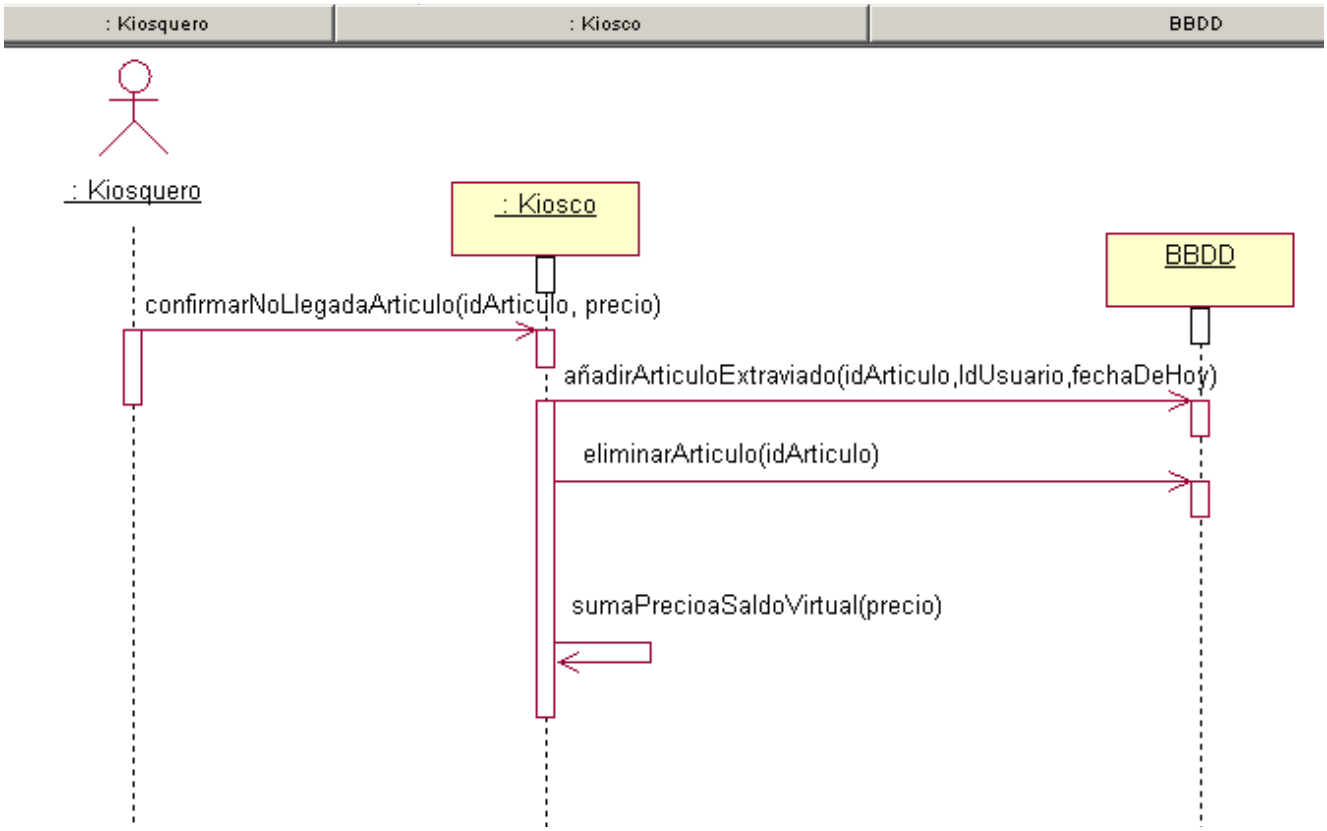


Diagrama de Secuencias de hacerReserva() – DS9

Restricciones OCL:

Context Kiosco :: hacerReserva (nombre String, ID idTipoArticulo)

pre:

- self.miAlmacen.Catalogo -> exists (t1: TipoArticulo | t1.identificador = ID)
- reservaInicial:= Kiosko.misReservas -> select (r: Reserva | r.nombre = nombre and r.miReserva.idtipoArticulo = ID) -> size ()

post:

- reservaFinal:= Kiosko.misReservas -> select (r: Reserva | r.nombre = nombre and r.miReserva.idtipoArticulo = ID) -> size ()
- reservaFinal = reservaInicial + 1

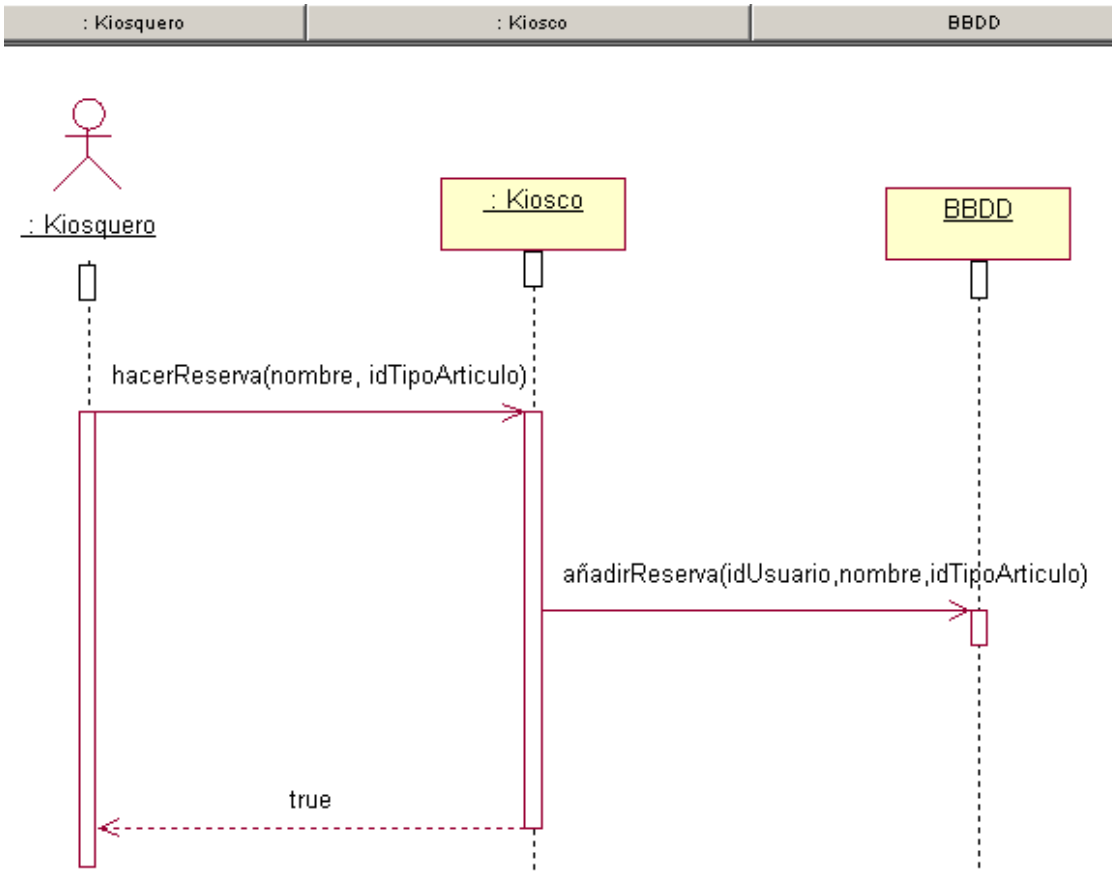


Diagrama de Secuencias de devolverArtículo() – DS10

Restricciones OCL:

Context Kiosco :: devolverArticulo (id idArticulo, fecha Date)

pre:

self.miStock -> exists (a1: Articulo | a1.idArticulo = id and not (a1.transito))

post:

self.miStock -> exists (a1: Articulo | a1.idArticulo = id and a1.transito)

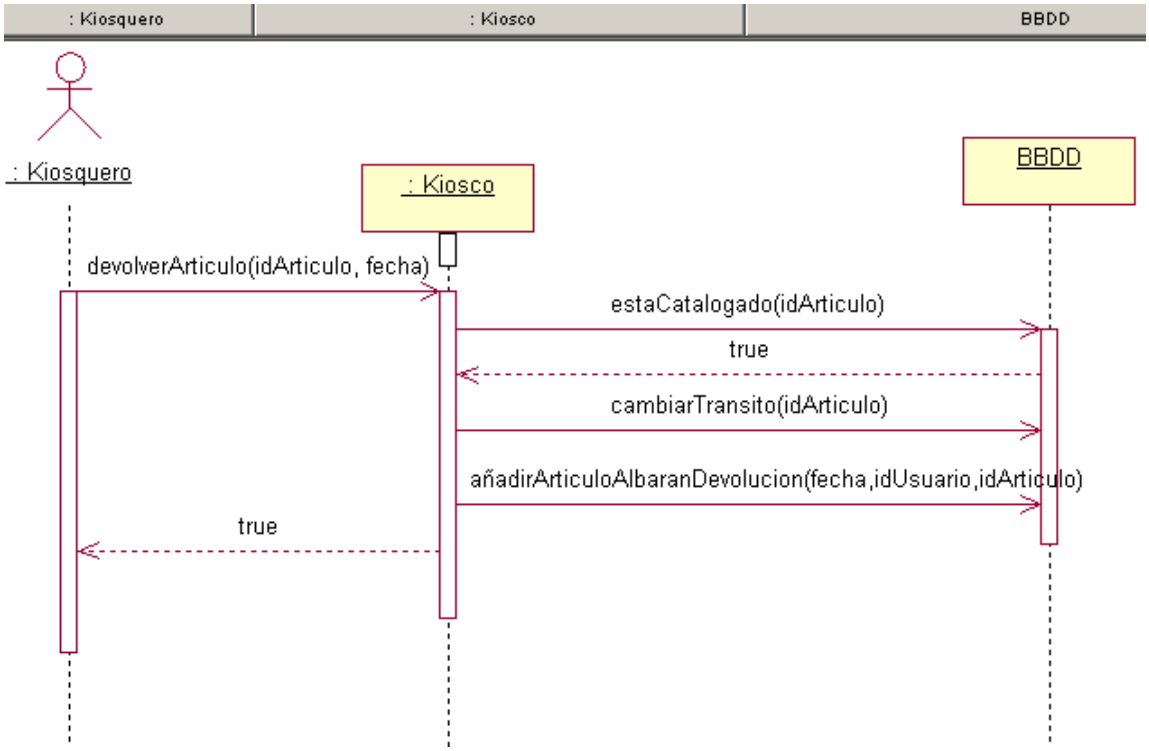


Diagrama de Secuencias de confirmarLlegadaArticuloDevuelto() – DS11

Restricciones OCL:

Context Almacen :: confirmarLlegadaArticulodevuelto (id idArticulo)

pre:

self.misKioscos -> exists (k1: Kiosko | k1.miStock -> exists (a1: Articulo | a1.transito and a1.idArticulo = id))

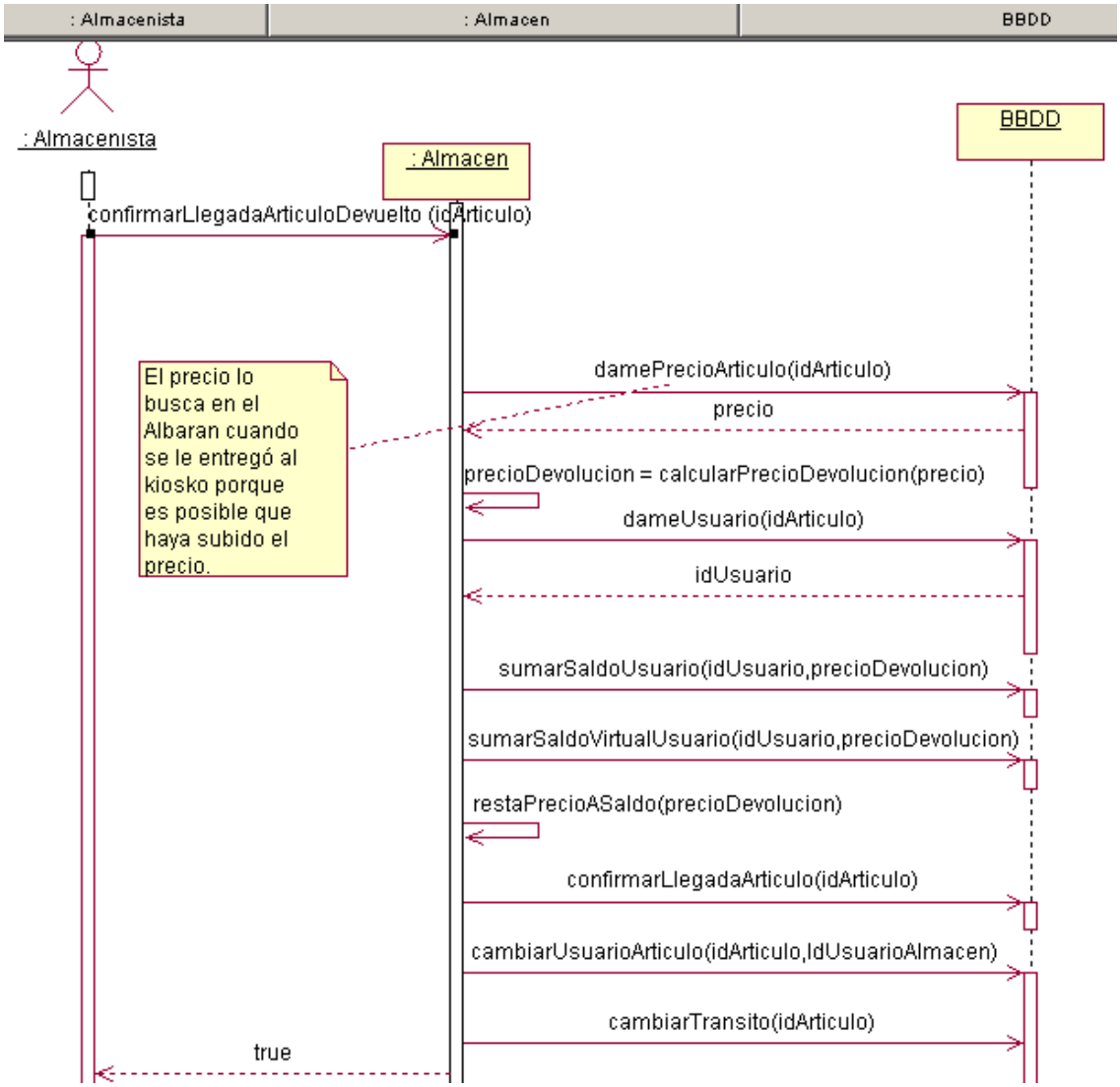


Diagrama de Secuencias de confirmarNoLlegadaArtículoDevuelto() – DS12

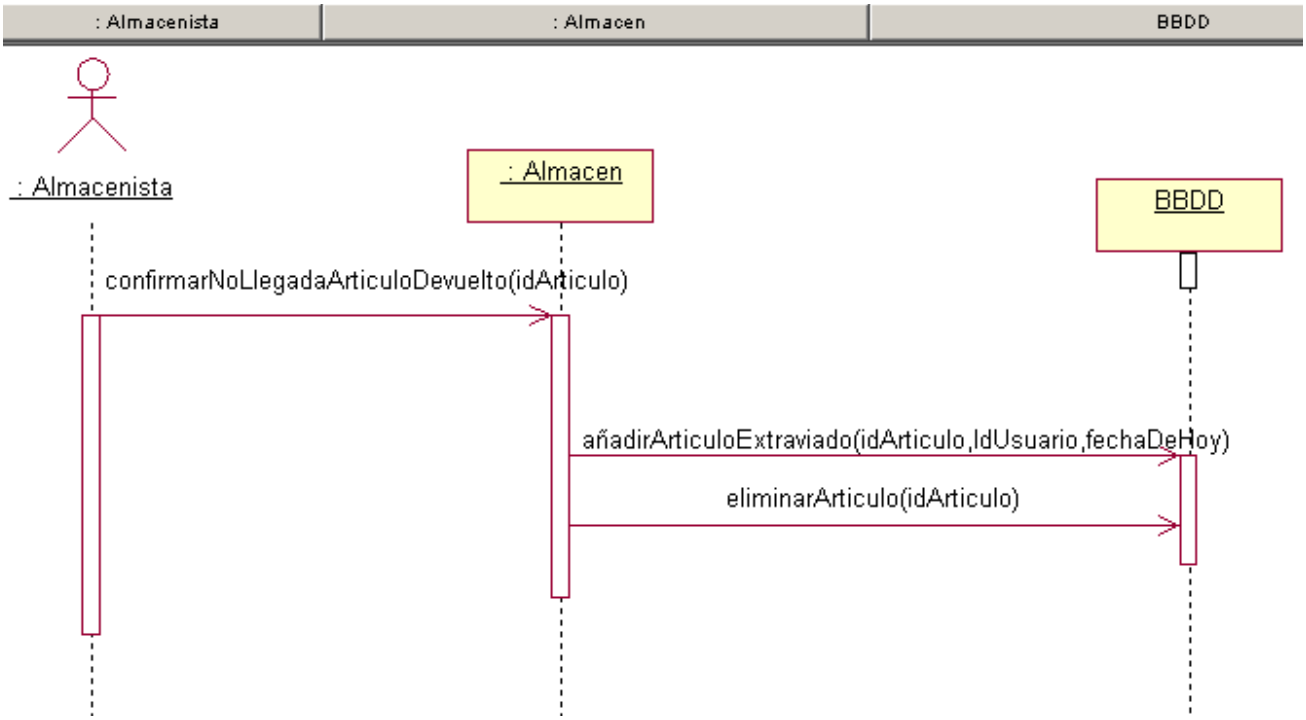


Diagrama de Secuencias de borrarReserva() – DS13

Restricciones OCL:

Context Kiosco :: borrarReserva (ID idReserva)

pre:

self.misReservas -> exists (r1: Reserva | r1.idReserva = ID)

post:

not (self.misReservas -> exists (r1: Reserva | r1.idReserva = ID))

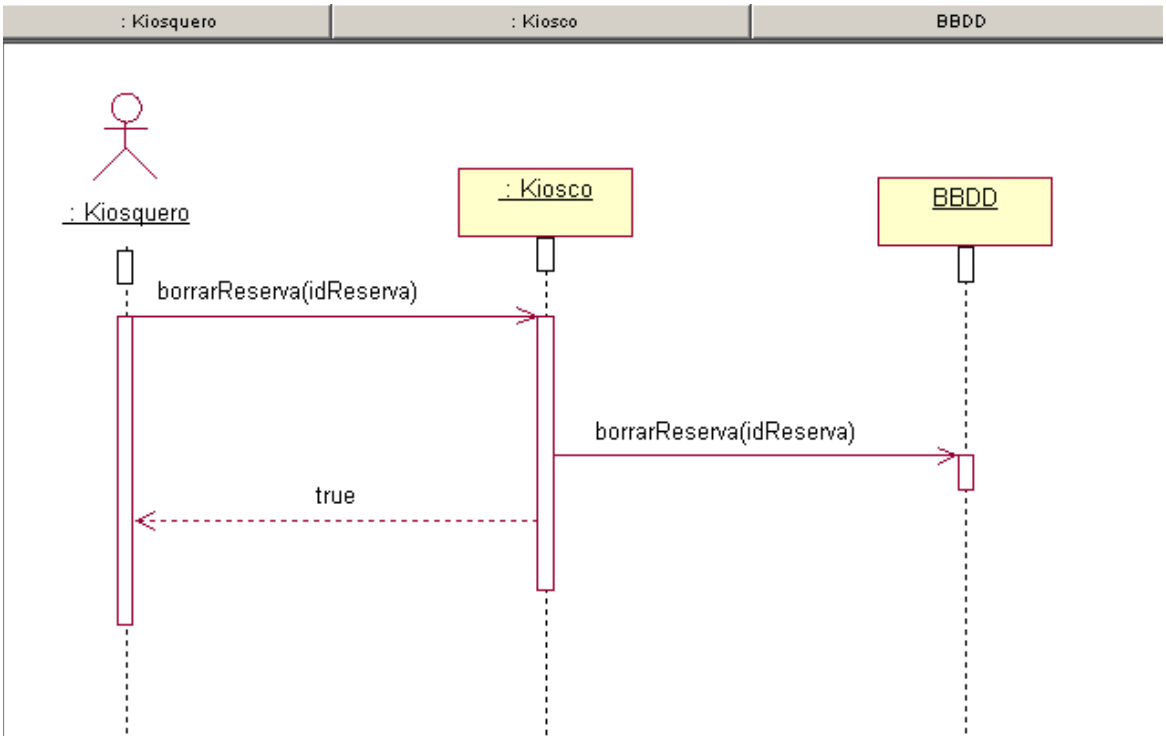


Diagrama de Secuencias de darDeAltaKiosco() – DS14

Restricciones OCL:

Context Almacen :: darDeAltaKiosco (idU idUsuario)

pre:

not (Usuario.filas -> exists (f1: fila | f1.idUsuario = idU))

post:

Usuario.filas -> exists (f1: fila | f1.idUsuario = idU and f1.saldo = 0 and f1.saldoVirtual = 0 and f1.contraseña = idU

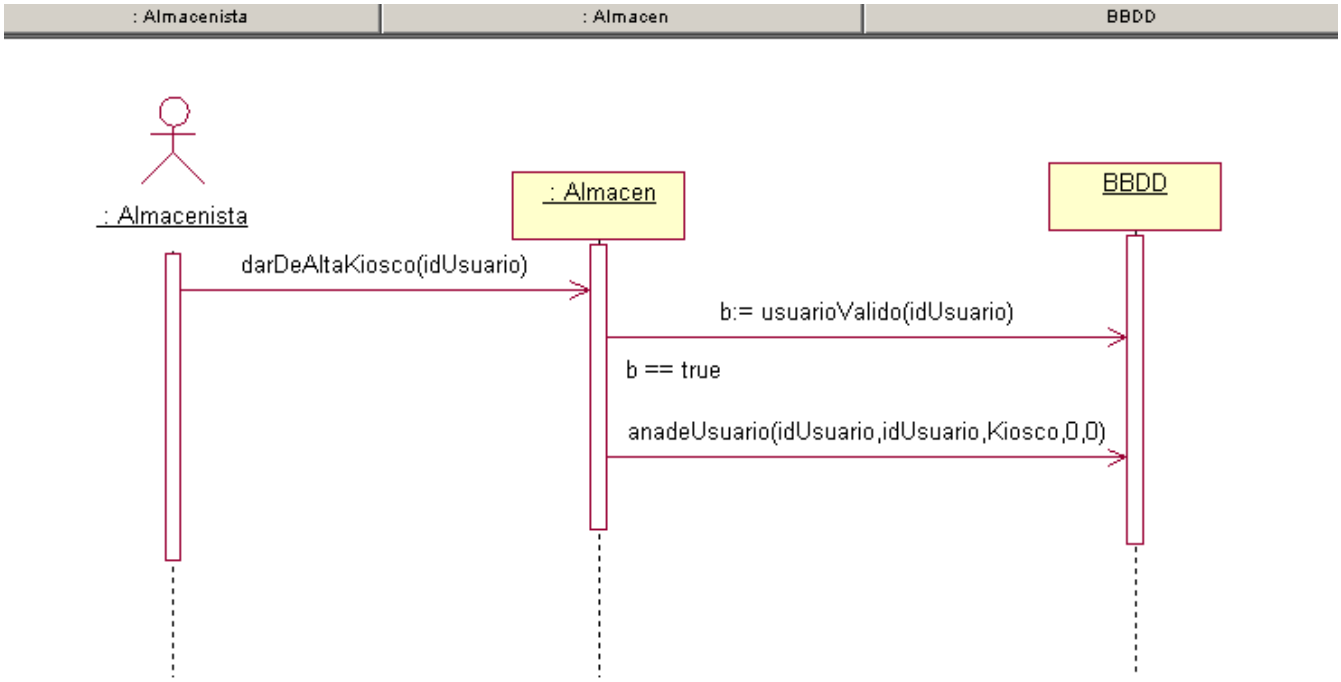


Diagrama de Ssecuencias de descatalogar() – DS17

L = lista de tuplas < fecha, idUsuario, idTipoArticulo, cantidad,precio>

Restricciones OCL:

Context Almacen :: descatalogar (ID idTipoArticulo)

pre:

```
self.Catalogo -> exists (t1: TipoArticulo | t1.idTipoArticulo = ID)
```

post:

```
not (self.Catalogo -> exists (t1: TipoArticulo | t1.idTipoArticulo = ID))
```

Context BBDD :: damePedidos (ID idTipoArticulo)

body:

```
Pedido.filas -> select (f1: fila | f1.idTipoArticulo = ID and f1.fecha >= fechaDeHoy)
```

Context BBDD :: borrarTipoArticuloAPedido (fecha Fecha, idU idUsuario, idTA idTipoArticulo)

post:

```
not (Pedido.filas -> exists (f1: fila | f1.fecha = fecha and f1.idUsuario = idU and
f1.idTipoArticulo = idTA) )
```

Context BBDD :: anadirArticuloDescatalogado (fechaDeHoy Fecha, idTA idTipoArticulo)

post:

```
ArticuloDescatalogado.filas -> exists (f1: fila | f1.fecha = fechaDeHoy and f1.idTipoArticulo =
idTA) )
```

Context BBDD :: eliminarArticulosDescatalogado (idTA idTipoArticulo, idU idUsuario)

post:

```
not  (Articulo.filas -> exists (f1: fila | (f1.idTipoArticulo = idTA and f1.idUsuario = idAlmacen
and transito = false)  or (f1.idTipoArticulo = idTA and f1.idUsuario <> idAlmacen and transito
= true and (not (AlbaranDevolucion.filas -> exists (f2: fila | (f2.idArticulo = f1.idArticulo and
f2.idUsuario = f1.idUsuario and f2.confirmado = false)))))))
```

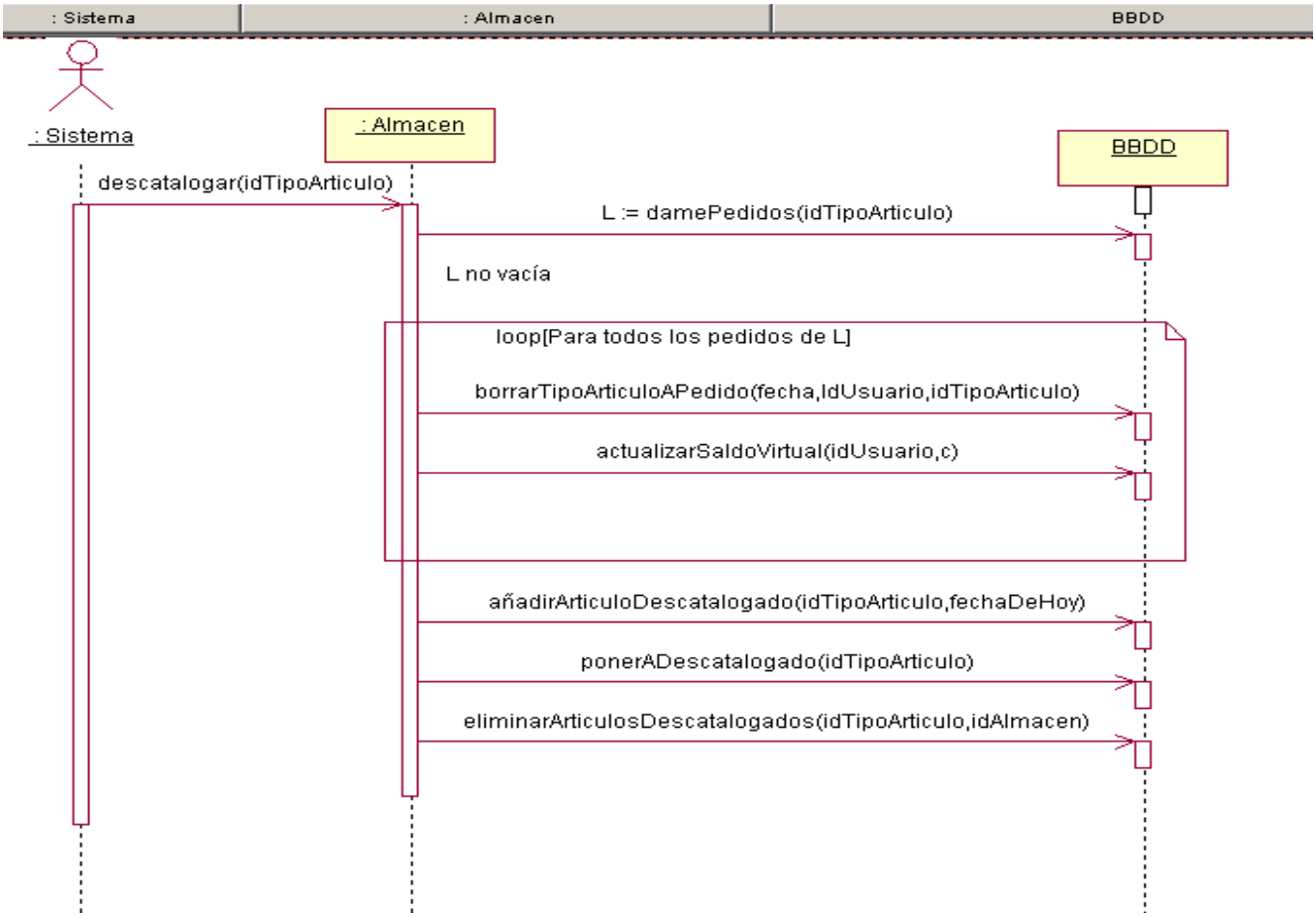

: Sistema
: Almacen
BBDD
: Sistema
: Almacen
BBDD
descatalogar(idTipoArticulo)
L := damePedidos(idTipoArticulo)
L no vacía
loop[Para todos los pedidos de L]
borrarTipoArticuloAPedido(fecha,IdUsuario,idTipoArticulo)
actualizarSaldoVirtual(idUsuario,c)
añadirArticuloDescatalogado(idTipoArticulo,fechaDeHoy)
ponerADescatalogado(idTipoArticulo)
eliminarArticulosDescatalogados(idTipoArticulo,idAlmacen)

Diagrama de secuencias de eliminarArtículosCaducados() – DS18

Restricciones OCL:

Context Almacen :: eliminarArticulosCaducados ()

pre:

horaActual () = 00:00

post:

not (self.Catalogo -> exists (t1: TipoArticulo | t1.idTipoArticulo = ID and t1.fecha <= fechaActual ()))

Context BBDD :: eliminarArticulosCaducados (idTA idTipoArticulo)

post:

Articulo.filas -> select (f: fila | f.idTipoArticulo = idTA) -> isEmpty () and (Albaran.filas -> select (f: fila | f.idArticulo.idTipoArticulo = idTA and f.confirmado = false) -> isEmpty () or AlbaranDevolucion.filas -> select (f: fila | f.idArticulo.idTipoArticulo = idTA and f.confirmado = false) -> isEmpty ())

Context Almacen :: descatalogar(idTipoArticulo)

Body:

tipoArt := self.catalogo->select(ta: TipoArticulo | ta.id = idTipoArticulo)->first()
tipoArt.misArticulos->forAll (a: Articulo | BBDD.descatalogar (a.id))

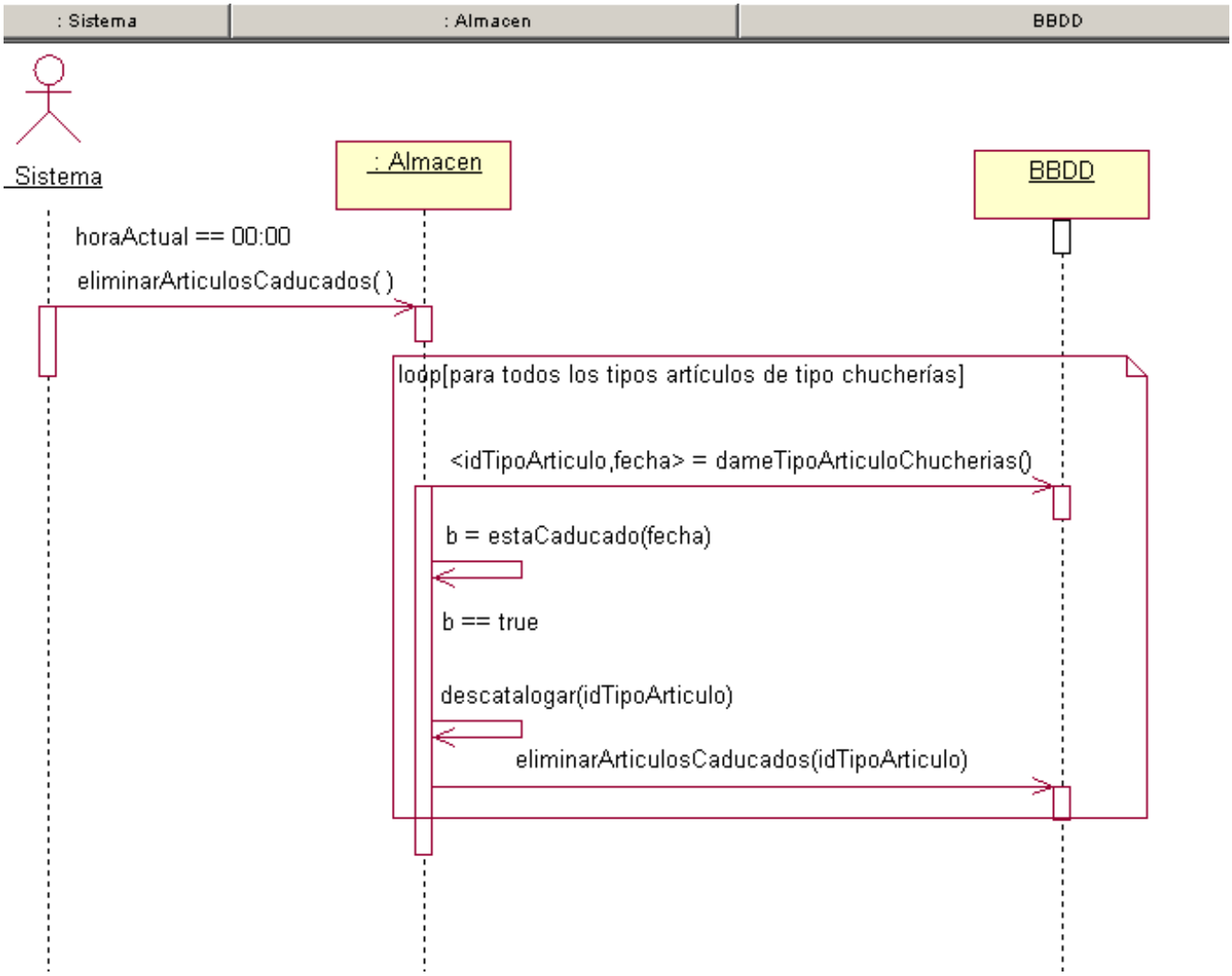


Diagrama de secuencias de añadirTipoArticulo() – DS19

Éste método se lanza desde la interfaz gráfica de almacén cuando se va a añadir un tipo de artículo al catálogo. El almacenista es el encargado de introducir correctamente los campos requeridos por la interfaz. El campo PCA corresponde al Precio de Compra del Almacén, el campo PVP, al Precio de Venta al Público. Si el tipo es publicación, la fecha corresponde a la fecha de publicación, si es de tipo chuchería, indica la fecha de caducidad correspondiente.

Se comprueba en la base de datos que no exista en la tabla TipoArticulo una entrada con nombre, tipo y fecha iguales, en cuyo caso se crea una nueva entrada.
En el caso de que ya exista una entrada, añadirTipoArticulo devuelve false y no se modifica la base de datos.

Restricciones OCL:

Context Almacen :: añadirTipoArticulo (nombre String, pca Double, pvp Double, tipo String, fecha Date)

pre:

pvp > pca > 0

post:

self.Catalogo -> exists (t1: TipoArticulo | t1.nombre = nombre and t1.fecha = fecha and t1.tipo = tipo)

Context BBDD :: existeTipoArticulo (nombre idTipoArticulo, tipo Tipo, fecha Fecha)

body:

TipoArticulo.filas -> exists (f: fila | f.idTipoArticulo = nombre and f.fecha = fecha and f.tipo = tipo)

Context BBDD :: añadirTipoArticulo (idTA idTipoArticulo, pca Double, pvp Double, tipo Tipo, fecha Fecha)

pre:

not (existeTipoArticulo (idTA, tipo, fecha))

post:

TipoArticulo.filas -> exists (f: fila | f.idTipoArticulo = idTA and f.fecha = fecha and f.tipo = tipo and f.PVP = pvp and f.PCA = pca)

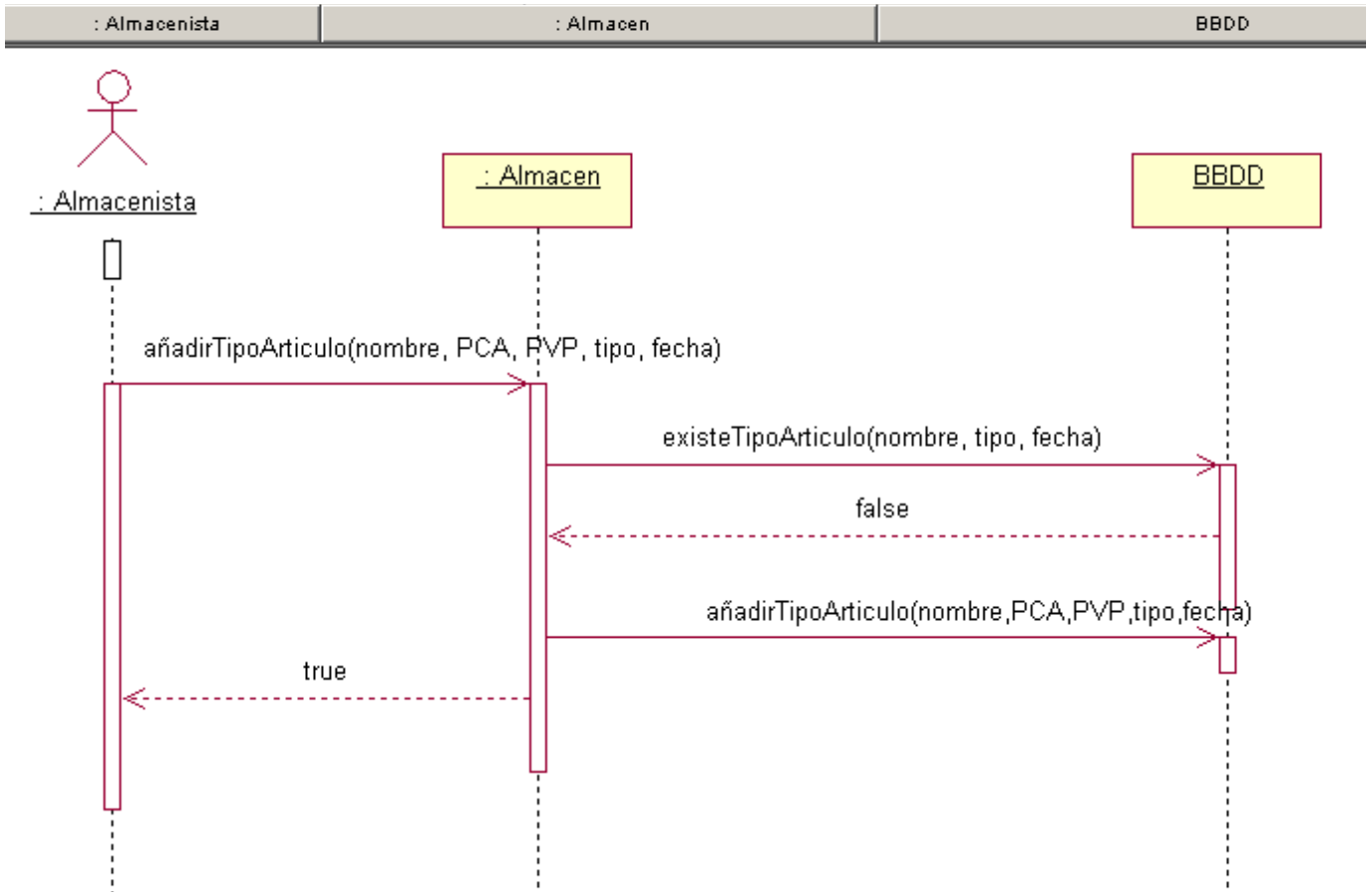

: Almacenista
: Almacen
BBDD
: Almacenista
: Almacen
BBDD
añadirTipoArticulo(nombre, PCA, PVP, tipo, fecha)
existeTipoArticulo(nombre, tipo, fecha)
false
añadirTipoArticulo(nombre,PCA,PVP,tipo,fecha)
true

Diagrama de Secuencias de añadirArtículo() – DS20

Este método se llama desde la interfaz gráfica del almacén cuando desde la vista de inventario se quiere añadir un artículo a un tipo existente, y para ello se le pasa una referencia única, el idArticulo, que se comprueba que no esté repetida en la base de datos.

Restricciones OCL:

Context Almacen :: añadirArticulo (referencia idArticulo, ID idTipoArticulo)

pre:

self.Catalogo -> exists(t1:TipoArticulo| t1.idTipoArticulo = ID)

post:

self.Catalogo.misArticulos -> exists (a1: Articulo | a1.idArticulo = referencia)

Context BBDD :: existeArticulo (referencia IdArticulo)

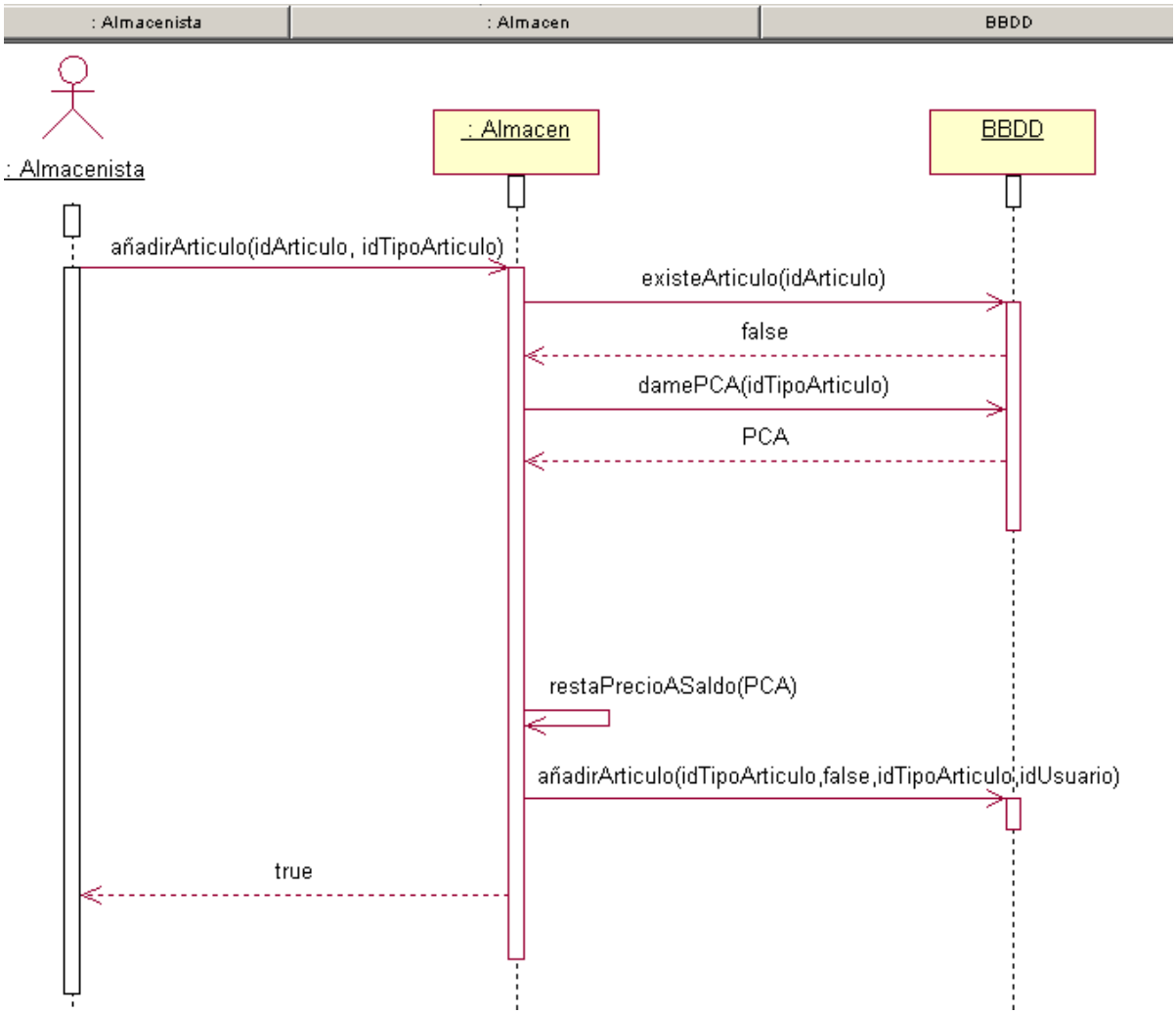
body:

not (Articulo.filas -> exists (f1: fila | f1.idArticulo = referencia) or ArticuloVendido.filas -> exists (f1: fila | f1.idArticulo = referencia) or ArticuloExtraviado.filas -> exists (f1: fila | f1.idArticulo = referencia) or ArticuloDescatalogado.filas -> exists (f1: fila | f1.idArticulo = referencia))

Context BBDD :: añadirArticulo (idA, false, idTA, idU)

post:

Articulo.filas -> exists (f1: fila | f1.idArticulo = idA and f1.idTipoArticulo = idTA and f1.transito = false and f1.idUsuario = idU)


: Almacenista
: Almacen
BBDD
: Almacenista
: Almacen
BBDD
añadirArticulo(idArticulo, idTipoArticulo)
existeArticulo(idArticulo)
false
damePCA(idTipoArticulo)
PCA
restaPrecioASaldo(PCA)
añadirArticulo(idTipoArticulo,false,idTipoArticulo,idUsuario)
true

Diagramas de Estados

Diagrama de Estados de Artículo

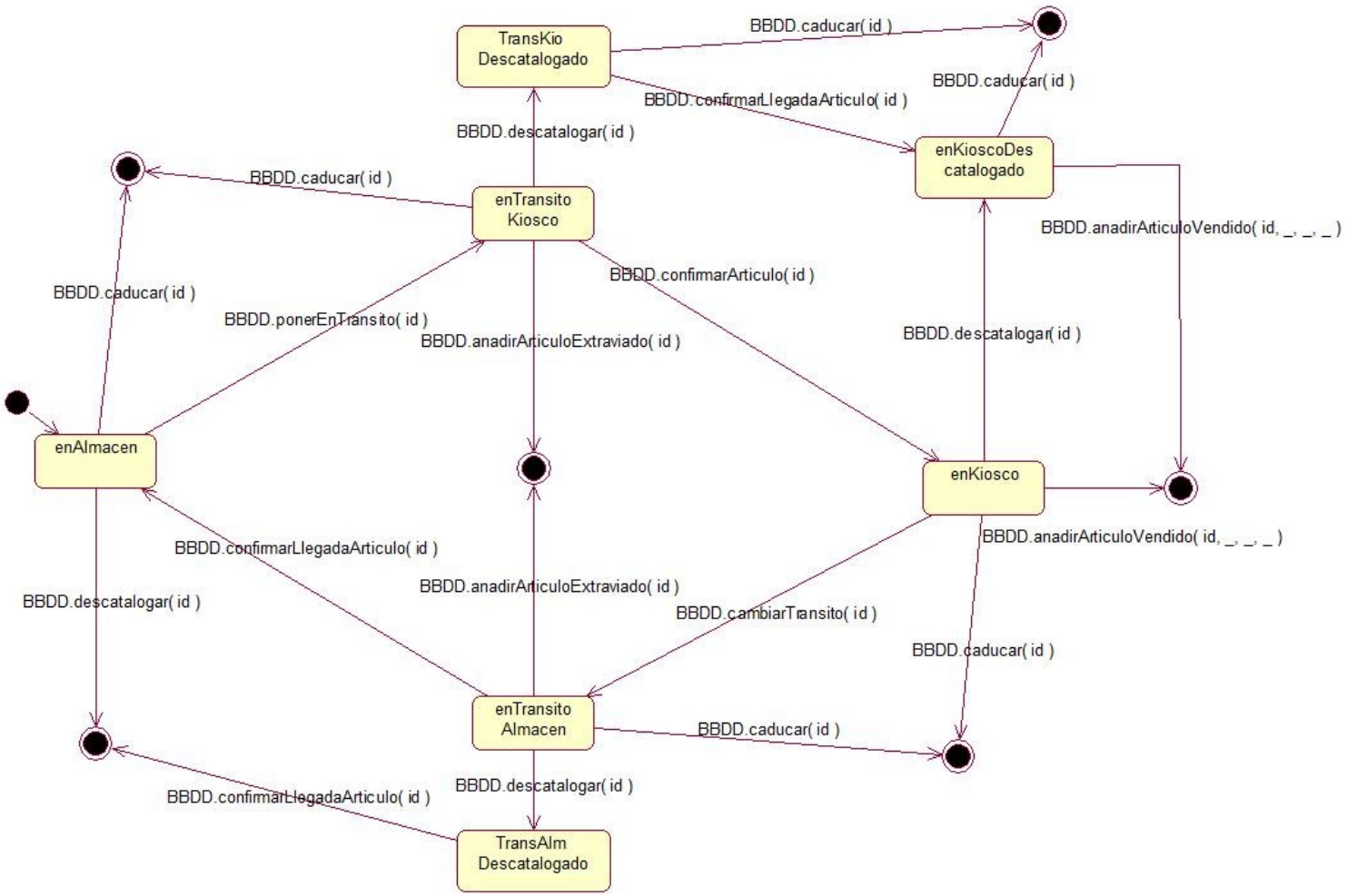
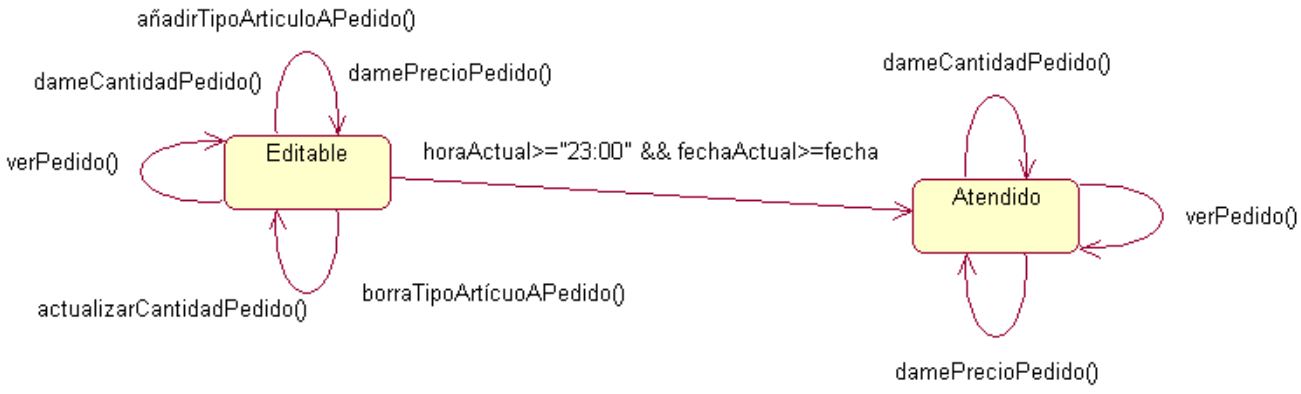


Diagrama de Estados de Pedidos



Interfaz Gráfica

Descripción en lenguaje natural

Acceso

El acceso a la aplicación por parte del usuario se realiza a través de un formulario de entrada **(IGr 01)**, en el cual se le solicitará el nombre de usuario y su correspondiente contraseña. De esta manera es posible determinar si el usuario es el almacenista o alguno de los kioscos.

IGr01. Formulario de entrada

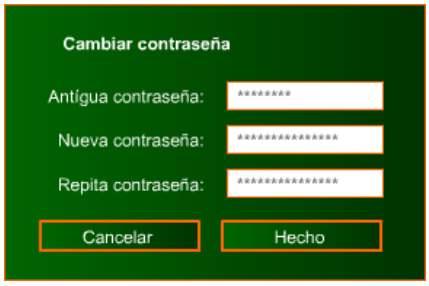
Estructura general

Una vez superado el primer paso, se accede al formulario principal de la aplicación, o marco **(IGr 02)**. Éste es común para todos los usuarios, y almacena varias pestañas con distinta información. Las pestañas visibles variarán dependiendo del rol del usuario.

El marco debe contener al menos una pestaña. De todas ellas, una (y sólo una) debe ser la pestaña seleccionada. Esto implica que sea la única pestaña visible y activa en ese momento. Se puede seleccionar cualquiera de las pestañas visibles.

Desde el marco también es posible modificar la contraseña del usuario. Para ello se habilitará una ventana emergente **(IGr 03)** que le solicitará al usuario los datos pertinentes.

IGr 02. Marco



IGr 03. Cambio de contraseña

Pestañas visibles para el kiosco

A continuación se describen las pestañas de la aplicación a las que puede acceder el kiosco.

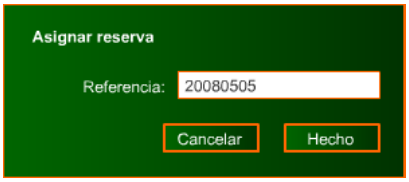
Vender

La pestaña Vender **(IGr 04)** tiene dos zonas diferenciadas. En la primera se muestra una lista de la compra, a la que se pueden añadir artículos indicando su referencia, o eliminarlos de la misma. Al pulsar el botón Vender se realizarán las gestiones oportunas con los artículos vendidos y su precio, y la lista se vaciará.

En la segunda zona se muestra la gestión de reservas. En ella se pueden ver las reservas hechas, agrupadas por nombre de reserva. Se pueden añadir y eliminar reservas **(IGr 05)**, así como mandarlas a la lista de compra (en este caso se solicitará la referencia de cada artículo marcado mediante una ventana emergente – **IGr 06**).

IGr 04. Pestaña Vender del kiosco

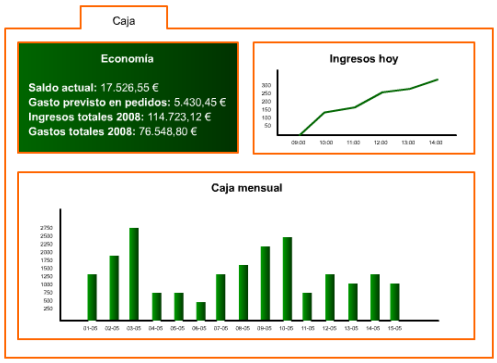
IGr 05. Nueva reserva



IGr 06. Asignar referencia

Caja

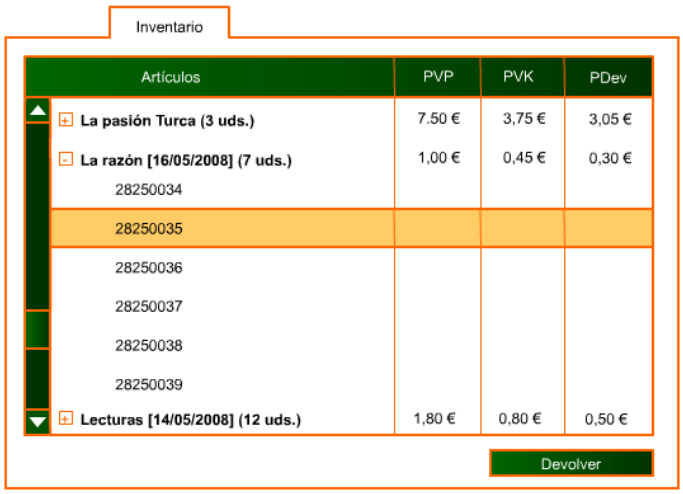
En esta pestaña (**IGr 07**) se muestran los datos económicos más relevantes del kiosco, como su saldo actual y los beneficios generados. Opcionalmente, se podrían mostrar datos más concretos, como evoluciones del estado económico del kiosco a lo largo del tiempo, o similares.



IGr 07. Datos económicos del kiosco

Inventario

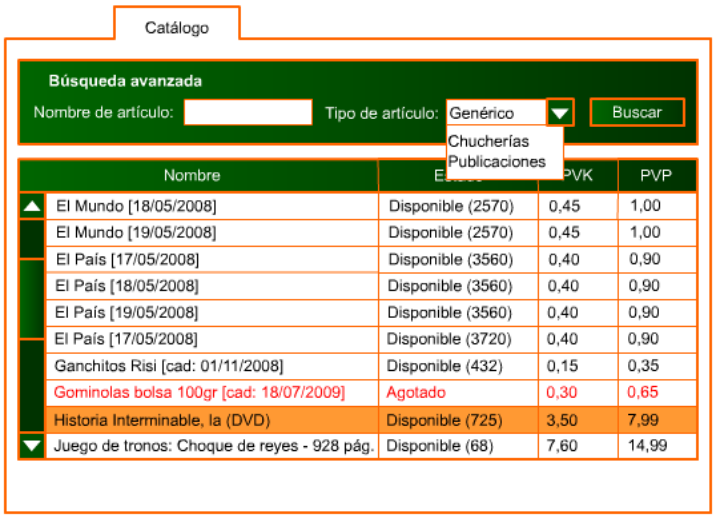
El inventario **(IGr 08)** muestra los artículos que posee el kiosco, así como sus datos relevantes. Desde esta sección se pueden devolver los productos que el kiosquero considere oportuno.



IGr 08. Inventario del kiosco

Catálogo

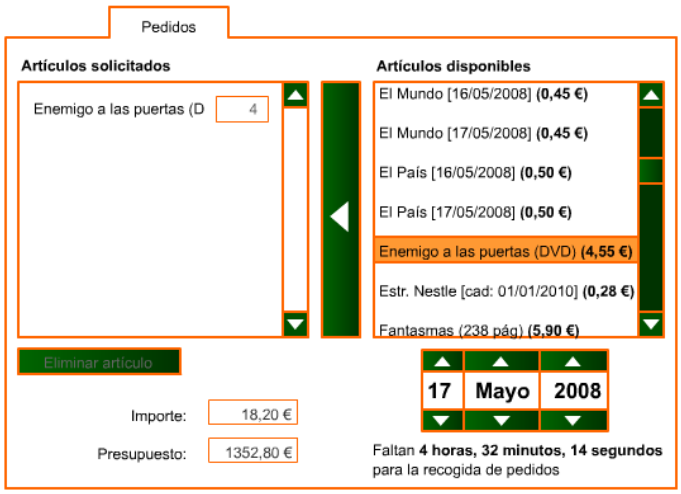
En el catálogo **(IGr 09)** se muestran todos los tipos de artículo disponibles en el almacén, así como sus datos más importantes.



IGr 09. Catálogo del almacén, visto desde un kiosco

Pedidos

En esta pestaña **(IGr 10)** se pueden gestionar los pedidos del kiosco para los días venideros, así como consultar los pedidos de días pasados.



IGr 10. Pestaña de pedidos del kiosco

Albarán

Los correcta llegada de los pedidos debe ser confirmada en esta pestaña **(IGr 11 y 12)**. Se podrán consultar los albaranes hasta la fecha actual, y sólo se podrán confirmar los albaranes no confirmados.

IGr 11. Albarán sin confirmar

IGr 12. Albarán confirmado

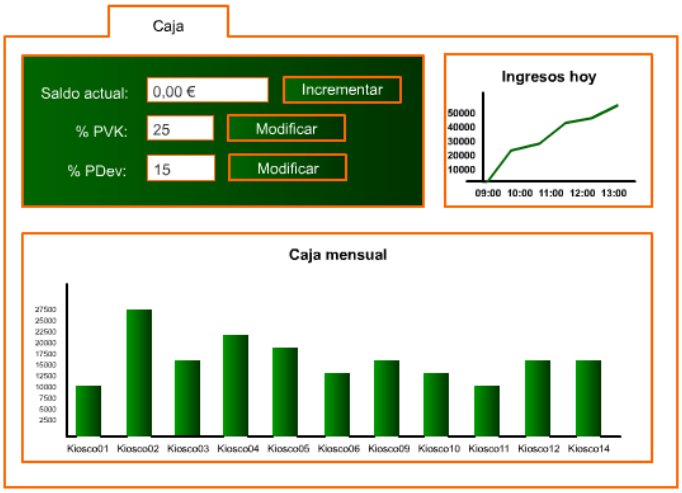
Pestañas visibles para el almacén

En este apartado se muestran las pestañas a las que el almacenista tiene acceso.

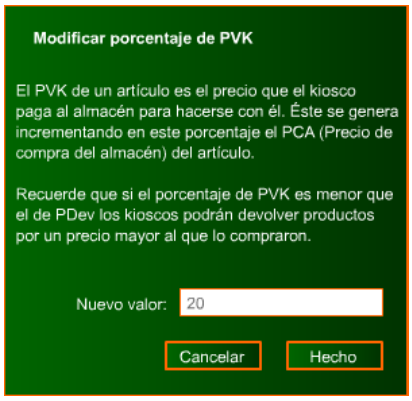
Caja

En esta pestaña **(IGr 13)** se muestran los datos económicos del almacén, así como la posibilidad de incrementar el saldo del mismo (para esto se habilita una ventana emergente).

Desde esta sección se pueden gestionar los porcentajes generadores de PVK y PDev a través de unas ventanas emergentes habilitadas para ello **(IGr 14)**.



IGr 13. Caja del almacén



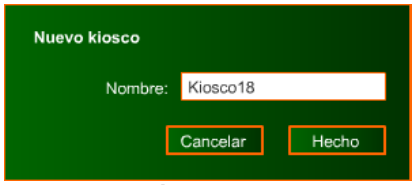
IGr 14. Modificar PVK

Kioscos

En la pestaña Kioscos **(IGr 15)** el almacén puede consultar los pedidos y el inventario de cualquier kiosco dado de alta en el sistema. También puede incrementar su saldo, o darlo de baja.

Por último, se puede añadir un nuevo kiosco al sistema, indicando su nombre en una ventana emergente habilitada para el caso **(IGr 16)**.

IGr 15. Un kiosco, visto desde el almacén

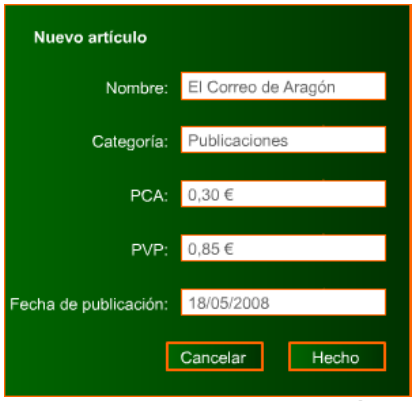


IGr 16. Creación de un nuevo kiosco

Catálogo

El catálogo del almacén **(IGr 17)** es muy similar al catálgo visto desde cualqueir kiosco, con la particularidad de que el almacén puede modificar el catálogo **(IGr 17a)**.

IGr 17. El catálogo, visto desde el almacén



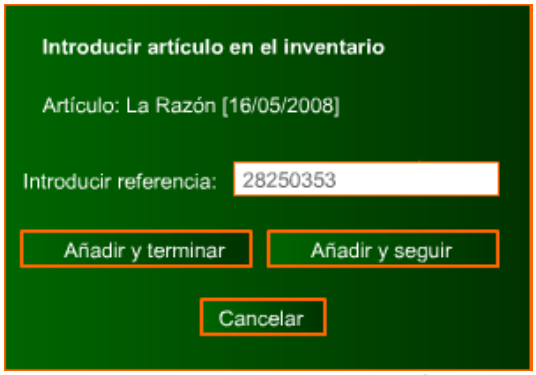
IGr 18. Añadir un tipo de artículo al catálogo

Inventario

En el inventario del almacén **(IGr 18)** se pueden ver todos los artículos que aún no han sido repartidos a los kioscos, o que han sido devueltos por los mismos. Es posible añadir nuevos artículos al mismo **(IGr 20)**.
Los artículos perdidos se podrán visualizar en la correspondiente vista **(IGr 19)**.

IGr 18. Inventario del almacén

IGr 19. Ver los artículos perdidos



IGr 20. Crear un nuevo artículo

Devoluciones

Desde esta pestaña (**IGr 21**) el almacén puede confirmar la llegada de los artículos devueltos por los kioscos para volver a incorporarlos a su inventario.

IGr 21. Sección de devoluciones del almacén

Especificación formal

Diagrama de Clases de las Ventanas Emergentes - DCIG01

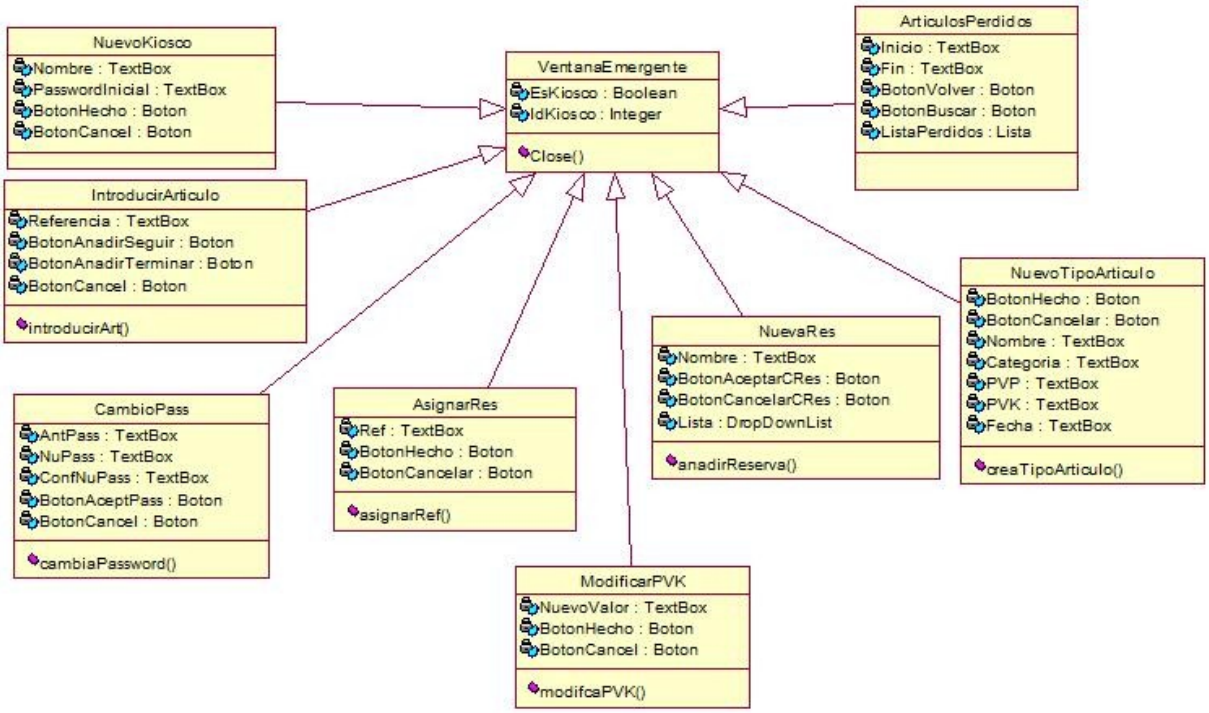


Diagrama de Clases de las Pestañas - DCIG02

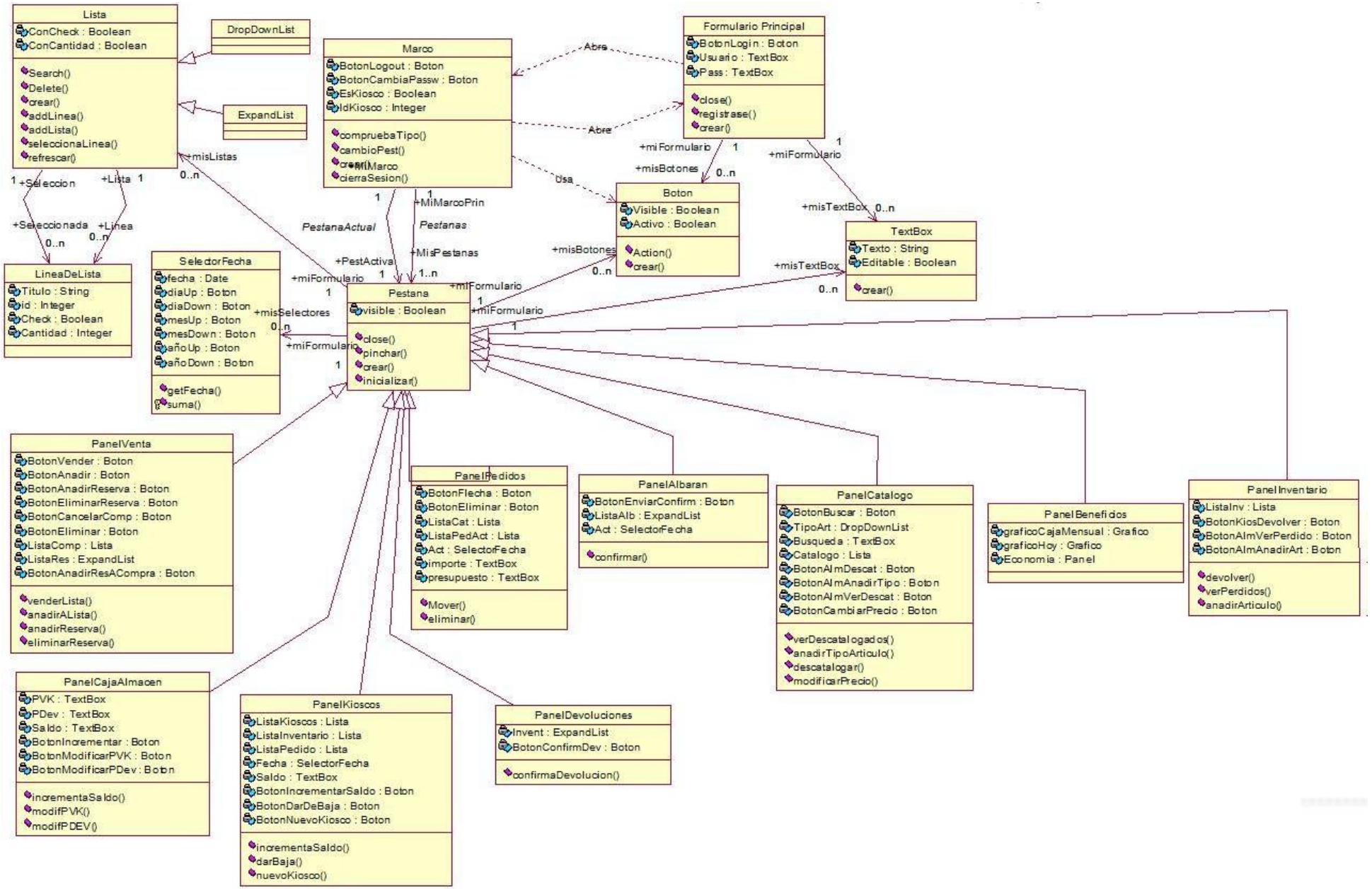


Diagrama de Secuencias de la Interfaz Gráfica de cambiar contraseña en kiosco

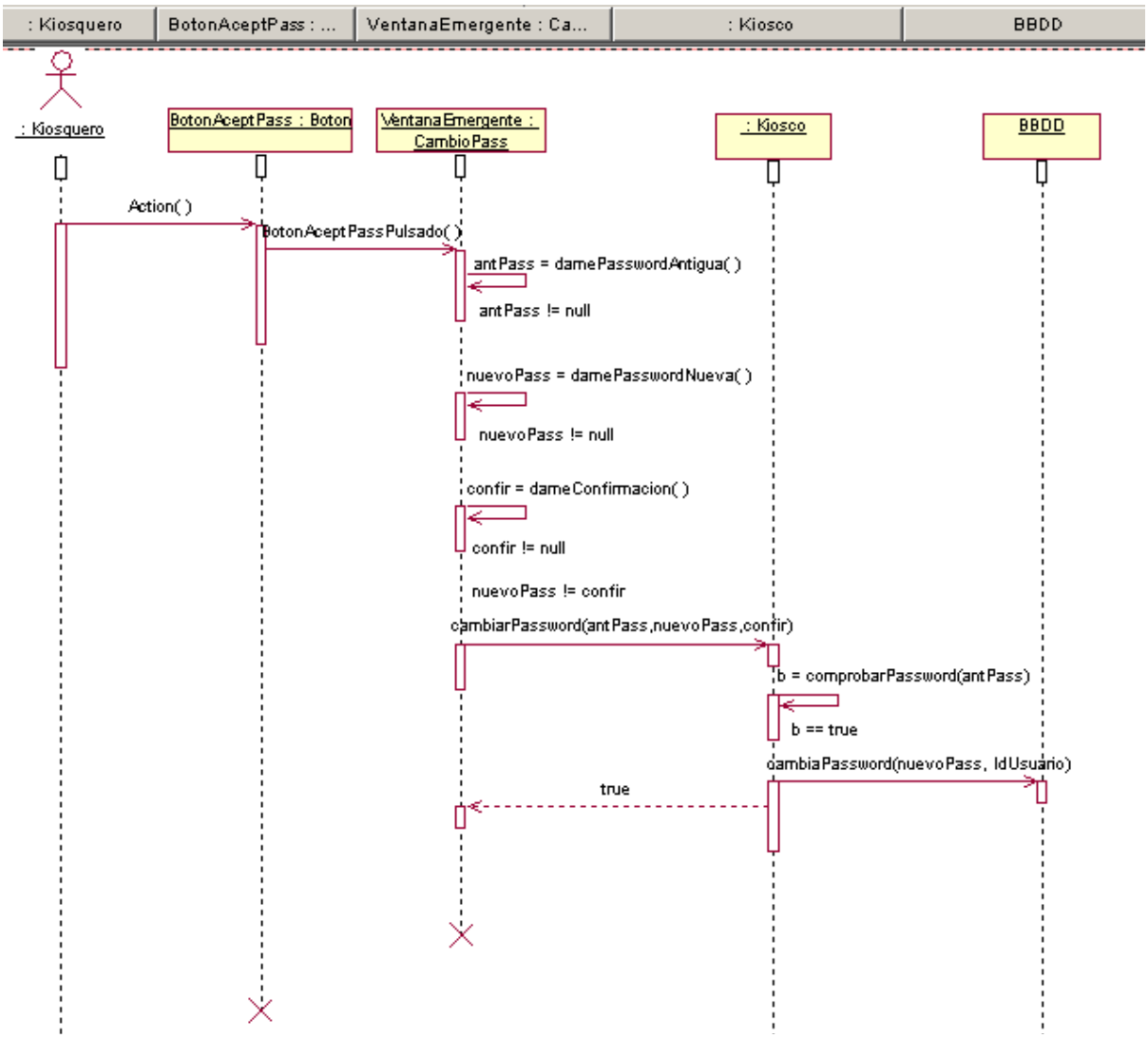


Diagrama de Secuencias de la Interfaz Gráfica de Acceso

Disponibilidad

Context FormularioPrincipal : Usuario.editable = true
Context FormularioPrincipal : Pass.editable = true

Context FormularioPrincipal : BotonLogin.visible = true
Context FormularioPrincipal : BotonLogin.activo =
(self.Pass.texto <> “”)

Funcionalidades

A continuación se muestra el diagrama de secuencias que determina el comportamiento de la aplicación al pulsar el botón *Login* con datos de kiosco. El comportamiento para el almacenista sería análogo:

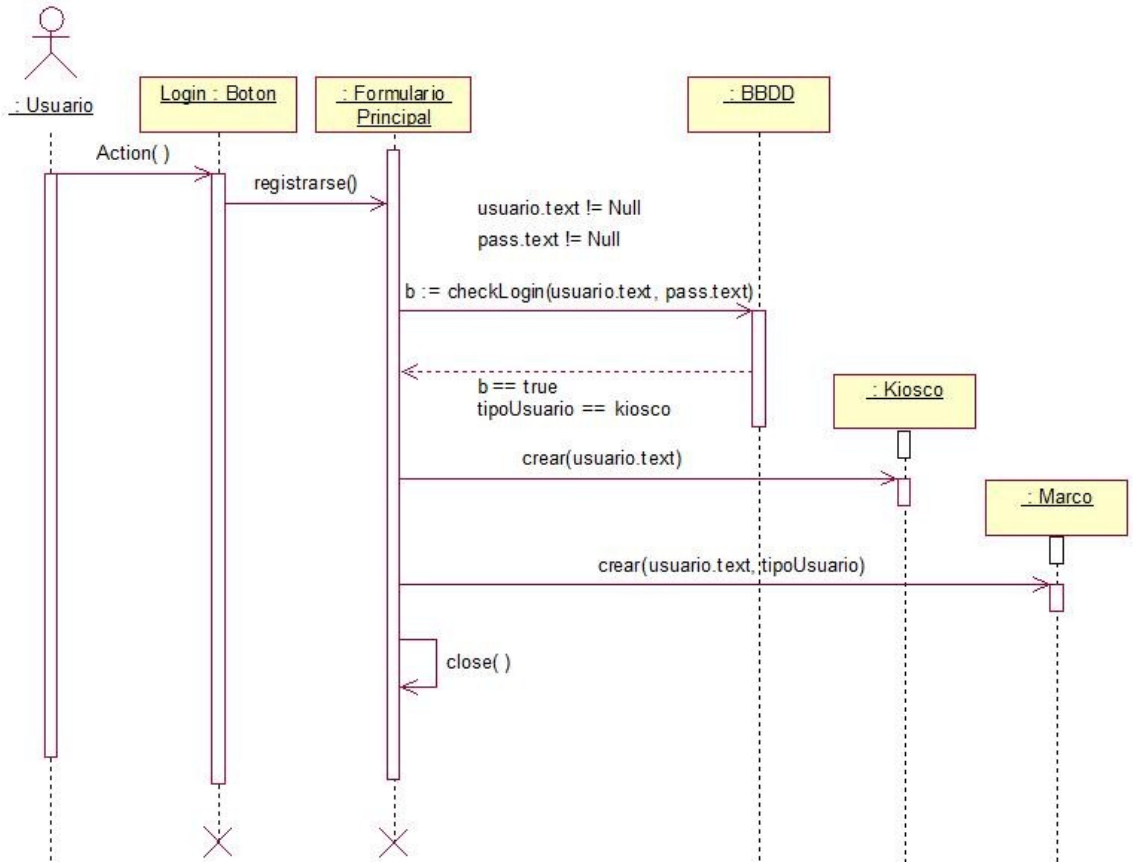


Diagrama de Secuencias de la Interfaz Gráfica de Creación de Marcos

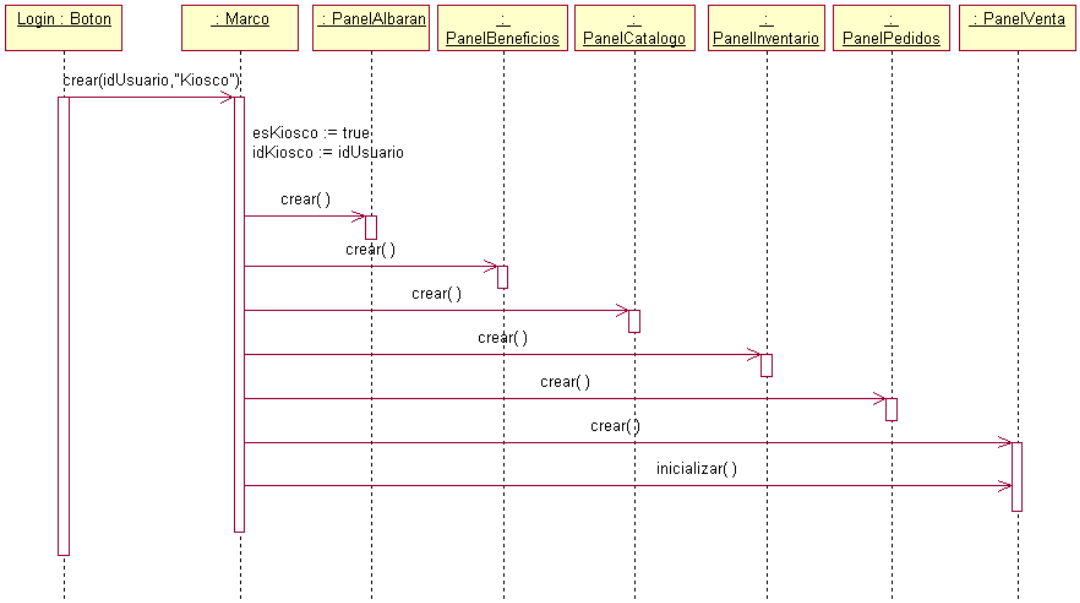
Disponibilidad

Context Marco : BotonLogout.visible = true
Context Marco : BotonLogout.activo = true

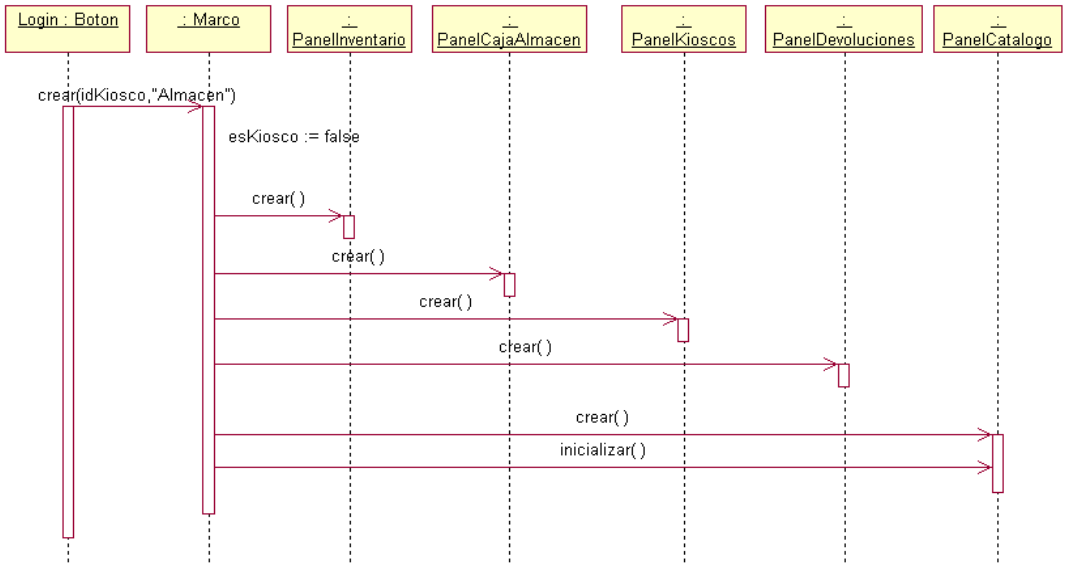
Context Marco : BotonCambiaPassw.visible = true
Context Marco : BotonCambiaPassw.activo = true

Funcionalidades

-- Creación del marco (para un kiosco)



-- Creación del marco (para el almacén)

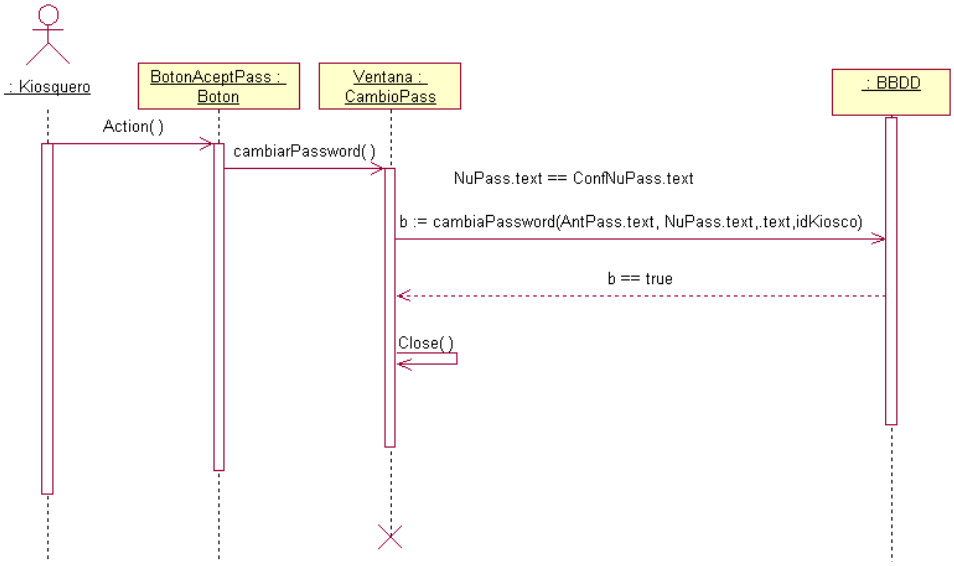


-- Cambio de pestaña

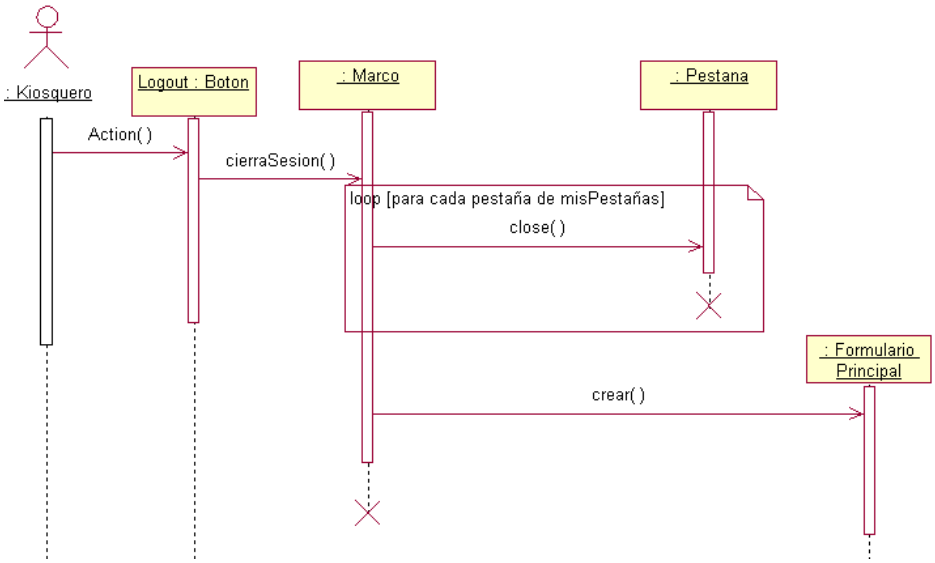
Context Marco :: cambioPest (p Pestana)

Post: self.pestanaActual = p

-- Cambiar contraseña

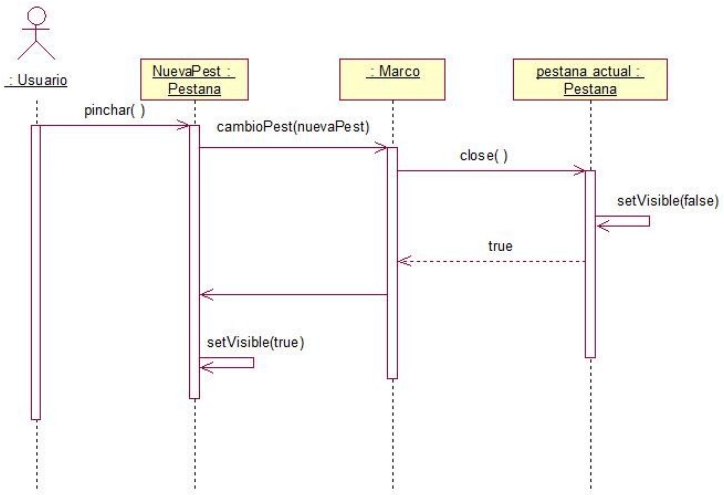


-- Comportamiento del marco al pulsarse el botón *Logout*.



Comportamiento de las pestañas

-- El método pinchar gestiona el paso de una pestaña a otra.



Ejemplo de comportamiento de pestaña: Vender

Disponibilidad

-- Pestaña vender

```
Context PanelVenta : BotonAnadir.visible = true
Context PanelVenta : BotonAnadir.activo = true

Context PanelVenta : BotonEliminar.visible = true
Context PanelVenta : BotonEliminar.activo =
                                (listaComp.seleccionada <> nil)

Context PanelVenta : BotonCancelarComp.visible = true
Context PanelVenta : BotonCancelarComp.activo =
                            (listaComp.linea->size() > 0)

Context PanelVenta : BotonVender.visible = true
Context PanelVenta : BotonVender.activo =
                           (listaComp.linea->size() > 0)

Context PanelVenta : BotonAnadirReservaACompra.visible = true
Context PanelVenta : BotonAnadirReservaACompra.activo =
                            (listaRes.seleccionada <> nil)

Context PanelVenta : BotonAnadirReserva.visible = true
Context PanelVenta : BotonAnadirReserva.activo = true

Context PanelVenta : BotonEliminarReserva.visible = true
Context PanelVenta : BotonEliminarReserva.activo =
                            (listaRes.seleccionada <> nil)
```

-- Ventana emergente de nueva reserva

Context NuevaRes : Nombre.editable = true

Context NuevaRes : BotonCancelarCRes.visible = true
Context NuevaRes : BotonCancelarCRes.activo = true

Context NuevaRes : BotonAceptarCRes.visible = true
Context NuevaRes : BotonAceptarCRes.activo =
(lista.linea->size() > 0)

-- Ventana emergente de asignar reserva

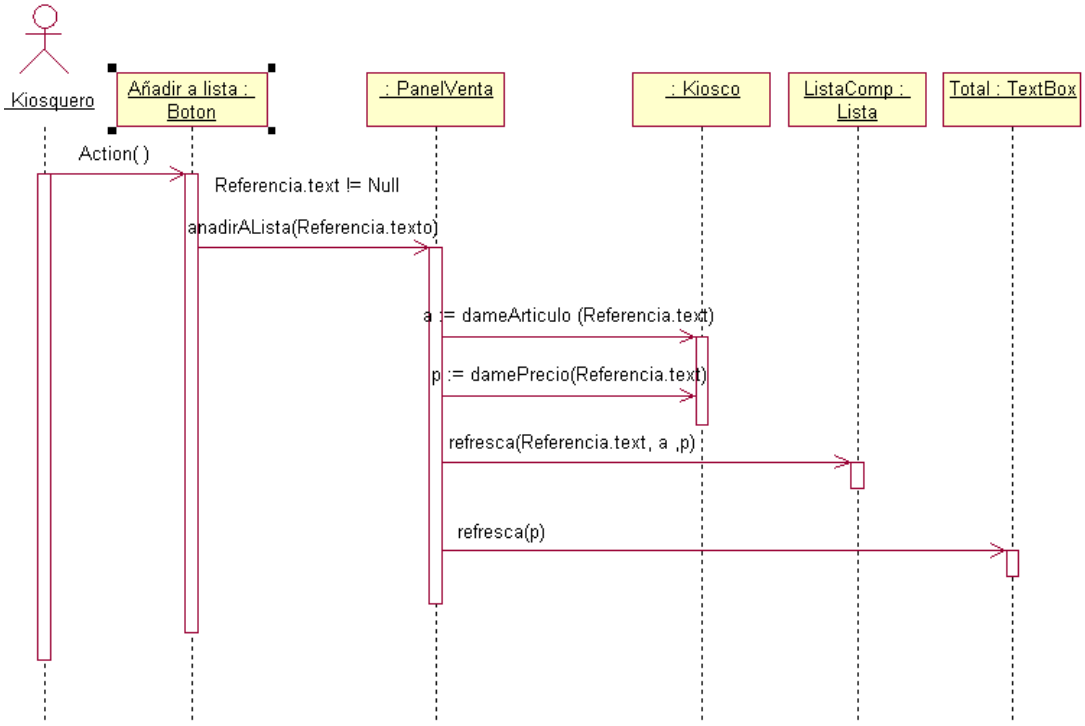
Context AsignarRes : Ref.editable = true

Context AsignarRes : BotonCancelar.visible = true
Context AsignarRes : BotonCancelar.activo = true

Context AsignarRes : BotonAceptar.visible = true
Context AsignarRes : BotonAceptar.activo = Ref.texto <> “”

Funcionalidades de PanelVender

-- Añadir elementos a la lista de venta



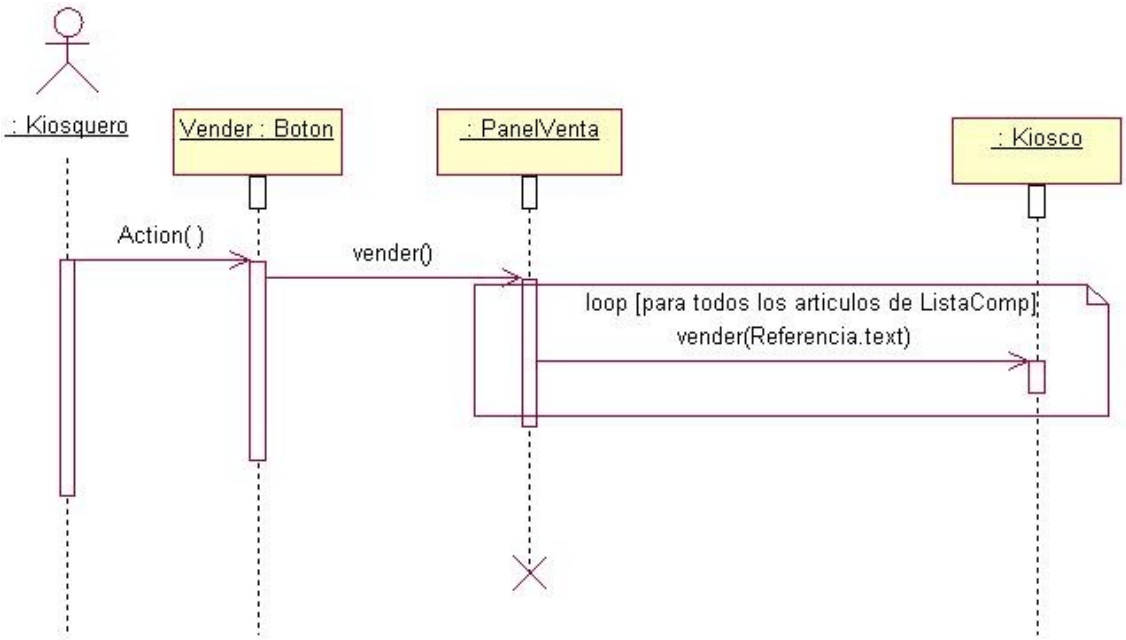
-- *Eliminar seleccionados* y *Eliminar todos* de la lista de compra

Context panelVenta :: BotonEliminar.Action()
Body: self.listaComp.delete(self.listaComp.seleccionada)

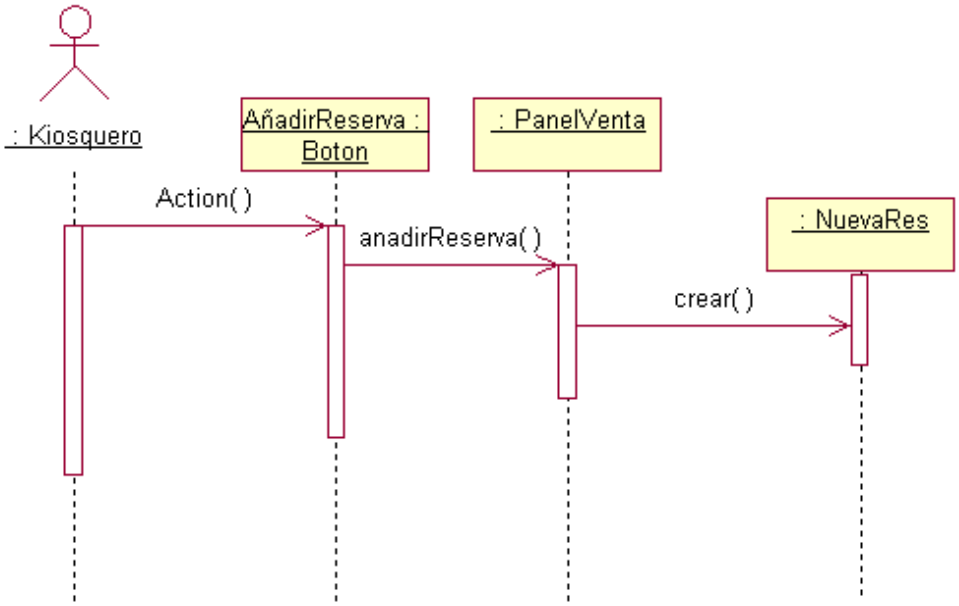
Context panelVenta :: BotonCancelarComp.Action()
Post: self.listaComp.linea->size() = 0

-- Vender artículos de la lista

Context panelVenta :: venderLista ()
Post: self.listaComp.linea->size() = 0



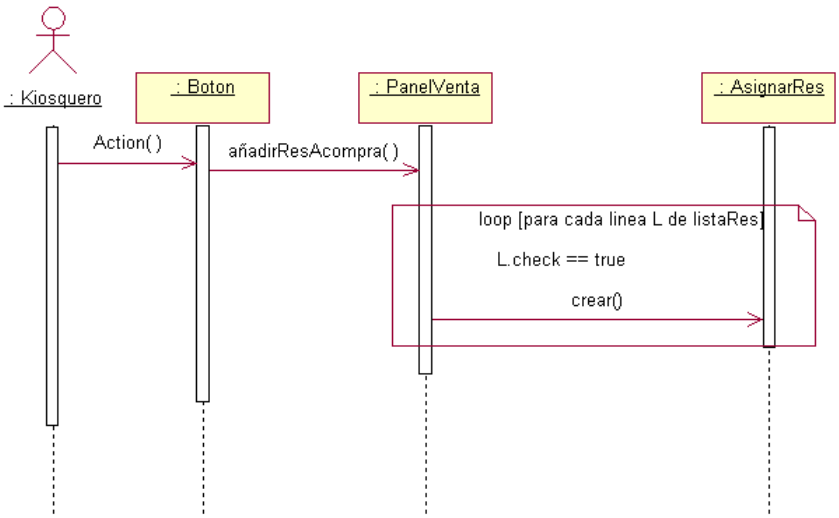
-- Añadir una reserva



-- Enviar las reservas seleccionadas a la lista de compra

Context panelVentana :: BotonAnadirResACompra.action()

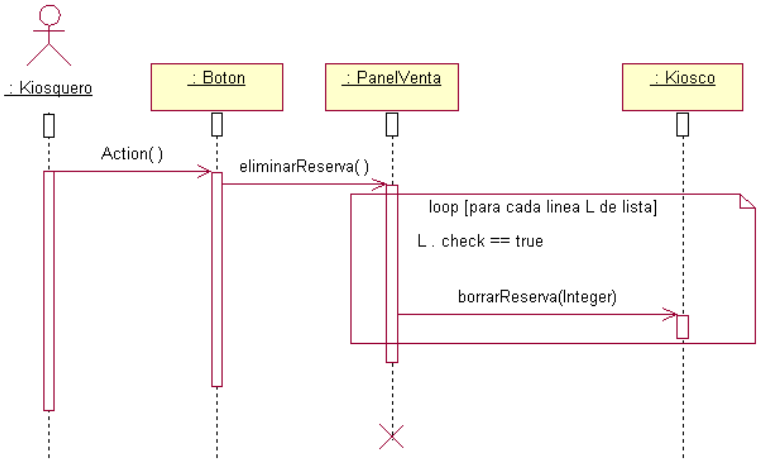
Pre: self.listaRes.seleccionada->size() > 0



-- Eliminar una reserva

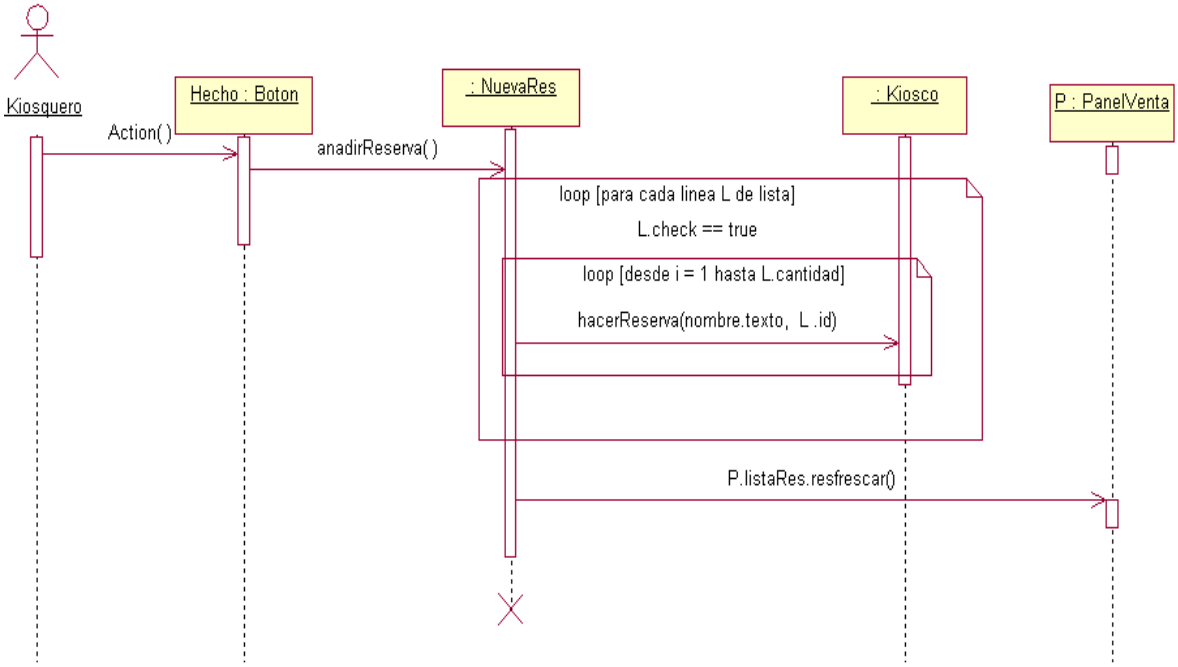
Context panelVenta :: BotonEliminarReserva.Action()

Pre: self.listaRes.seleccionada->size() > 0

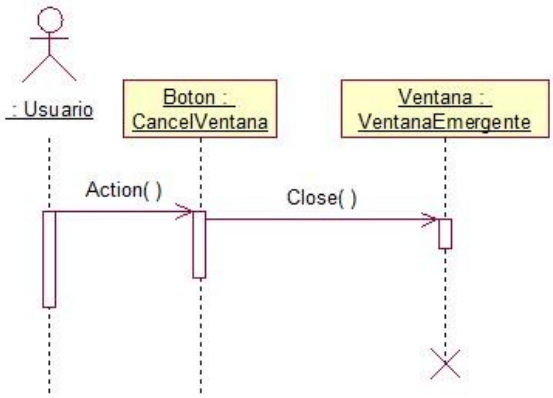


Funcionalidades de nuevaRes (ventana de creación de nueva reserva)

-- Aceptar la nueva reserva

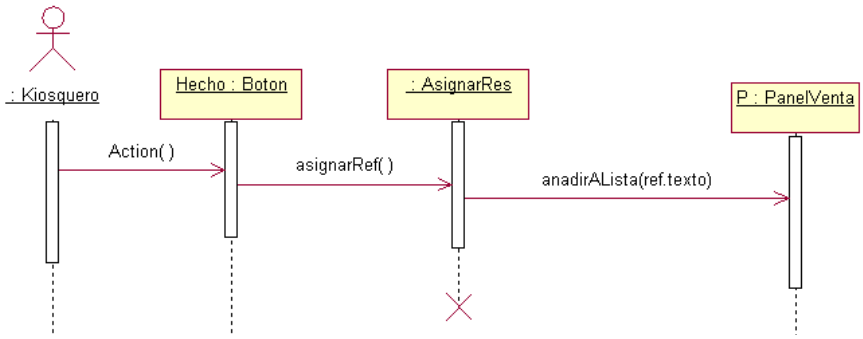


-- Cancelar (vuelve a la pestaña vender)



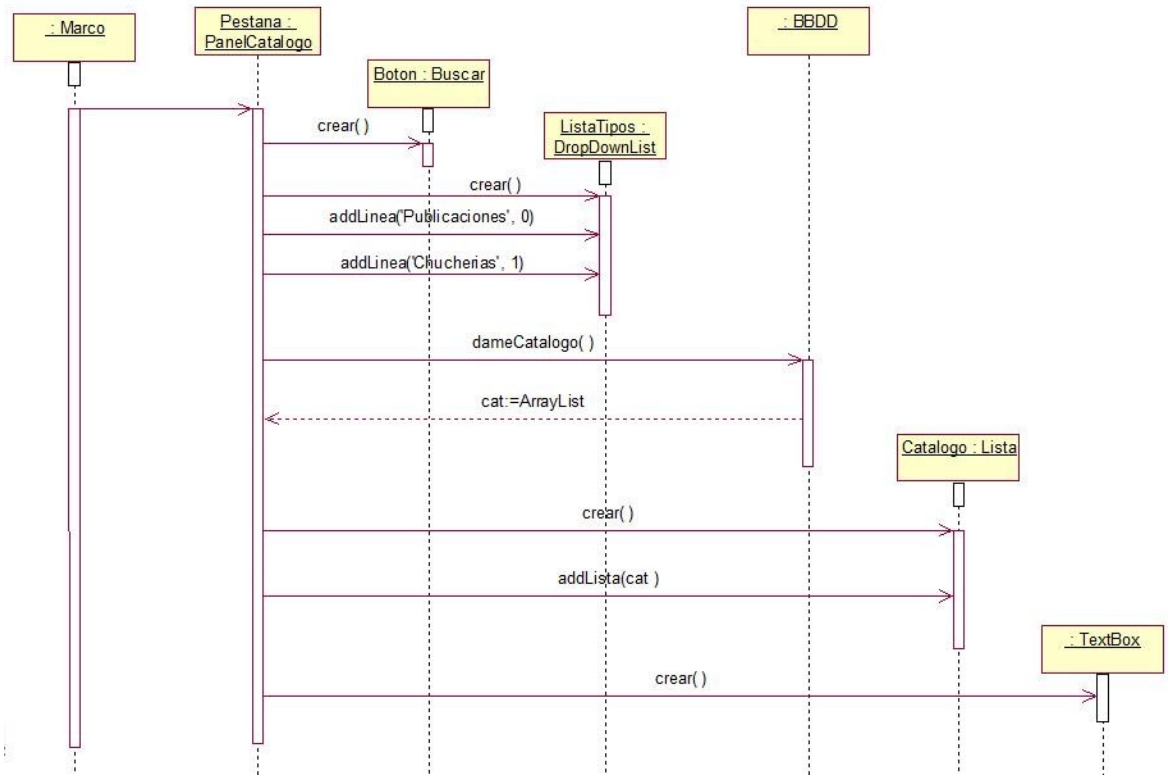
Funcionalidades de asignarRes (Ventana de venta de reserva)

-- El botón *Hecho* introduce el artículo en la lista de venta



Cancelar es análogo al *Cancelar* de nuevaRes.

Ejemplo de creación de pestaña: Catálogo



El selector de fecha

Este componente se utiliza en distintas pestañas. Permite al usuario indicar una fecha concreta de manera intuitiva. Su comportamiento es el siguiente:

Context SelectorFecha::diaUp.Action()
Body: self.suma(1, "dia")

Context SelectorFecha::diaDown.Action()
Body: self.suma(-1, "dia")

Context SelectorFecha::mesUp.Action()
Body: self.suma(1, "mes")

Context SelectorFecha::mesDown.Action()
Body: self.suma(-1, "mes")

Context SelectorFecha::añoUp.Action()
Body: self.suma(1, "año")

Context SelectorFecha::añoDown.Action()
Body: self.suma(-1, "año")

Descripción de Recursos Físicos

Diagrama de Componentes

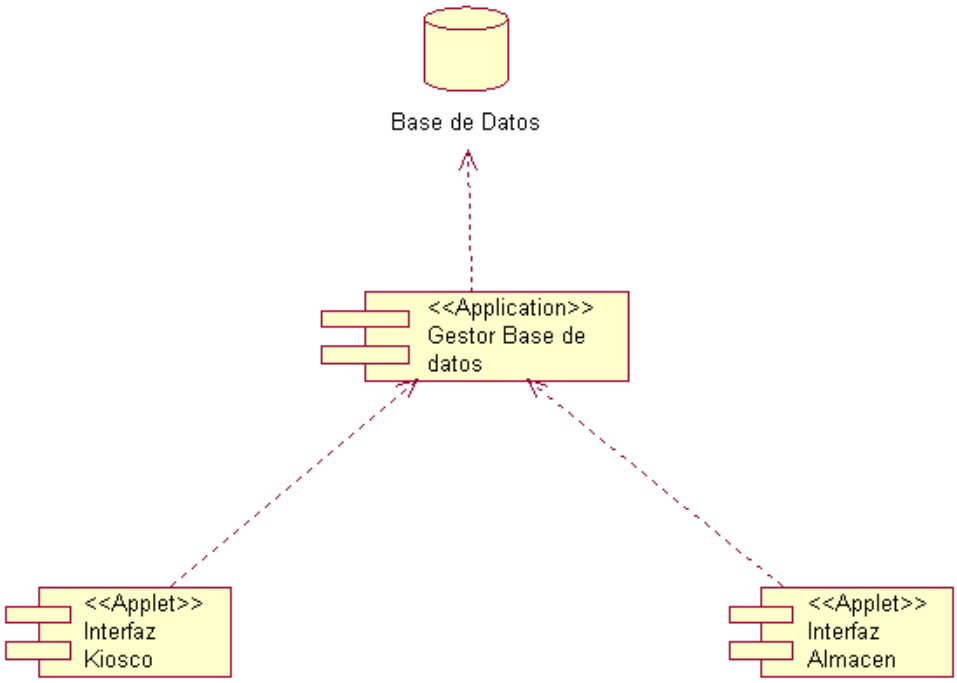
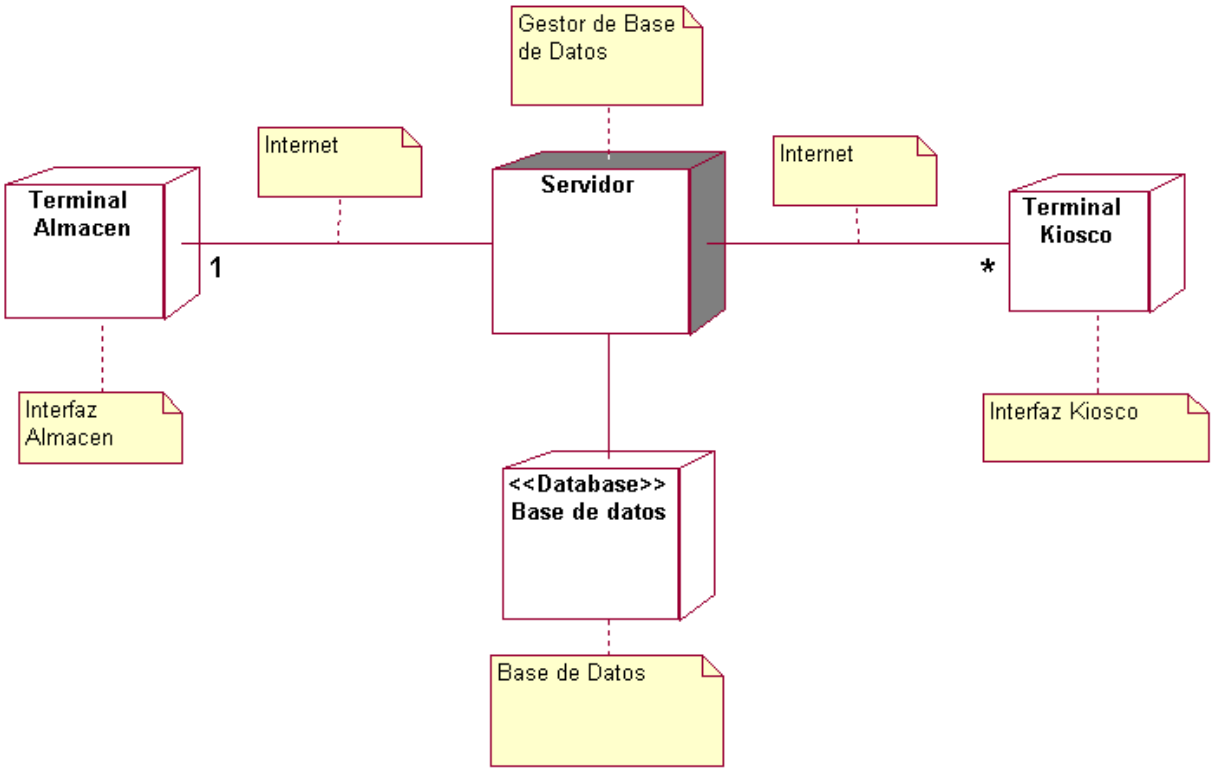


Diagrama de Despliegue



Pruebas

- **Módulo/unidad:** A medida que se van implementado las clases que forman nuestra aplicación se tiene que comprobar que su funcionamiento es el correcto. Para ello, cada vez que una nueva clase se termina de programar se realizan una serie de pruebas:
    - Se ejecutan uno a uno los métodos de la clase y se monitorizan las salidas para confirmar que son correctas
    - Si la clase tiene que comunicarse con alguna otra, se implementan unas interfaces genéricas para imitar el comportamiento de la clase de la que hay que usar métodos o enviar/recibir respuestas para que siempre devuelva unos valores tipo.

  En nuestro proyecto concreto probamos las constructoras, destructoras, modificadores y accesores de todas las clases. Además se prueban todos los métodos de Kiosco, Almacén (Suponiendo que a la hora de implementar estas clases del modelo se corresponden con clases reales) y del gestor de la Base de Datos, y su comunicación entre si, en los casos en los que sea aplicable.

- **Integración**: Una vez nuestros componentes han pasado satisfactoriamente las pruebas de módulo/unidad se arma nuestro sistema y se comprueba que lo que antes funcionaba correctamente módulo a módulo sigue funcionando correctamente. Se repiten las mismas pruebas aunque sin utilizar interfaces genéricas ya que disponemos de las clases reales.

- **Sistema/Carga:** Se comprueba que el sistema puede atender las suficientes peticiones por parte de varios usuarios simultáneamente. En nuestro caso, creamos una cantidad de Kioscos razonable y hacemos que efectúen pedidos, reservas, ventas… Más adelante se detallan una serie de pruebas de aceptación que serian igualmente validas como pruebas de carga si se aumentase la cantidad de usuarios de tipo Kiosco involucrados en ellas.
  Además, el sistema debe ser escalable y funcionar incluso con volúmenes grandes de datos, ya que la cantidad de artículos que tendrá que almacenar la base de datos crecerá rápidamente. Comprobaremos que funciona correctamente introduciendo una gran cantidad de artículos y Kioscos, en la base de datos y probando a realizar las acciones más comunes del sistema. El funcionamiento debería mantenerse dentro de unos límites aceptables.
  Probamos la estabilidad del sistema contra errores inesperados. Ya que nuestra aplicación esta pensada para trabajar en modo cliente-servidor a traves de internet, es posible que se produzcan cortes de conexión que habra que gestionar. Creamos un escenario de pruebas en el que haya un Kiosco y un Almacen conectados a la base de datos haciendo diferentes tareas de consulta y actualizacion(Consultar albaranes o pedidos, vender articulos). Desactivamos la conexión. El sistema debe permanecer funcionando, avisando al usuario correspondiente de que no se puede conectar con el servidor, sin realizar ninguna actualizacion sobre la base de datos porque ésta no esta accesible. Puesto que todas las operaciones de la aplicación son consultas o actualizaciones de la base de datos el programa se queda a la espera de recuperar la conexión.

- **Usabilidad:** Probamos que la aplicación desarrollada no sea demasiado difícil o ineficiente de utilizar desde el punto de vista de la interfaz. Para ello realizaremos una serie de tareas para simular el comportamiento más común de nuestro sistema y comprobaremos el tiempo que unos usuarios tardan en realizarlas, o las dificultades que se encuentren en el proceso. Podemos o bien pedirle a los testeadores que realicen una serie de tareas (Crear un artículo, Editarlo, Vender) y dejarles que investiguen la forma de hacerlo o bien darles una serie de pasos a seguir. Podemos usar las pruebas de aceptación detalladas más adelante (exceptuando aquellos pasos que dan ERROR) para comprobar la sencillez o complejidad de la interfaz. Por ejemplo, algunas posibles pruebas seria ver cuanto tarda un usuario en hacer la secuencia de Abrir la pestaña Catalogo, hacer clic en el botón Crear Nuevo Tipo Articulo, rellenar los datos y hacer Clic en Aceptar.

- **Regresión:** Siempre que introducimos una modificación en nuestro sistema debemos repetir las pruebas realizadas anteriormente para comprobar que nuestro sistema tiene, como mínimo las funcionalidades que presentaba en anteriores versiones. Dado que nuestro proyecto tiene una única versión asumiremos que cualquier fallo por cambios en el programa se detecta en alguna de las otras pruebas.

- **Aceptación:** En esta categoría vamos a probar diferentes aspectos de nuestro sistema para comprobar que el comportamiento del mismo se corresponde en la mayor medida posible con el comportamiento que el cliente nos ha solicitado

PASOS A SEGUIR Y SALIDA ESPERADA	REQUISITOS
1. Abrir sesión Almacén. 2. Cambiar contraseña a "123456" 3. Comprobar saldo (0). 4. Aumentar saldo en 5000€. 5. Comprobar saldo (5000). 6. Añadir un nuevo tipo de artículo genérico de nombre A1 al catálogo. 7. Comprobar que se ha añadido correctamente. 8. Añadir un nuevo tipo de artículo genérico de nombre A1 al catálogo (ERROR, tipo repetido). 9. Comprobar que no se ha añadido. 10. Añadir un nuevo tipo de artículo publicación de nombre P1 y fecha de publicación 1-1-2008 al catálogo. 11. Comprobar que se ha añadido correctamente. 12. Añadir un nuevo tipo de artículo publicación de nombre P1 y fecha de publicación 1-1-2008 al catálogo (ERROR, tipo repetido). 13. Comprobar que no se ha añadido. 14. Añadir un nuevo tipo de artículo chuchería de nombre C1 y fecha de caducidad 31-12-2220 al catálogo. 15. Comprobar que se ha añadido correctamente. 16. Añadir un nuevo tipo de artículo chuchería de nombre C1 y fecha de caducidad 31-12-2220 al catálogo (ERROR, tipo repetido). 17. Comprobar que no se ha añadido. 18. Añadir un nuevo tipo de artículo chuchería de nombre C1 y fecha de caducidad 1-1-2100 al catálogo. 19. Comprobar que se ha añadido 20. Cerrar sesión. 21. Abrir sesión Almacén. 22. Comprobar que los artículos añadidos continúan en catalogo.	A1, A2, A3, A4, A5, A6, A8, A19 Ar1, Ar2, Ar4, Ar5, Ar7, Ar8
Partiendo del punto 22 de la prueba anterior: 1. Modificar el PCA y el PVP de A1 a 1€ y 3€ respectivamente 2. Modificar el PCA y el PVP de P1 a 2€ y 4€ respectivamente 3. Añadir 1000 artículos de tipo A1 y 2500 artículos de tipo P1 al inventario del almacén (Las referencias de los artículos deben ser distintas entre si) 4. Comprobar saldo del Almacén (-1000) 5. Comprobar inventario del almacén (A1 = 1000 ; P1 = 2500) 6. Modificar el PVK y el PDev del Almacén a un 10% y 5% 7. Cerrar sesión	A7, A9, A12, A13 Ar3, Ar6, Ar7, Ar9
Partiendo del punto 6 de la prueba anterior: 1. Dar de alta a un Kiosco con identificador Kiosco_01 2. Dar de alta a un Kiosco con identificador Kiosco_02 3. Dar de alta a un Kiosco con identificador Kiosco_01 (ERROR, Identificador duplicado) 4. Comprobar lista de kioscos (Kiosco_01, Kiosco_02) 5. Incrementar el Saldo de Kiosco_01 en 1000€ 6. Incrementar el Saldo de Kiosco_02 en 2000€ 7. Comprobar los inventarios de Kiosco_01 y Kiosco_02 (vacíos) 8. Comprobar el saldo de Kiosco_01 (1000) 9. Comprobar el saldo de Kiosco_02 (2000) 10. Cerrar Sesión	A14, A15, A16, A17, A20

Partiendo del punto 9 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: 1. **Iniciar sesión como Kiosco_03 (Error, Usuario inexistente)** **2. Iniciar sesión como Kiosco_01** **3. Cambiar contraseña a "paco2008"** 4. **Comprobar Saldo propio (1000)** 5. **Comprobar inventario del Almacén (A1 1000unidades; P1 2500unidades; C1 agotado)** **6. Hacer un pedido de 454 P1 al Almacén para el día 2-1-2009** 7. **Comprobar el pedido (998,8€ de pedido)** **8. Modificar el pedido anterior añadiendo un articulo A1** **9. Comprobar el pedido (999,9€ de pedido)** 10. **Modificar el pedido añadiendo un articulo P1 (ERROR, no hay saldo suficiente)** **11. Comprobar el pedido (454 P1, 1 A1)** 12. **Hacer un pedido para el día 3-1-2009 de una unidad de P1 (ERROR, no hay saldo suficiente)**	A8 K1, K2, K3, K5, K6, K8, K9, K11
Partiendo del punto 12 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: **1. Consultar el pedido existente (454 P1, 1 A1)** **2. Eliminar el articulo A1 de dicho pedido antes de las 23.00 del día 1-1-2009** **3. Confirmar el pedido** **4. Crear un pedido para el día 4-1-2009 de un artículo A1** **5. Comprobar que se ha añadido correctamente** **6. Eliminar el pedido del día 4-1-2009** **7. Comprobar que se ha eliminado** **8. Añadir un articulo de A1 al pedido después de las 23.00 del día 1-1-2009 (ERROR, pedido ya registrado)** **9.** Cambiar el PCA de P1 a 4€ **10. Eliminar un articulo de P1 del pedido (ERROR, pedido ya registrado)** **11. Comprobar la lista de pedidos (2-1-2009)** **12. Consultar el pedido 2-1-2009 (454 P1)** **13.** Comprobar la lista de albaranes (2-1-2009; 998,8€, 454 artículos P1, Kiosco_01) (El precio de P1 se mantiene como estaba antes de la prueba anterior) **14. Desde Kiosco_01 comprobar la lista de albaranes (2-1-2009; 998,8€, 454 artículos P1)** **15. Desde Kiosco_02 comprobar la lista de albaranes (vacía)** **16.** Comprobar el inventario del almacén (1000 A1; 2500 P1) **17. Desde Kiosco_01 comprobar el inventario propio (vacío)**	K9, K10 As1, As2 A19 K13, K18

Partiendo del punto 5 de la prueba anterior, considerando que la fecha actual es 2-1-2009 **(En negrita las acciones desde las vistas del Kiosco)**: 1. **Desde Kiosco_01 confirmar la llegada de los artículos del albaran creado en la prueba anterior** 2. **Comprobar inventario propio (454 P1)** 3. **Comprobar saldo propio (1,2€)** 4. Comprobar saldo del Almacén (-1,2€) 5. Comprobar el inventario del almacén (1000 A1; 2046 P1)	Ar10 K7, K14, K15, K17, K19
Partiendo del punto 4 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: 1. **Desde Kiosco_01 comprobar inventario propio (454 P1)** 2. **Comprobar saldo propio (1,2€)** 3. **Vender un articulo P1** 4. **Comprobar inventario propio (453 P1)** 5. **Comprobar saldo propio (5,2€)** 6. **Vender un artículo P3 (articulo con una referencia inexistente) (ERROR, articulo no existente)** 7. **Vender un articulo A1 (ERROR, articulo agotado)** 8. Comprobar lista de artículos vendidos(1 P1 Kiosco_01)	K4, K20, K21, K22
Partiendo del punto 7 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: 1. Modificar el PCA y el PVP de C1 a 1€ y 2€ respectivamente 2. Añadir 100 artículos de tipo C1 al catalogo del Almacén 3. Aumentar el saldo de Kiosco_01 en 1000€ 4. **Kiosco_01: Hacer un pedido de 100 artículos C1 para el día siguiente** 5. **Kiosco_02: Hacer un pedido de 100 artículos C1 para el día siguiente** 6. Comprobar lista de pedidos (Kiosco_01 Día_siguiente 100 C1; Kiosco_02 Día_siguiente 100 C1) 7. Comprobar que se han generado dos albaranes con 50 artículos C1. 8. **Kiosco_01: Eliminar el pedido para el dia siguiente (ERROR, pedido ya atendido)** 9. **Kiosco_01: No aceptar la llegada de los 50 artículos** 10. **Comprobar que no se han añadido al inventario de Kiosco_01** 11. Comprobar que se han eliminado del inventario del Almacén 12. Comprobar que los artículos se han añadido a la lista de extraviados	A18 As3 K12, K16
Partiendo del punto 6 de la prueba anterior, y habiendo aceptado el envío de los artículos **(En negrita las acciones desde las vistas del Kiosco)**: 1. **Kiosco_02: Reservar 49 artículos C1 a nombre de Bob** 2. **Reservar 1 articulo C1 a nombre de Alice** 3. **Comprobar lista de reservas (Bob 49 C1; Alice 1 C1)** 4. **Vender un articulo C1 a Mary (ERROR, no tiene reserva y no hay suficientes artículos)** 5. **Vender un articulo C1 a Alice** 6. **Comprobar lista de reservas (Bob 49 C1)** 7. **Vender un articulo C1 a Bob** 8. **Comprobar lista de reservas (Bob 48 C1)** 9. **Eliminar una reserva de Bob** 10. **Comprobar lista de reservas (Bob 47 C1)**	K23, K24, K25, K26

11. **Vender un articulo C1 a Mary** 12. Comprobar que la venta se ha realizado	
Partiendo del punto 10 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: **1. Kiosco_02: Comprobar inventario (47 C1)** **2. Devolver 23 artículos de tipo C1 al almacén** **3. Comprobar que se ha generado el albarán correctamente** 4. Aceptar la devolución de los 23 artículos 5. **Kiosco_02:Comprobar inventario (vacío)** 6. Comprobar que el saldo del almacén se ha actualizado correctamente 7. Comprobar que el saldo de Kiosco_02 se ha actualizado correctamente 8. Comprobar el inventario del almacén (1000 A1; 2046 P1; 23 C1)	Ar11 K27
Partiendo del punto 8 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: **1. Kiosco_02: Realizar un pedido de un articulo C1** 2. **Comprobar lista de reservas (Bob 47 C1)** 3. Dar de baja al Kiosco_02 4. Comprobar pedidos(vacío) 5. Comprobar lista de Kioscos(Kiosco_01) 6. Comprobar que los artículos del inventario del Kiosco_02 han ido a parar al almacén 7. **Kiosco_02: Consultar saldo propio (ERROR, conexión terminada)** 8. **Conectarse como Kiosco_02 (ERROR, Kiosco inexistente)** 9. Dar de alta al Kiosco_02 10. **Kiosco_02: Comprobar lista de reservas (vacía)**	A21, A22, A23
Partiendo del punto 9 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: 1. Descatalogar el tipo de articulo C1 **2. Kiosco_01: Vender un artículo C1** 3. Comprobar lista de artículos vendidos 4. Comprobar inventario del Almacén (1000 A1; 2046 P1) 5. Comprobar catalogo (A1, P1, C1 Descatalogado) 6. **Kiosco_02: Realizar un pedido de 100 artículos de C1 (ERROR, articulo descatalogado)** **Avanzamos la fecha del sistema hasta la fecha de caducidad del tipo de articulo C1** 7. Comprobamos el catalogo (A1, P1) **8. Kiosco_01: Comprobamos que no quedan C1 en el inventario**	A10, A11 Ar12, Ar13

Partiendo del punto 8 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: 1. Crear el tipo de articulo A3 con PCA de 1€ y PVP de 3€ 2. Añadir 100 unidades de A3 al inventario del almacen 3. Aumentar el saldo de Kiosco_02 en 1000€ 4. **Kiosco_02: Realizar un pedido de 50 unidades de A3** 5. Descatalogar el tipo de articulo A3 6. Comprobar la lista de articulos descatalogados/caducados(46 C1, 50 A3)(Los articulos de tipo A3 del pedido no se descatalogan puesto que ya estaban en un albaran) 7. **Kiosco_02: Aceptar el albaran del pedido** 8. **Kiosco_02: Comprobar inventario(50 A3)** 9. **Kiosco_02: Vender 1 A3** 10. **Kiosco_02: Comprobar lista de articulos vendidos** 11. **Kiosco_02: Devolver 49 articulos de tipo A3 (ERROR, articulos descatalogados no se pueden devolver)**	A10
Partiendo del punto 10 de la prueba anterior **(En negrita las acciones desde las vistas del Kiosco)**: 1. Dar de alta a Kiosco_03 2. Añadir un nuevo tipo de artículo genérico de nombre A4 al catálogo. 3. Añadir un nuevo articulo de tipo A4 al inventario 4. **Kiosco_01: Realizar un pedido de 1 A4** 5. **Kiosco_02: Realizar un pedido de 1 A4** 6. **Kiosco_03: Realizar un pedido de 1 A4** 7. Comprobar que los albaranes que se crean están vacíos, ya que no se pueden repartir fracciones de articulo	As3 A19