

Documento de DISEÑO



**Sistema de Administración y
Gestión de RESTaurantes**



Adrián Víctor Pérez Lopera
Sergio Rodríguez Lumley
Nabil Sabeg

V1.3

ÍNDICE DE CONTENIDO

Apartado de control de versiones.....	4
1. Estilo arquitectónico.....	5
2. Diagrama de clases del diseño.....	6
<i>Descripción de clases.....</i>	<i>7</i>
Clase Factura:.....	7
Clase restaurante:.....	7
Clase Pedido.....	8
Clase ElementoPedido.....	8
3. Diagrama de paquetes estructurales.....	9
4. Estilo arquitectónico detallado.....	11
5. Identificación de las interfaces.....	13
6. Diagrama de componentes.....	19
7. Diagramas de colaboración.....	20
<i>Funciones de base de datos.....</i>	<i>20</i>
actualizaPedido.....	20
obtienePedidosNoFacturados.....	21
obtieneSecciones.....	22
<i>getSiguientePedidoBar.....</i>	<i>23</i>
<i>getSiguientePedidoCocinaEnCola.....</i>	<i>24</i>
<i>getPedidosCocinaPreparandose.....</i>	<i>25</i>
<i>seleccionaBebida.....</i>	<i>26</i>
<i>seleccionaPlato.....</i>	<i>27</i>
<i>getNumPlatosEnCola.....</i>	<i>28</i>
<i>getNumBebidasEnCola.....</i>	<i>29</i>
<i>nuevoPedido.....</i>	<i>30</i>
<i>InsertaPedido.....</i>	<i>31</i>
<i>modificaPedido.....</i>	<i>32</i>
<i>eliminaPedido.....</i>	<i>33</i>
<i>getPedidosMesaModificables.....</i>	<i>34</i>
<i>getPedidosMesaModificablesBD.....</i>	<i>35</i>
<i>getElementosPedido.....</i>	<i>36</i>

<i>getElementosPedido.....</i>	<i>37</i>
<i>getSecciones.....</i>	<i>38</i>
<i>imprimeFactura.....</i>	<i>39</i>
<i>pideFactura.....</i>	<i>40</i>
<i>confirmaPagoFactura.....</i>	<i>41</i>
<i>getFacturasEnCola.....</i>	<i>41</i>
<i>getFacturasImprimidas.....</i>	<i>42</i>
<i>getFactura.....</i>	<i>43</i>
8. Diseño de la base de datos.....	44
9. Diseño de las interfaces de usuario.....	45
<i>Interfaz de usuario de Cliente.....</i>	<i>46</i>
<i>Interfaz de usuario de Metre.....</i>	<i>49</i>
<i>Interfaz de usuario de Cocinero.....</i>	<i>51</i>
<i>Formato de factura.....</i>	<i>52</i>
10. Consideraciones finales sobre el diseño.....	53
Apéndice 1.0.....	54
Apéndice 1.1.....	55
Apéndice 1.2.....	56

Sagres

APARTADO DE CONTROL DE VERSIONES

Todas las versiones están especificadas a fondo en el apartado de “Apéndices”, al final de este documento, cada apéndice se corresponde en nombre con su número de versión. Por ejemplo, el “Apéndice 0.1” se corresponde con la versión v0.1. Para ver los cambios realizados sobre cada versión, hay que ir deshaciendo los cambios desde el final.

<i>Versión</i>	<i>Fecha</i>	<i>Descripción</i>
V1.0	02/05/10	Se ha generado el documento de diseño.
V1.1	07/05/10	Se han realizado grandes cambios en todos los apartados. Se ha especificado el sistema en mayor detalle.
V1.2	13/05/10	Se han realizado actualizaciones en varios apartados.
V1.3	18/05/10	Se han modificado ciertos diagramas de colaboración y de componentes.

Sagres

1. ESTILO ARQUITECTÓNICO

Para esta iteración, utilizaremos de nuevo una arquitectura basada en capas cerradas que separen la aplicación en tres niveles: interfaz de usuario, lógica de aplicación y servicios. Esto mejora la estructura del sistema y fomenta la flexibilidad en cuanto a sustitución de servicios o interfaces de usuario.

Por otra parte se usará también una arquitectura MVC (Modelo-Vista-Controlador) para desacoplar la lógica de la aplicación del interfaz de usuario. Esto facilita el diseño de distintas interfaces de usuarios adaptados a los distintos terminales, una para los clientes, otra para la cocina y otra para la barra. Con esto obtenemos un alto nivel de cohesión y un bajo nivel de acoplamiento, separando datos de todo lo demás, ya que cambios en nuestro diagrama de clases no repercutirán en cambios en la interfaz de usuario y viceversa.

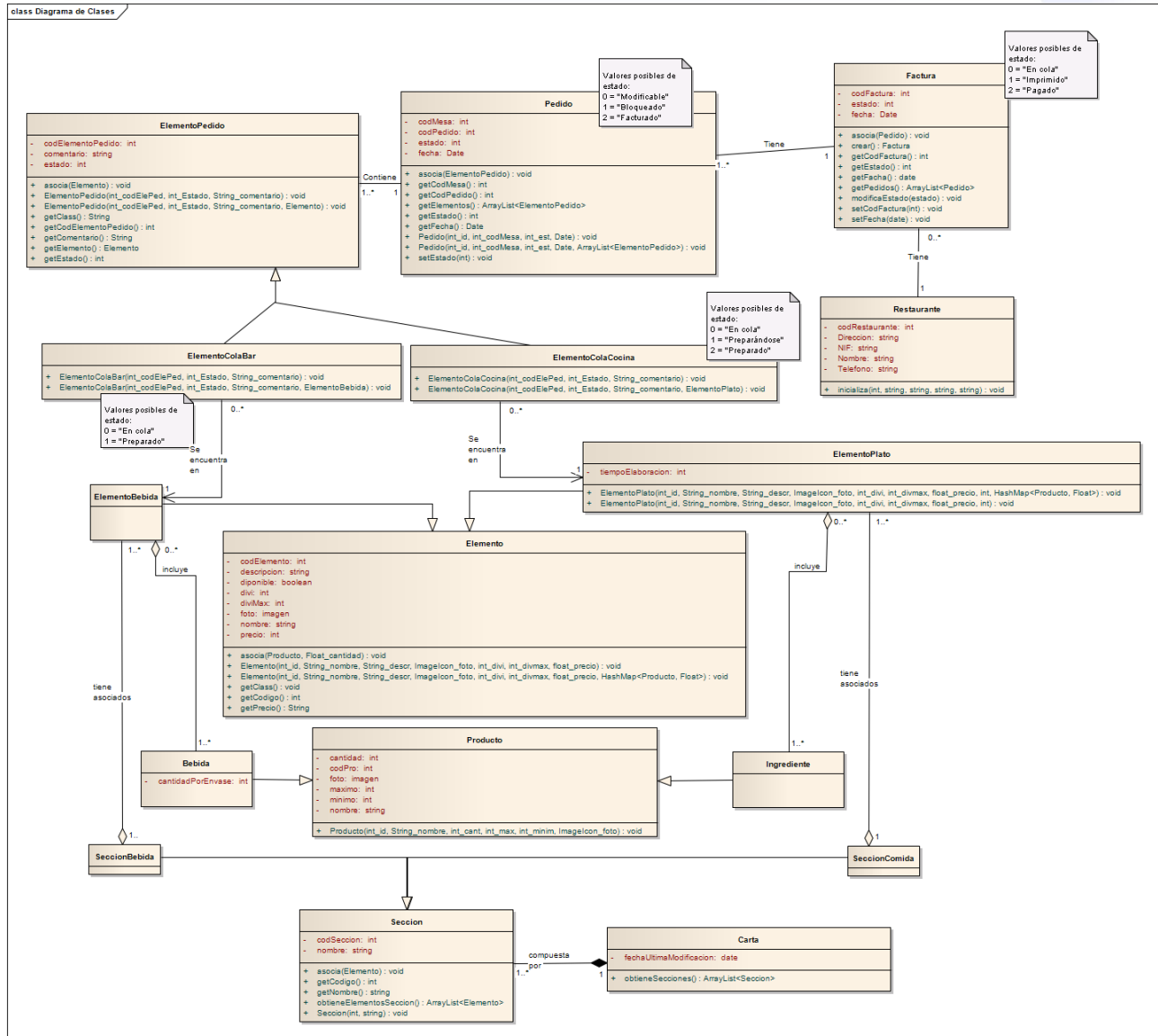
El estilo cliente-servidor podríamos utilizarlo para establecer la comunicación entre los subsistemas de cada capa que pueden estar situados en distintos nodos del sistema. Esto permite distribuir los distintos subsistemas del sistema reduciendo el acoplamiento sólo a través del protocolo de comunicaciones. Fomenta la interoperabilidad, y facilita la escalabilidad del sistema (por ejemplo, aumentar el número de clientes sin penalizar la gestión de datos).



En los apartados siguientes mostraremos, entre otras cuestiones, cómo adaptaremos esta arquitectura conforme a las necesidades de nuestro sistema.

2. DIAGRAMA DE CLASES DEL DISEÑO

En este apartado mostramos el diagrama de clases de diseño, el cuál obtenemos a partir del análogo en la etapa de análisis, añadiendo a cada clase los métodos obtenidos en los diagramas de colaboración de dicho análisis.



Descripción de clases

Clase Factura:

Esta clase representa una factura y varios pedidos relacionados con ella. Se ha hecho esta clase para permitir relacionar varios pedidos con la misma factura.

- Atributos:
 - codFactura: Es el identificador único de cada instancia de Factura.
 - Estado: nos dice en que estado se encuentra una factura, valores posibles son:
 - en_cola=0: significa que un cliente pidió la factura.
 - Imprimido=1: significa que se ha imprimido la factura.
 - Pagado: significa que se ha pagado la factura.
 - Fecha: fecha de pago de la factura.
- Operaciones:
 - crear(): crea una instancia de la clase.
 - modificaEstado(estado:int): modifica el estado de una factura con el valor del parámetro, valores posibles son : 0,1,2.

Clase restaurante:

Esta clase contendrá la información del restaurante necesaria, se ha usado para obtener información necesaria para generar una factura. Esta clase tendrá una sola instancia.

- Atributos:
 - codRestaurante: es el identificador de la clase.
 - Dirección: dirección del restaurante.
 - NIF: Numero de identificación fiscal del restaurante.
 - Nombre: nombre del restaurante.
 - Telefono: telefono del restaurante.
- Operaciones:
 - inicializa(int,string,string,string,string): inicializa la única instancia de esta clase.

Clase Pedido

Esta clase representa un pedido de cliente. Este lleva guardada toda la información relevante que un pedido pueda tener, es decir, un **estado**, que indica si el pedido es modificable, si se está preparando o preparado o bien si ya ha sido facturado; también un ArrayList de “ElementoPedido”, que son los elementos de los que se compone un pedido.

Clase ElementoPedido

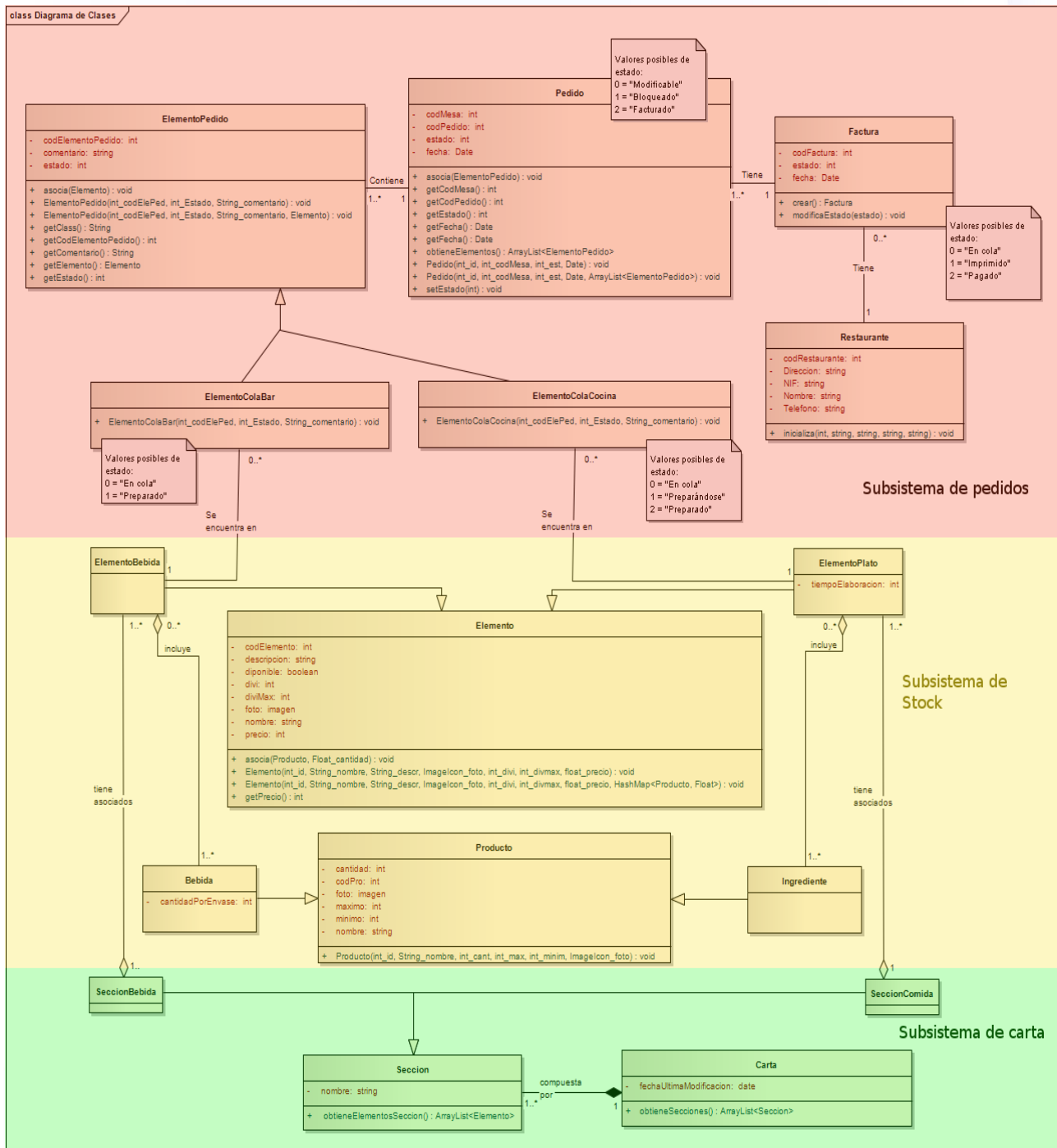
Es la clase de la cual se definen los ElementoColaCocina y ElementoColaBar. Este elemento guarda dentro su **estado**, que tiene un significado y valores distintos para cada tipo de elemento (Plato o Bebida). Dentro, cada ElementoColaCocina y ElementoColaBar tiene la definición de lo que consiste el plato o la bebida, que son las clases ElementoBebida y ElementoPlato. Ambos heredan de la clase Elemento, que engloba toda su funcionalidad y atributos. Estos, a su vez, tienen asociados una lista de Productos los cuales indican de qué está compuesto un Elemento.

A excepción de las directamente relacionadas con ElementoPedido (es decir, ElementoColaCocina y ElementoColaBar) el resto de clases fueron definidas e implementadas en la primera iteración. Siguiendo sus diagramas hemos estructurado nuestro sistema para hacerlo acorde al suyo, ya que la revisión planificada de la primera iteración complicó esta situación.

Sagres

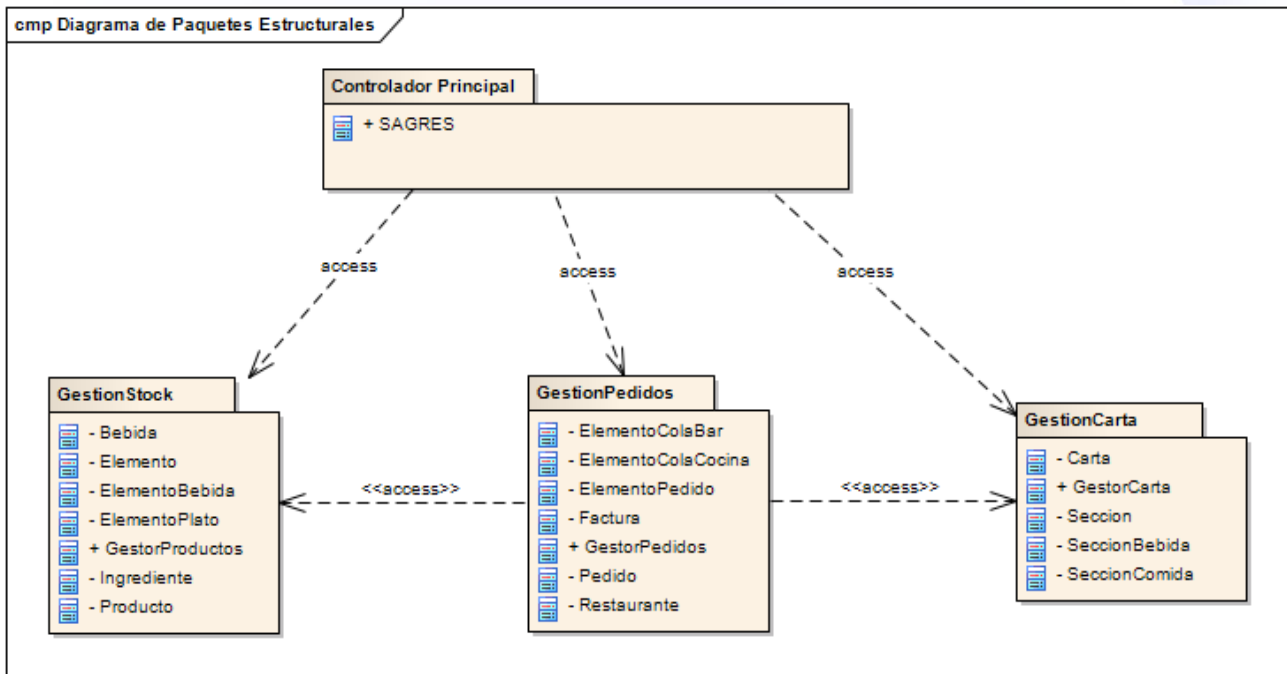
3. DIAGRAMA DE PAQUETES ESTRUCTURALES

Considerando el diagrama de clases del diseño como punto de partida, el primer paso que tomaremos es realizar la división del mismo en subsistemas estructurales. En el diagrama de paquetes funcionales obtenido en el modelado de requisitos de esta iteración concluimos que el sistema a desarrollar en esta iteración estaría compuesto por un sólo subsistema funcional, el subsistema de gestión de pedidos. Teniendo en cuenta que en el diagrama de clases tuvimos la necesidad de añadir clases procedentes de subsistemas desarrollados en la anterior iteración, la distribución en subsistemas estructurales, en el cual sólo se deberá tener en cuenta la disposición de las clases (y no de sus atributos u operaciones) será la siguiente:



Se observa como además del subsistema de gestión de pedidos, que es el que lleva a cabo las funcionalidades requeridas en esta iteración, se tienen en cuenta también los subsistemas de stock y de carta, ya que necesitaremos reutilizar y/o desarrollar algunas de sus funcionalidades.

A partir de esta división del diagrama de clases obtenemos el siguiente diagrama de paquetes estructurales, donde se introducen sólo aquellas conexiones entre paquetes que son necesarias para las funcionalidades a desarrollar en esta iteración:

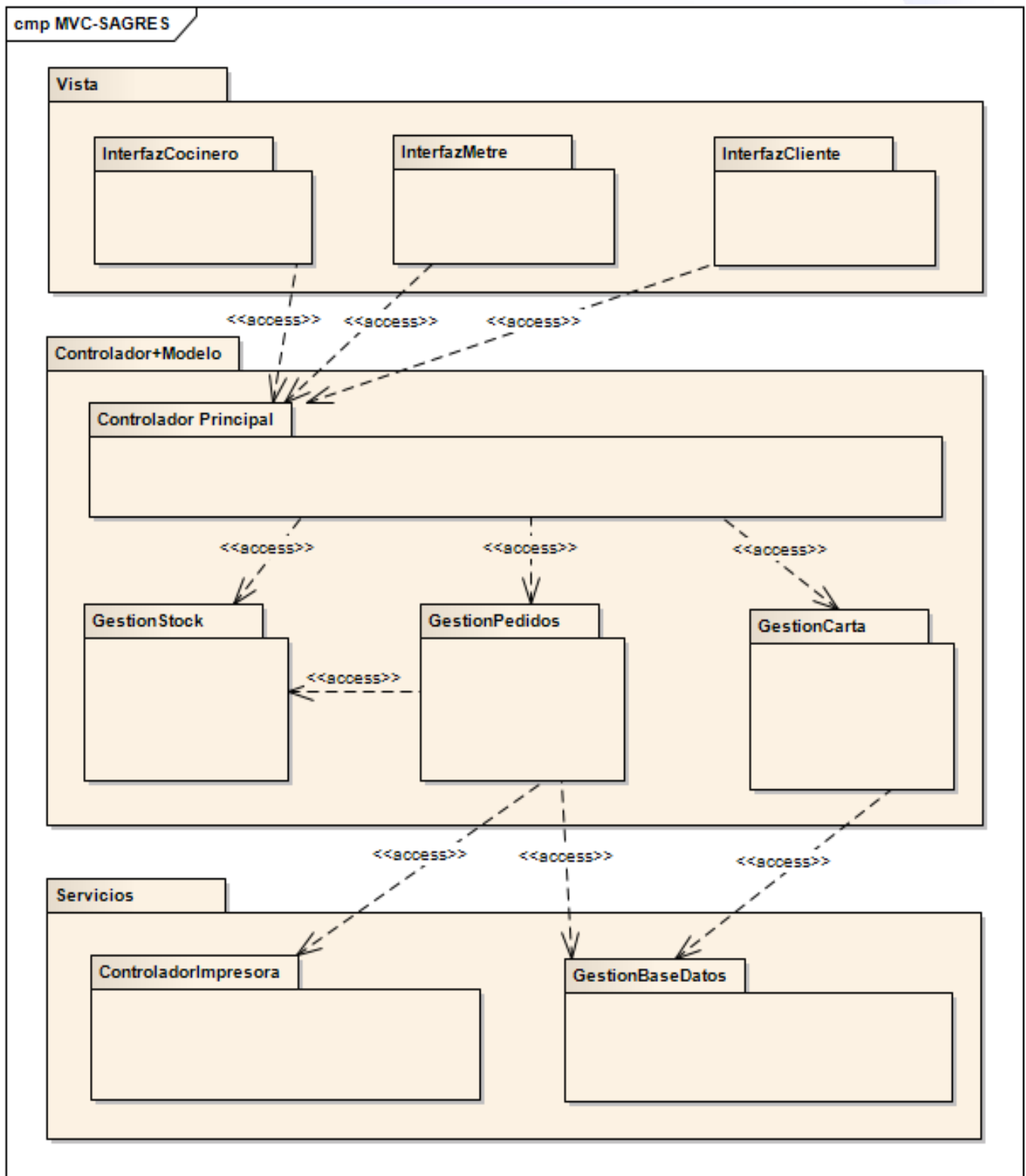


Observar que se ha introducido por vez primera el paquete con el controlador principal, con una clase tipo control SAGRES. En la fase de análisis se consideró en los diagramas de colaboración un controlador genérico SAGRES, para poder definir las operaciones del sistema un nivel conceptual de abstracción. En correspondencia con dicho controlador genérico, el controlador principal mostrado en este momento será en el sistema un componente más a desarrollar, ya que constituirá la fachada la lógica de nuestra aplicación.

Observar además que en los paquetes estructurales se han incluido clases gestoras. Estas clases de tipo control se encargarán de llevar a cabo los servicios que ofrecen sus respectivos paquetes. Se puede ver que el acceso de a las clases en los paquetes se restringe, de manera que para poder acceder a las funcionalidades que ofrecen hay que comunicarse con las clases de control.

4. ESTILO ARQUITECTÓNICO DETALLADO

Una vez definido el diagrama de paquetes estructurales, podemos definir de manera detallada cuál será la arquitectura del sistema. A continuación se muestra como se distribuirán los distintos subsistemas en las capas del MVC que hemos decidido adoptar:



Se observa que en la **vista** tenemos tres paquetes, cada uno de los cuales correspondiente a un grupo de interfaces con el que tendrá interacción un tipo de actor. Tendremos uno para el jefe de cocina, otro para el metre y otro para los clientes.

En el **controlador+modelo** hemos introducido todo aquello que conforma la lógica de la aplicación, en definitiva, todos los subsistemas que aparecen en el diagrama de paquetes estructurales.

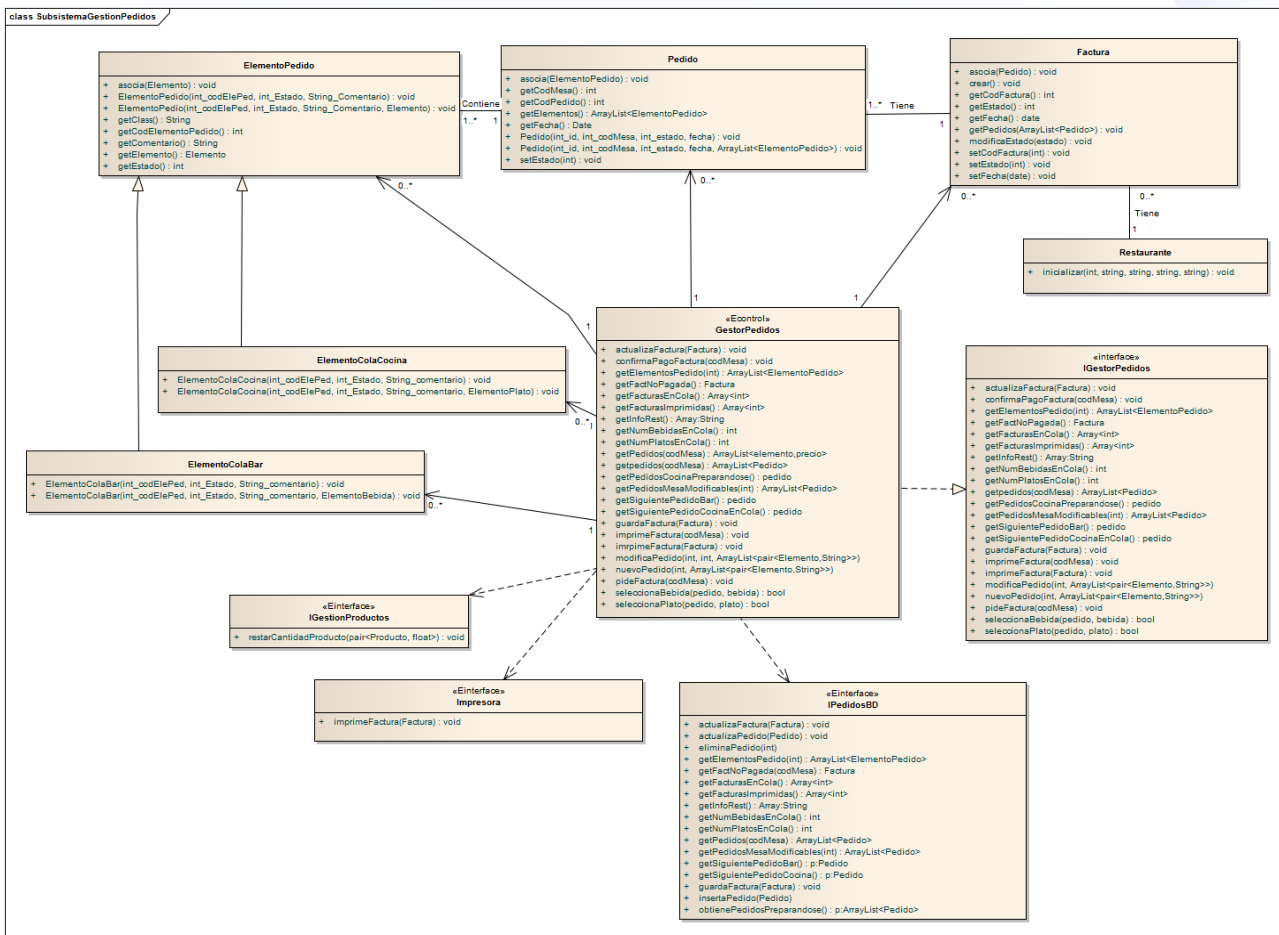
En los **servicios** incluimos todas aquellos subsistemas que escapan de la lógica de aplicación, pero que sin embargo resultan imprescindibles para llevar a cabo las funcionalidades a desarrollar. En nuestro caso necesitaremos un subsistema que permita hacer persistentes todos los cambios en los datos del sistema, y otro que nos permita llevar a cabo las impresiones de las facturas del cliente.

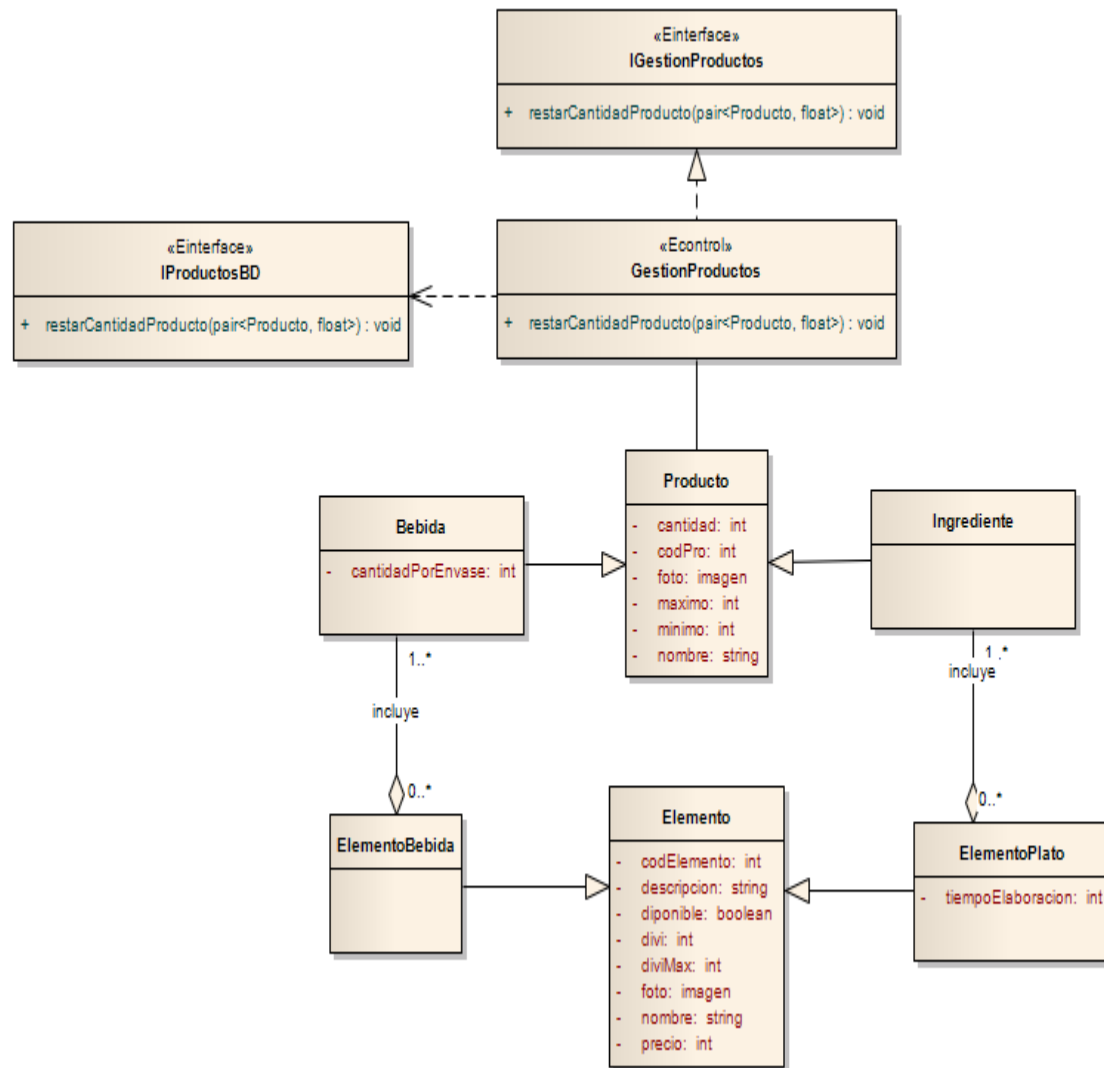
Una vez que tenemos la arquitectura detallada, debemos describir cada uno de los subsistemas subyacentes en cada una de las capas. Detallaremos, en este orden, los subsistemas de la capa controlador+modelo, los subsistemas de la capa de servicios y cada una de las interfaces de usuario pertenecientes a la vista del sistema.

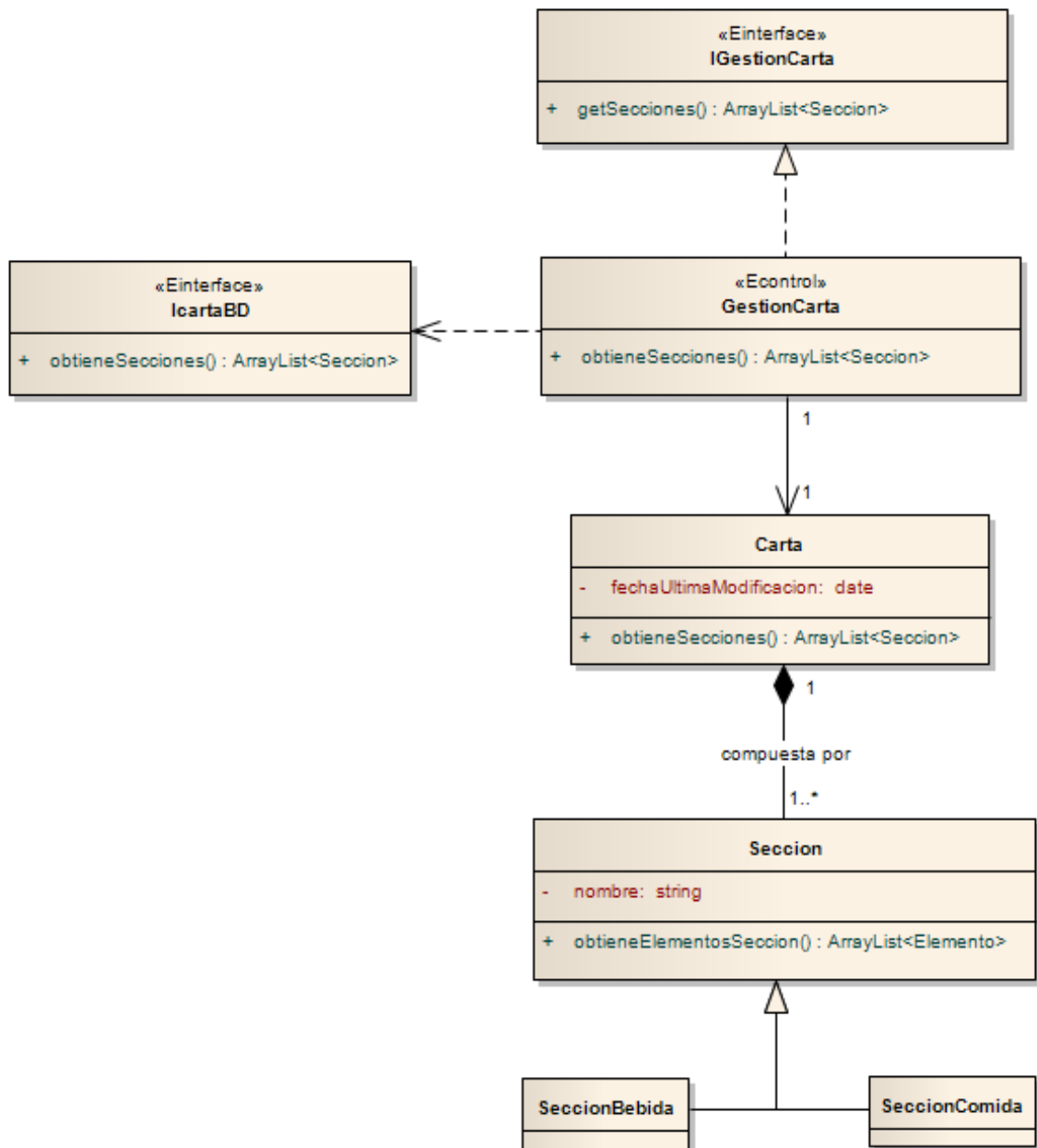


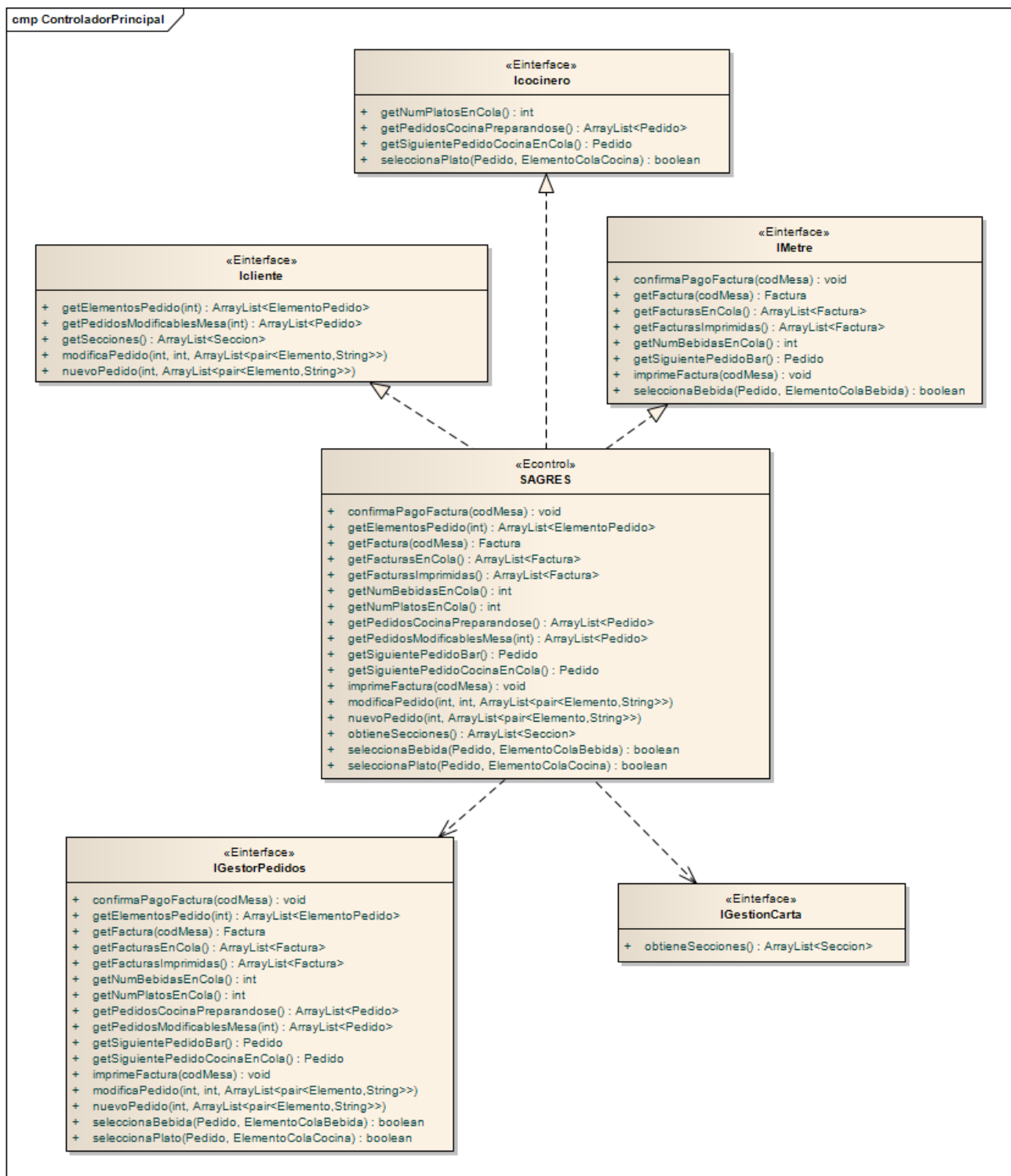
5. IDENTIFICACIÓN DE LAS INTERFACES

A continuación se detallarán las interfaces de los subsistemas que conforman la capa de controlador+modelo. Cada subsistema podrá tener interfaces “ofrecidas” y “requeridas”, cada una de las cuales corresponderán a un conjunto de servicios que ofrecen a otros subsistemas, en el primer caso, o bien a un conjunto de servicios que necesitan de otro subsistema.

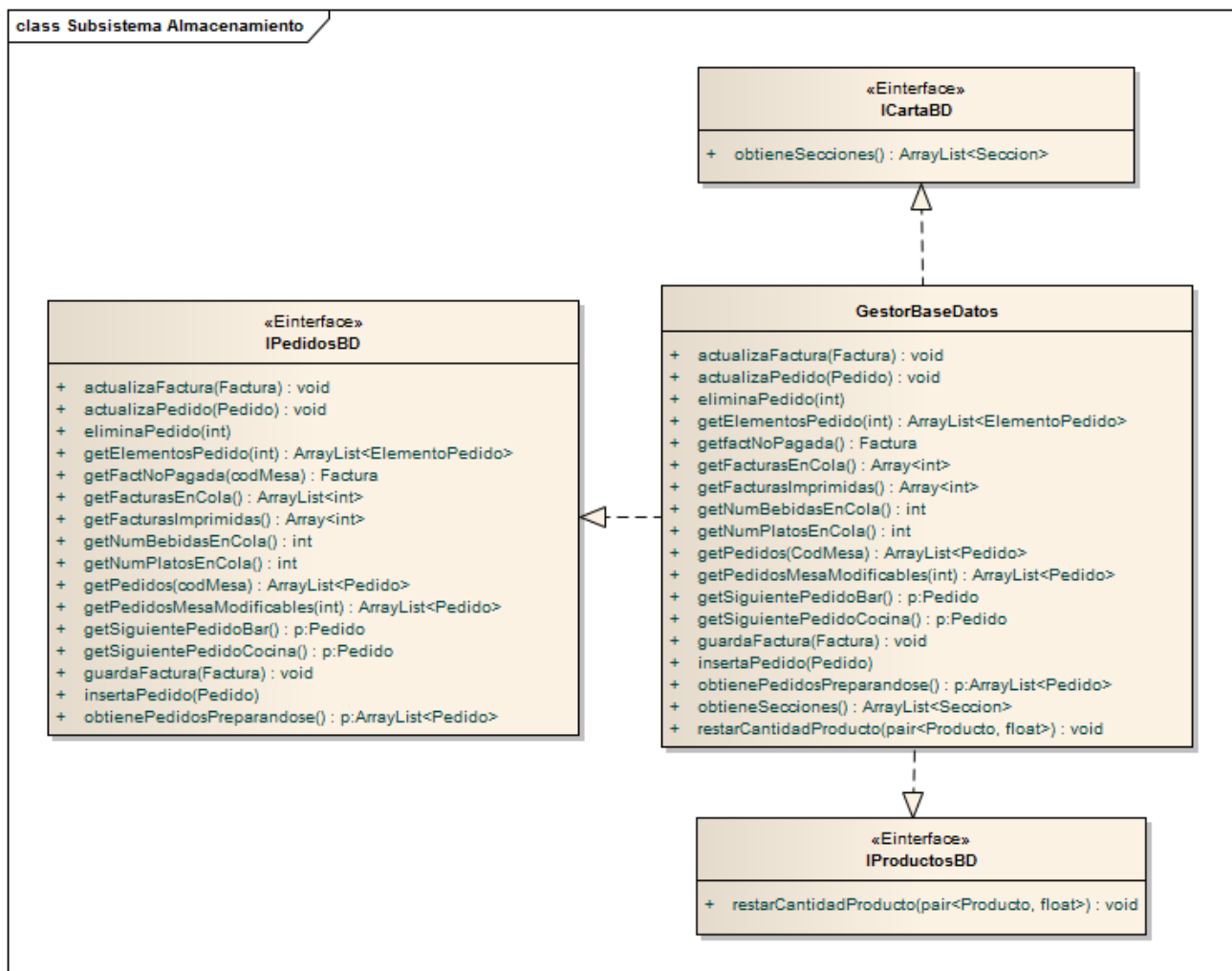




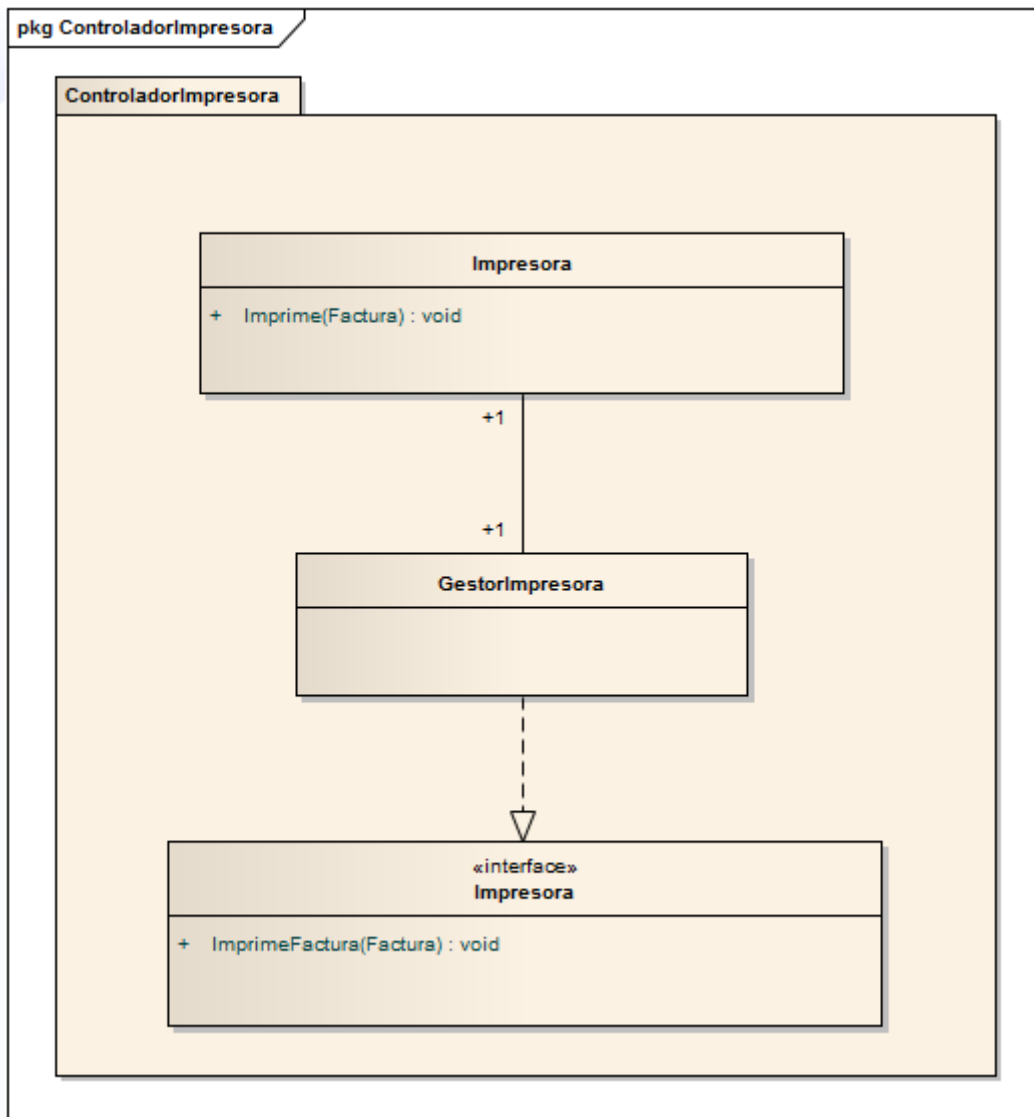




Se puede comprobar que han aparecido nuevas operaciones en SAGRES, como son *getNumPlatosEnCola()* y *getNumBebidasEnCola()*. Estas operaciones son necesarias para la vista para conocer el número de platos y bebidas que le queda al cocinero o metre por servir.



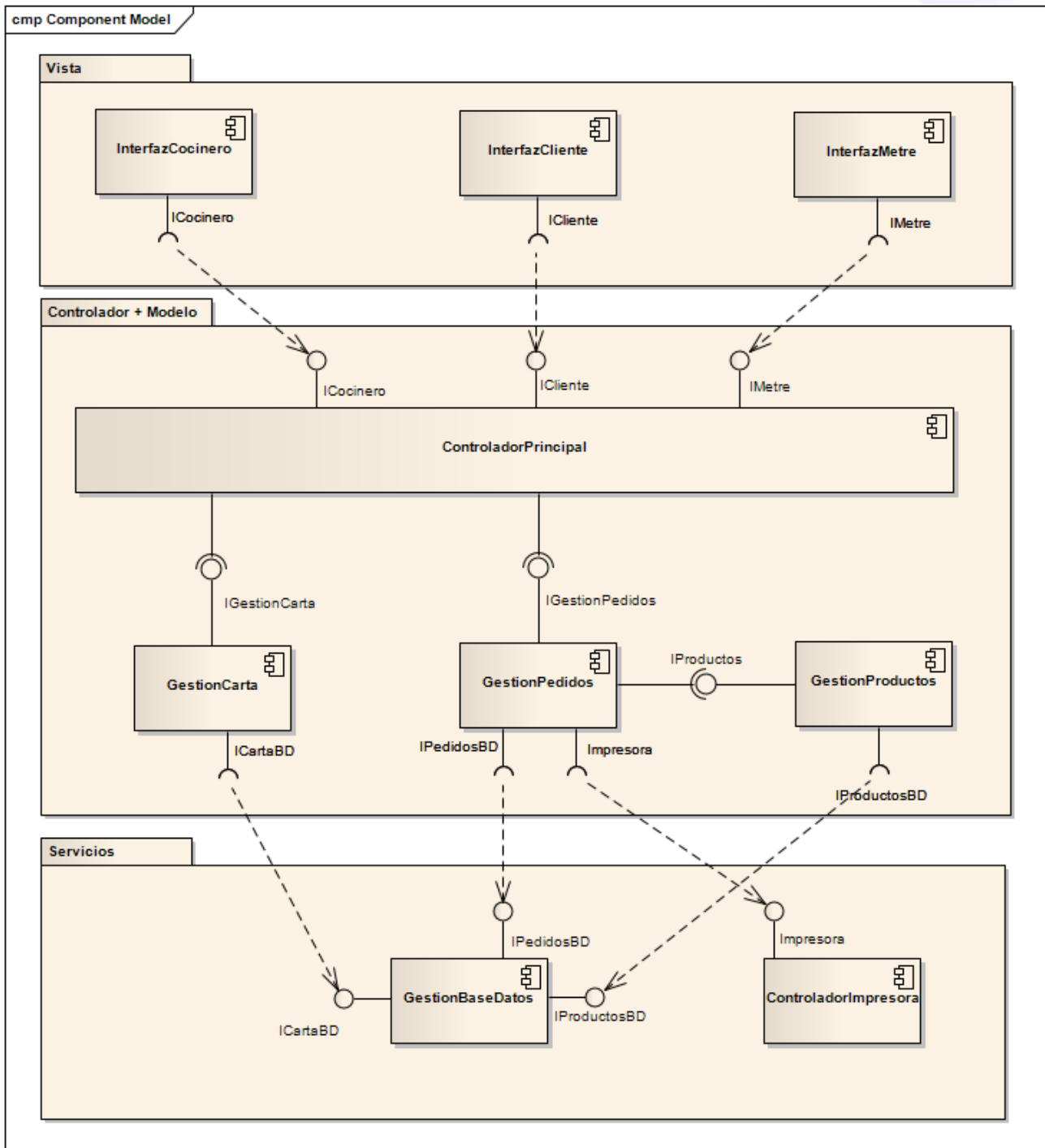
En esta iteración hacemos uso de la operación `restarCantidadProducto` en la interfaz `IproductosBD`, realizada en la iteración anterior.



La función `imprime` recibe como parámetros la lista de elementos con sus respectivos precios, el total y un array de string que contiene la información del restaurante.

6. DIAGRAMA DE COMPONENTES

Una vez definidas las interfaces de los subsistemas, podremos definir a continuación el diagrama de componentes del sistema:



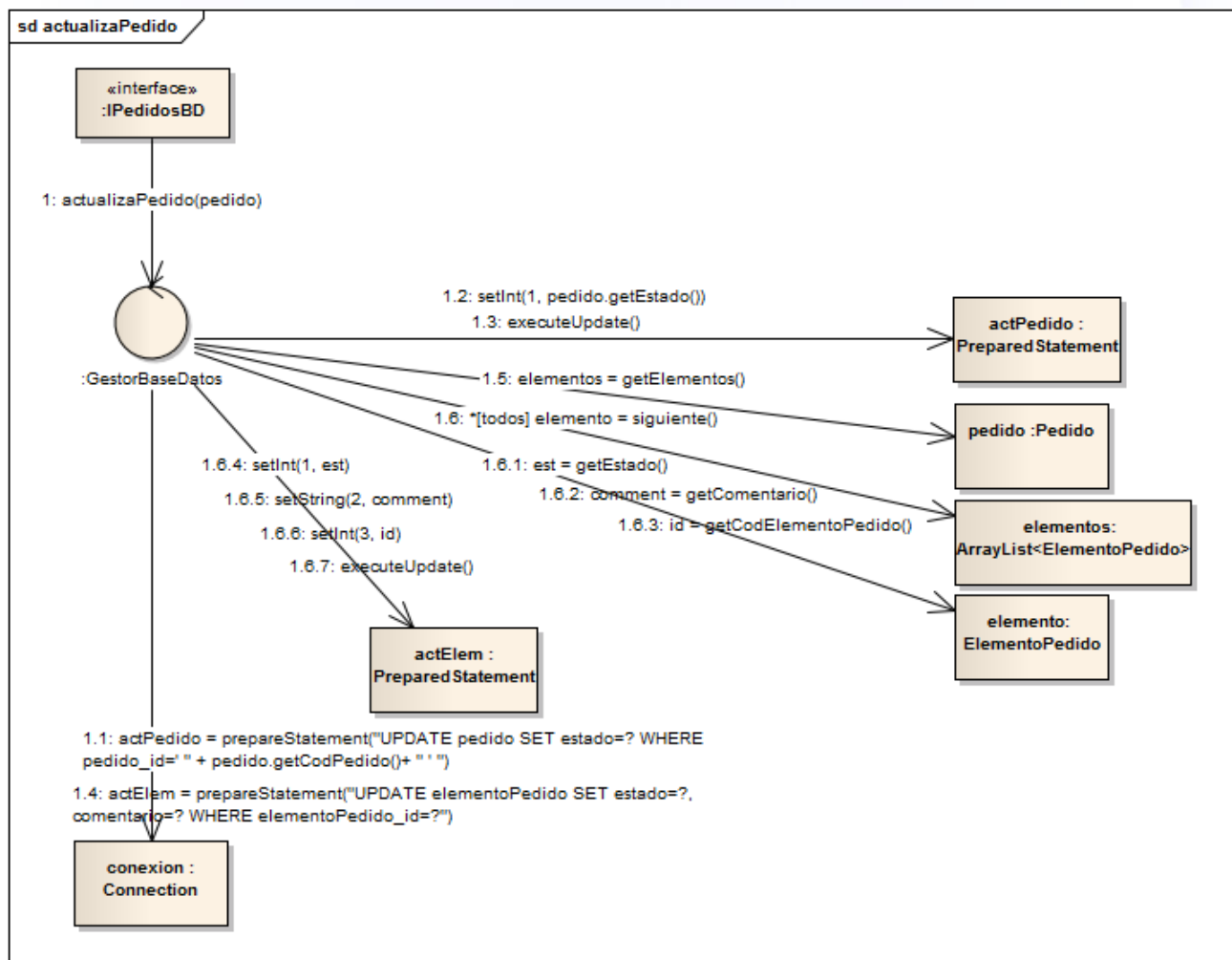
7. DIAGRAMAS DE COLABORACIÓN

Funciones de base de datos

A continuación se especifican algunas funciones que son necesarias para la obtención de elementos de la base de datos. Se definen a parte porque son utilizadas en varios diagramas de colaboración.

actualizaPedido

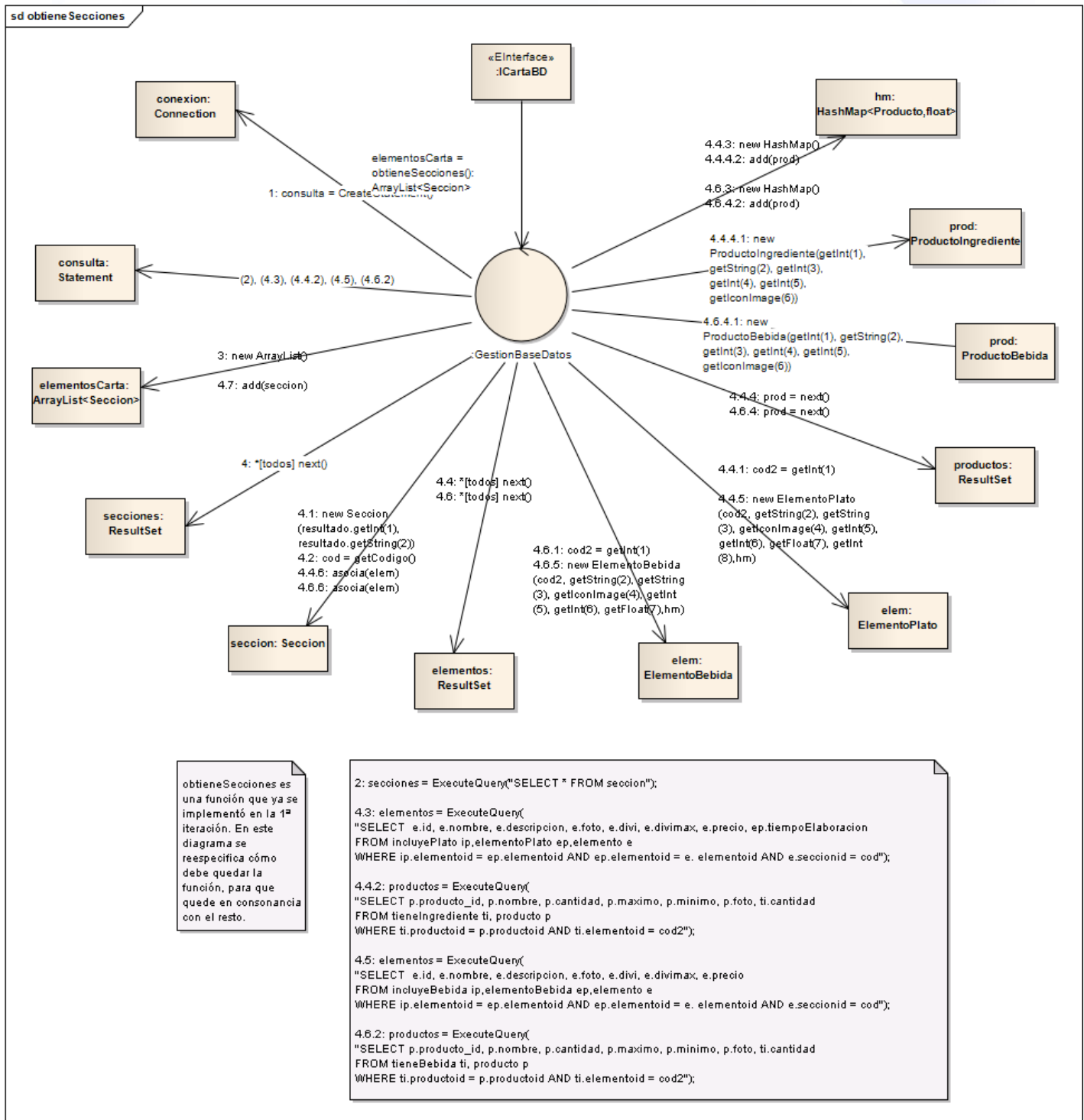
Actualiza los atributos de un pedido así como los estados de sus ElementoPedido.



Realizado por Sergio Rodríguez Lumley

obtieneSecciones

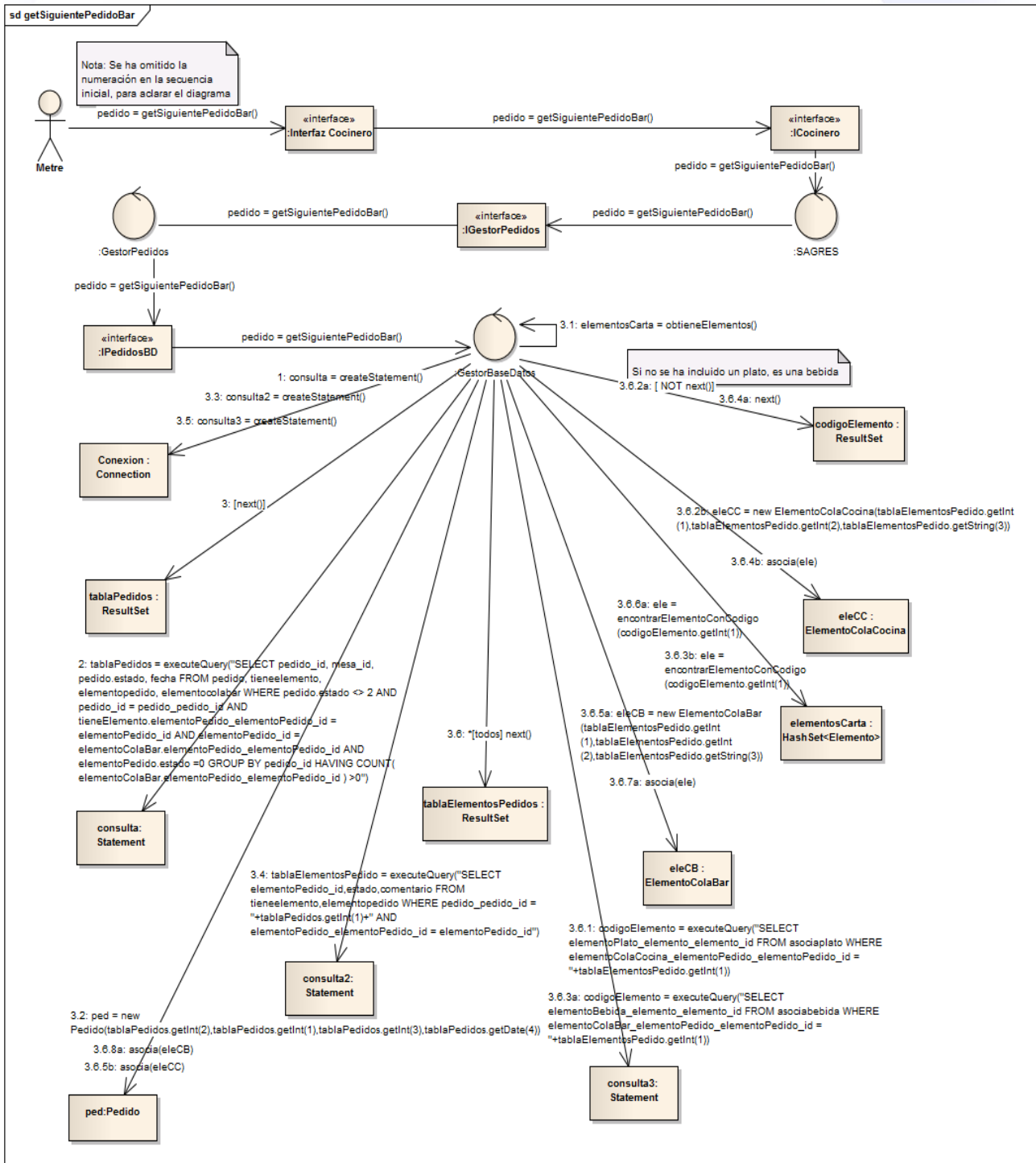
Obtiene las secciones de la carta desde la base de datos.



Realizado por Adrián Víctor Pérez Lopera.

getSiguientePedidoBar

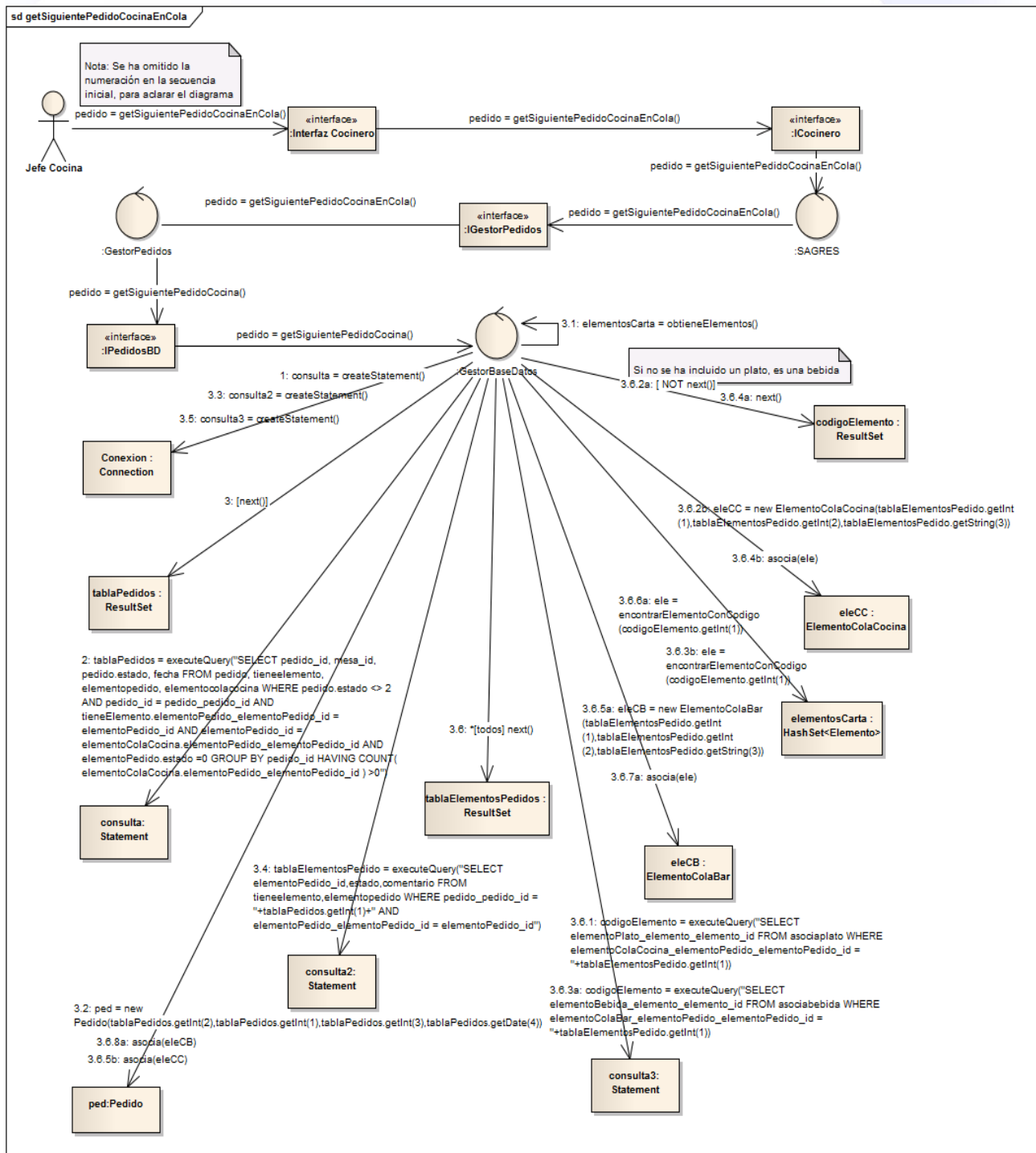
Obtiene el pedido con una **bebida** en cola más antiguo.



Realizado por Sergio Rodríguez Lumley

getSiguientePedidoCocinaEnCola

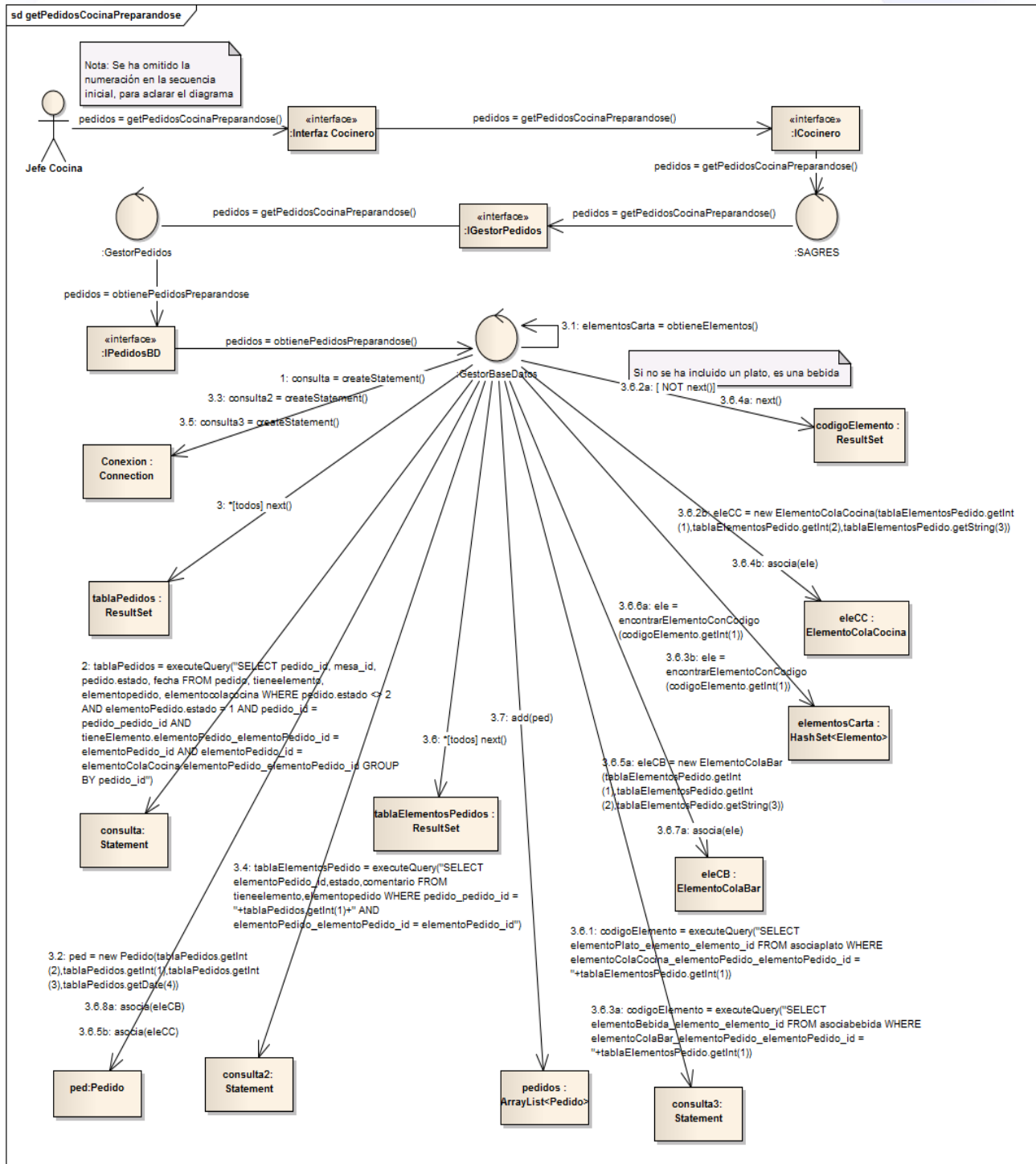
Obtiene el pedido con un **plato** en cola más antiguo.



Realizado por Sergio Rodríguez Lumley

getPedidosCocinaPreparandose

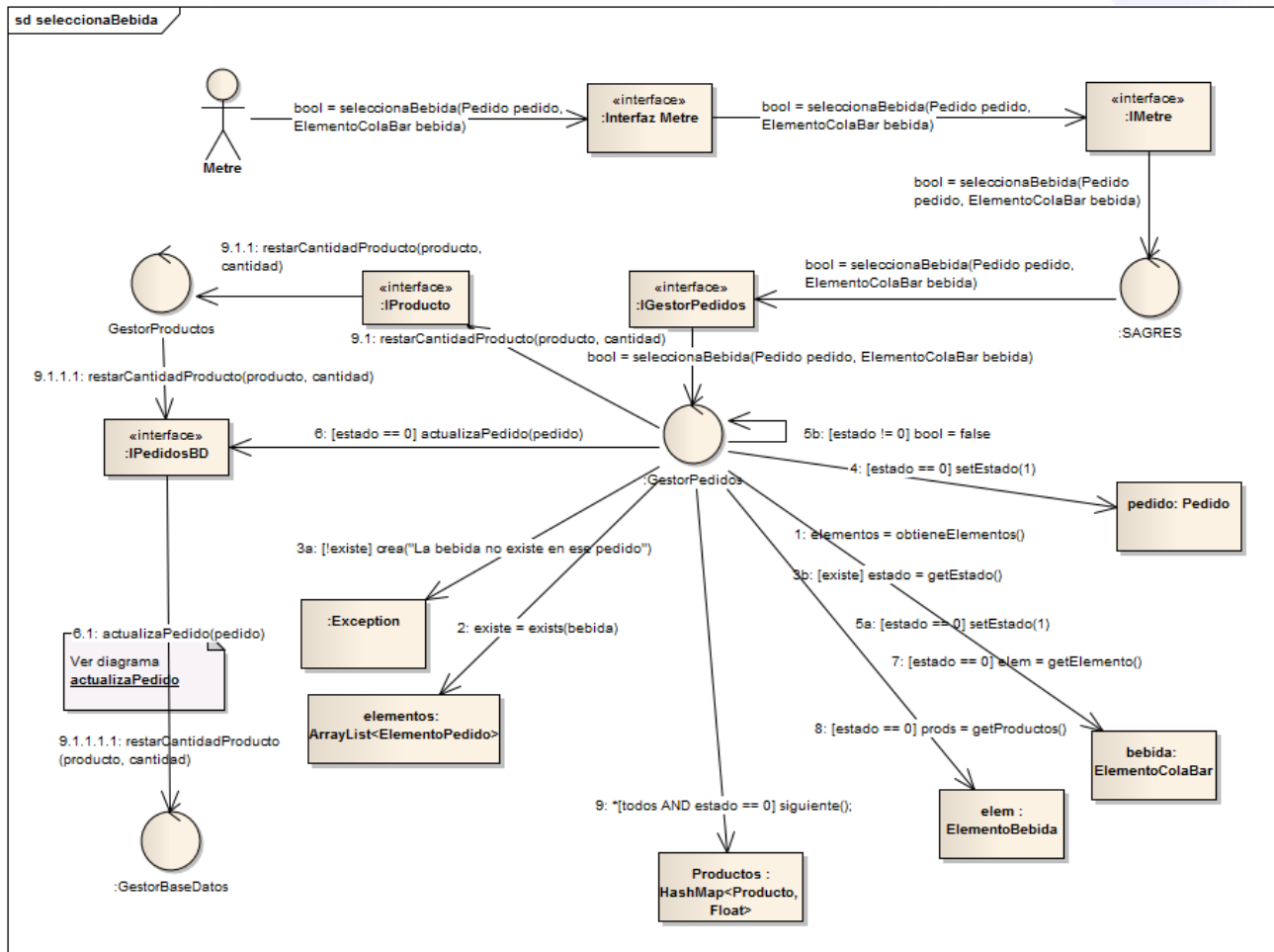
Obtiene todos los pedidos que tengan un **plato** preparándose.



Realizado por Sergio Rodríguez Lumley

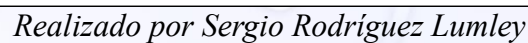
seleccionaBebida

Cambia el estado de una bebida y, si es necesario, reduce la cantidad de bebida que consume de la base de datos.



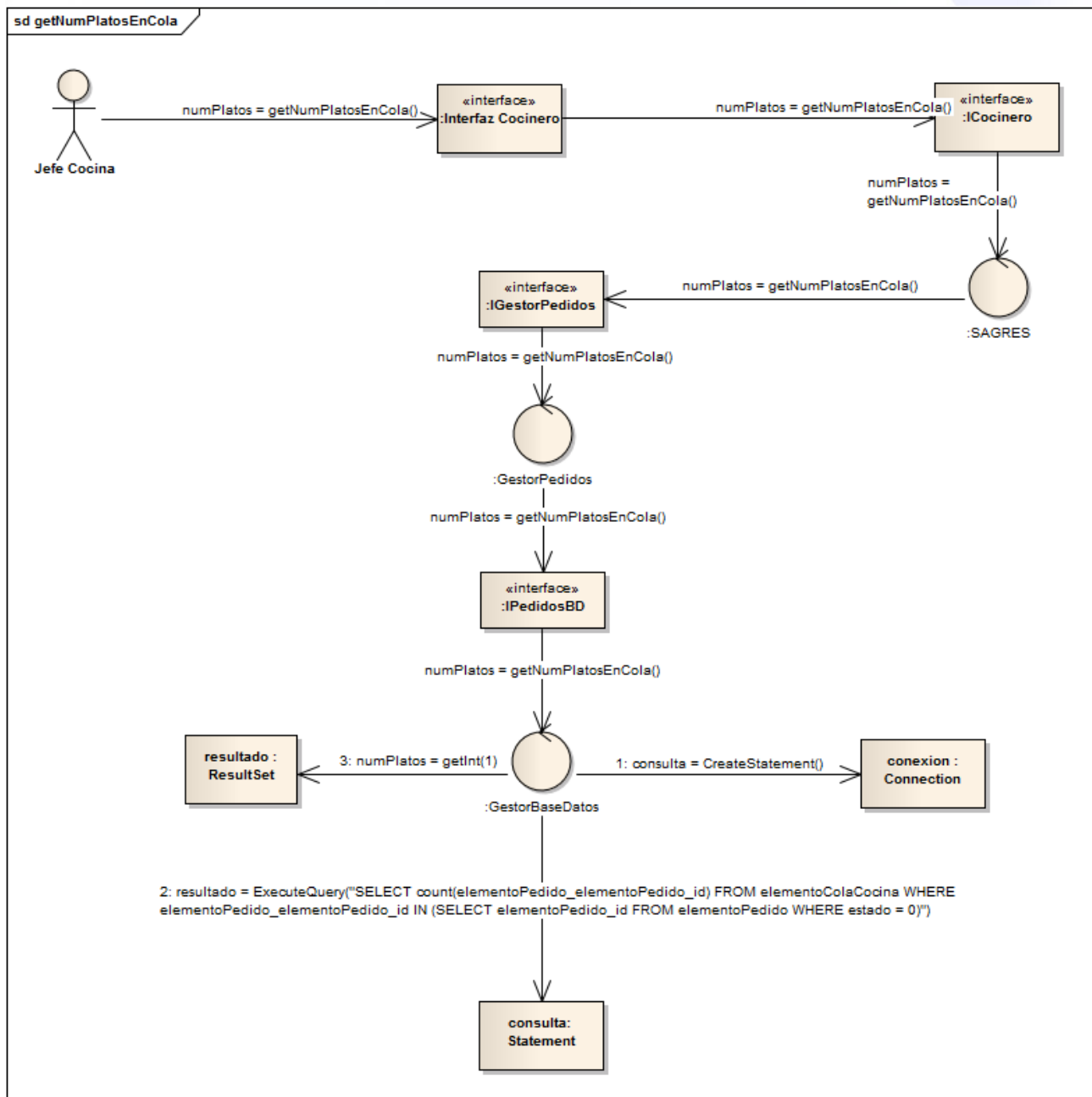
Realizado por Sergio Rodríguez Lumley

Cambia el estado de un plato y, si es necesario, reduce la cantidad de ingredientes que consume de la base de datos.



getNumPlatosEnCola

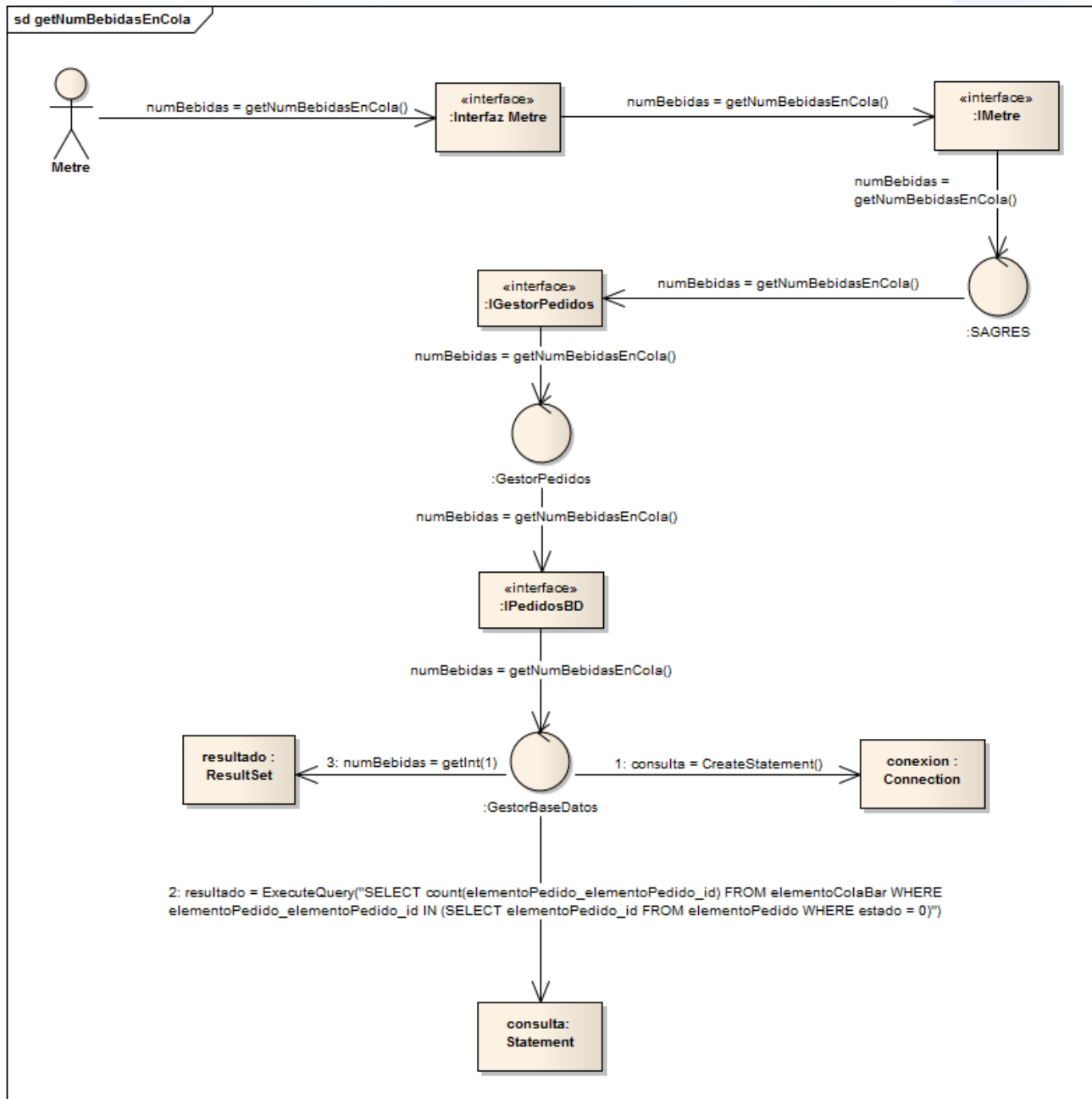
Obtiene el número de platos “En cola” que quedan en cola para informar al jefe de cocina. Esta función tiene uso únicamente informativo para la vista.



Realizado por Sergio Rodríguez Lumley

getNumBebidasEnCola

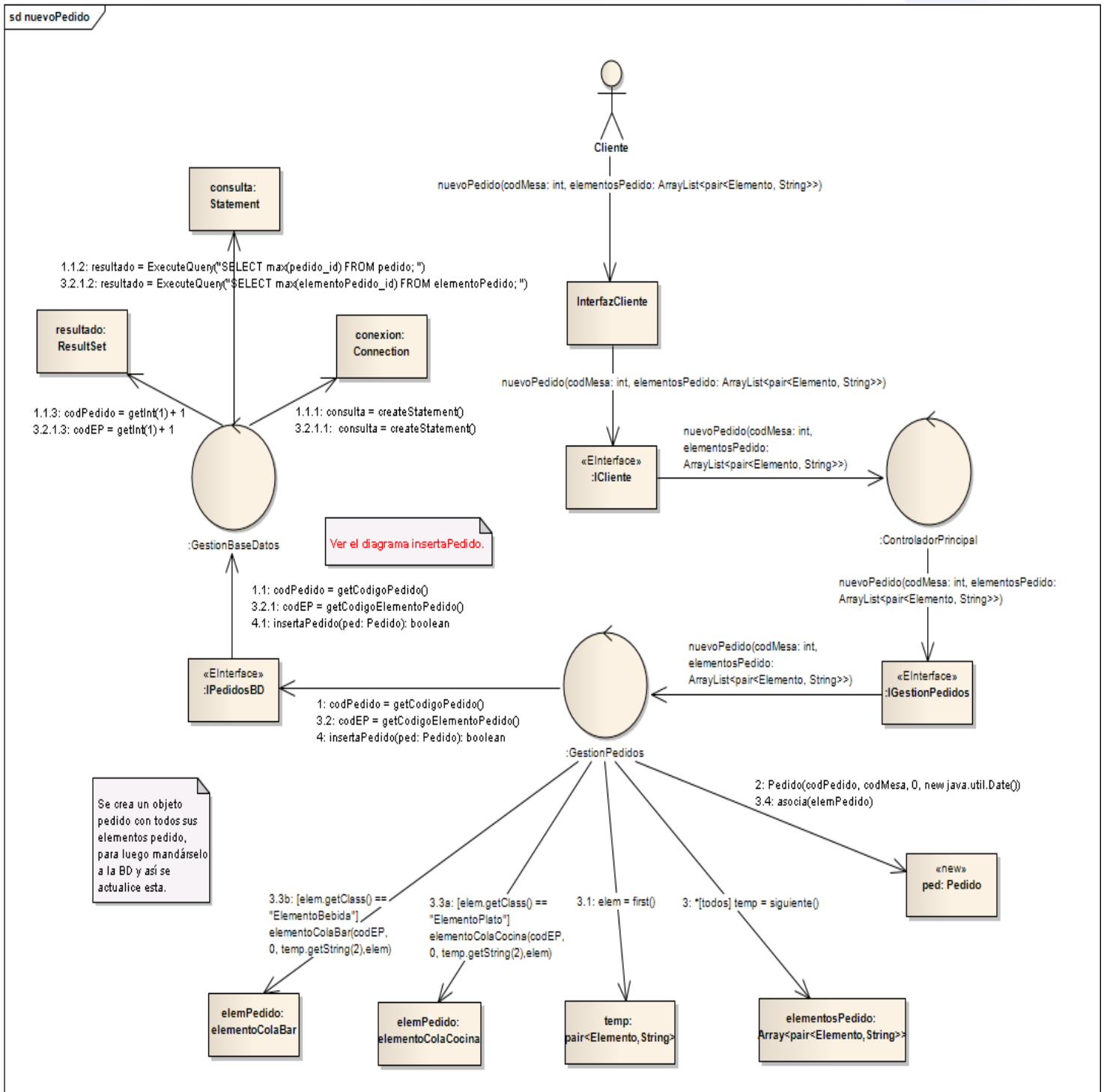
Obtiene el número de bebidas que quedan en cola para informar al metre. Esta función tiene uso únicamente informativo para la vista.



Realizado por Sergio Rodríguez Lumley

nuevoPedido

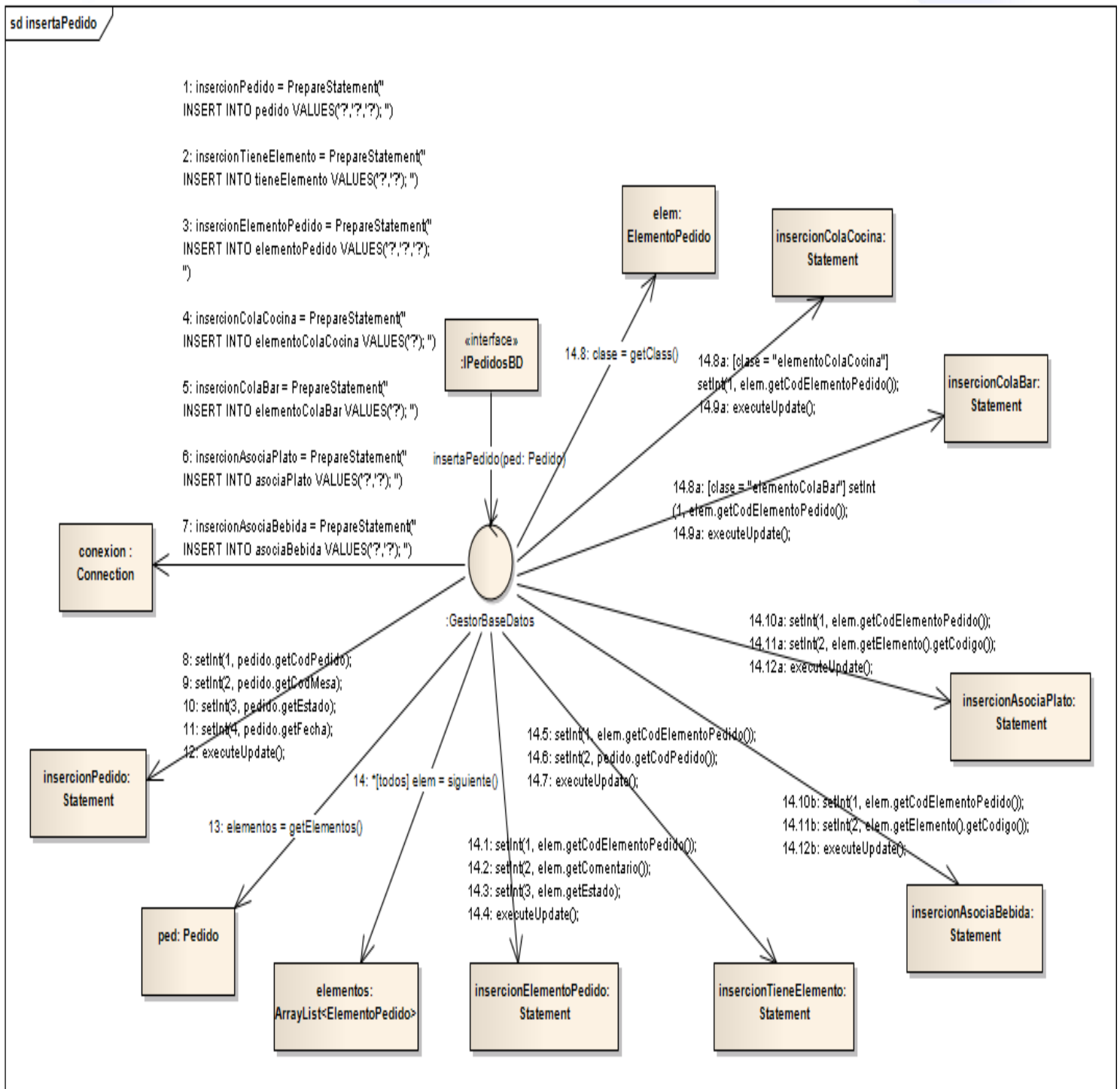
Incluye un nuevo pedido en el sistema.



.Realizado por Adrián Víctor Pérez Lopera.

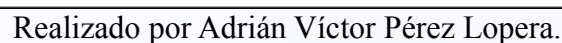
InsertaPedido

Almacena los datos de un pedido en la base de datos.



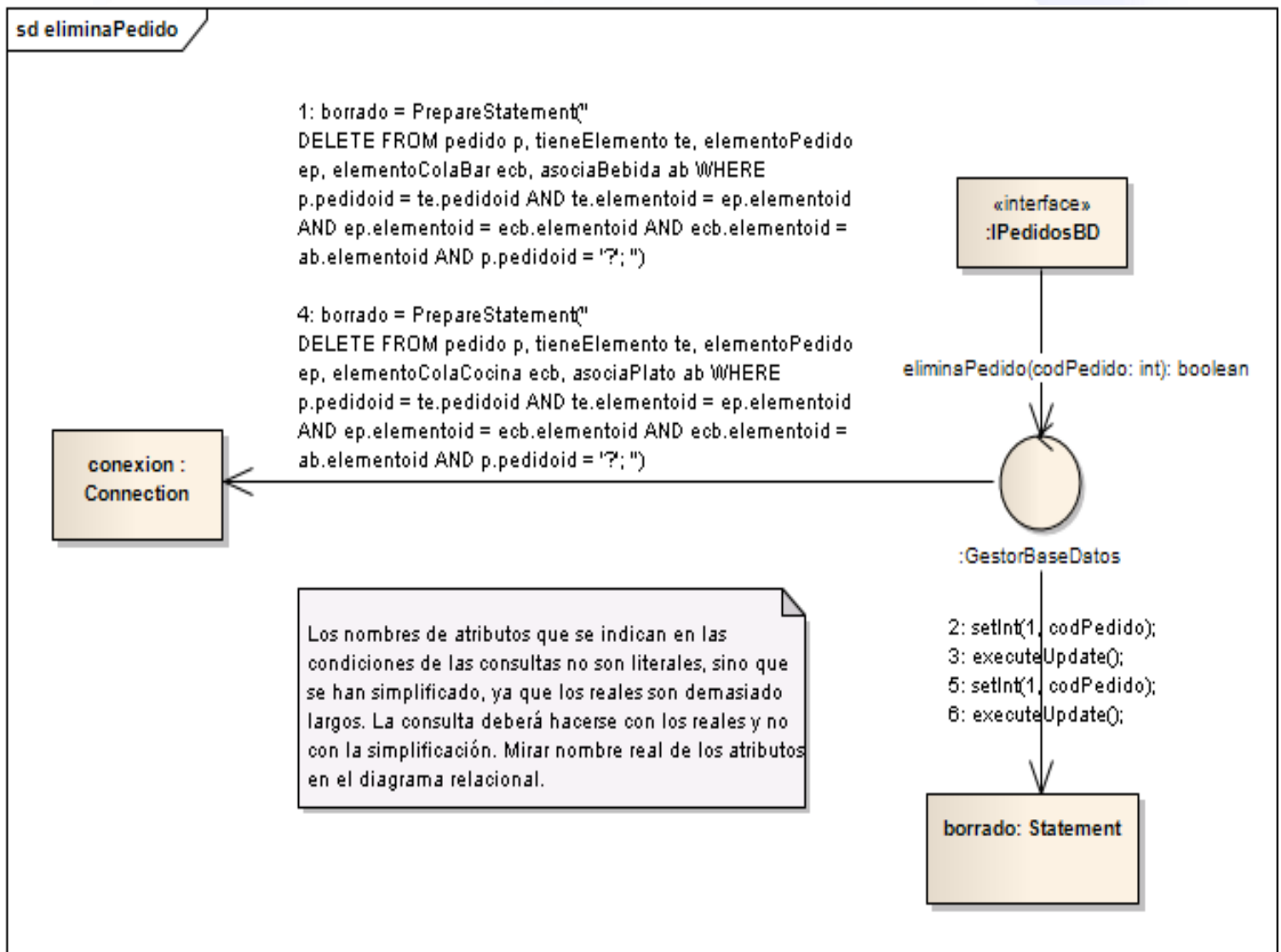
Realizado por Adrián Víctor Pérez Lopera.

Modifica los elementos de un pedido del sistema.



eliminaPedido

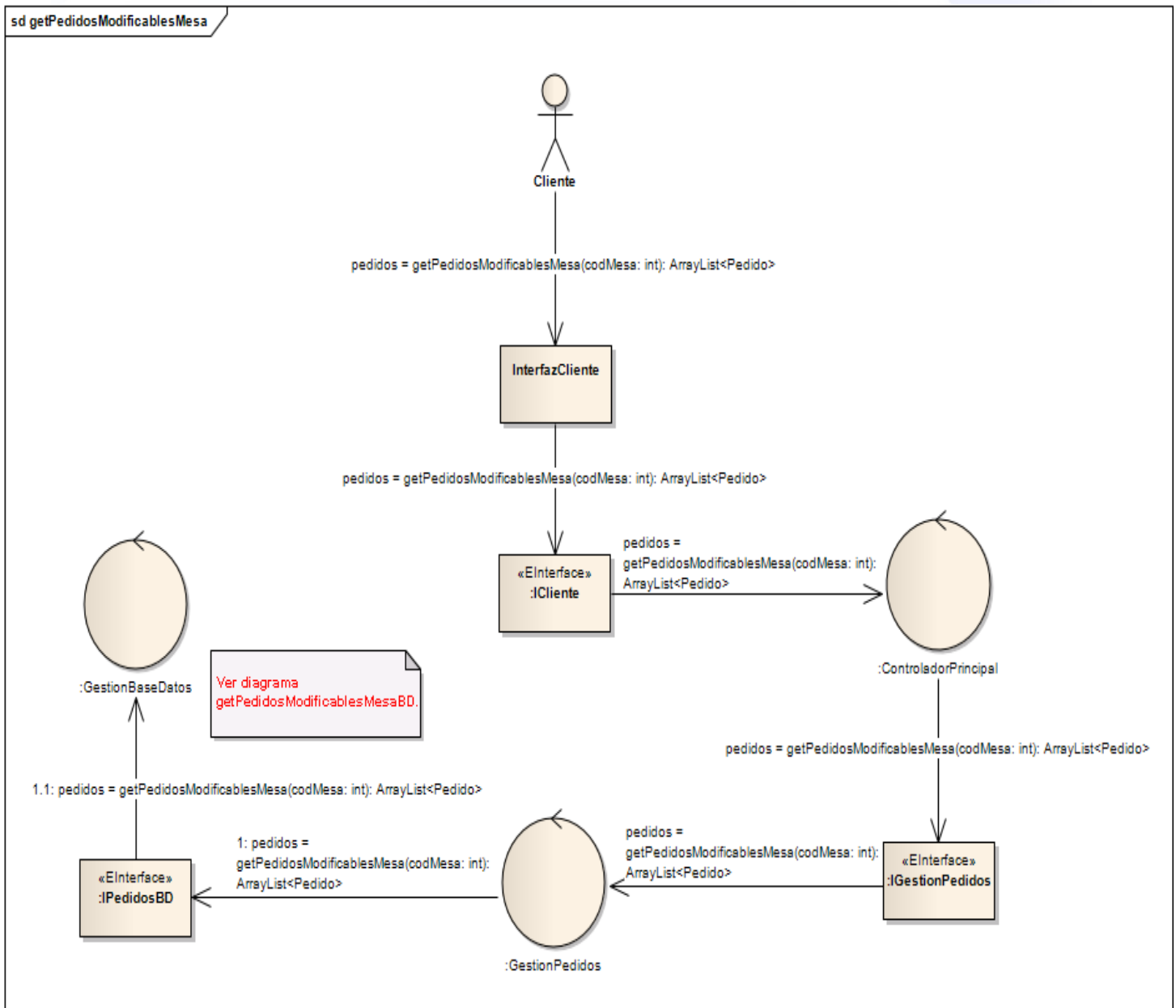
Elimina un pedido del sistema.



Realizado por Adrián Víctor Pérez Lopera.

getPedidosMesaModificables

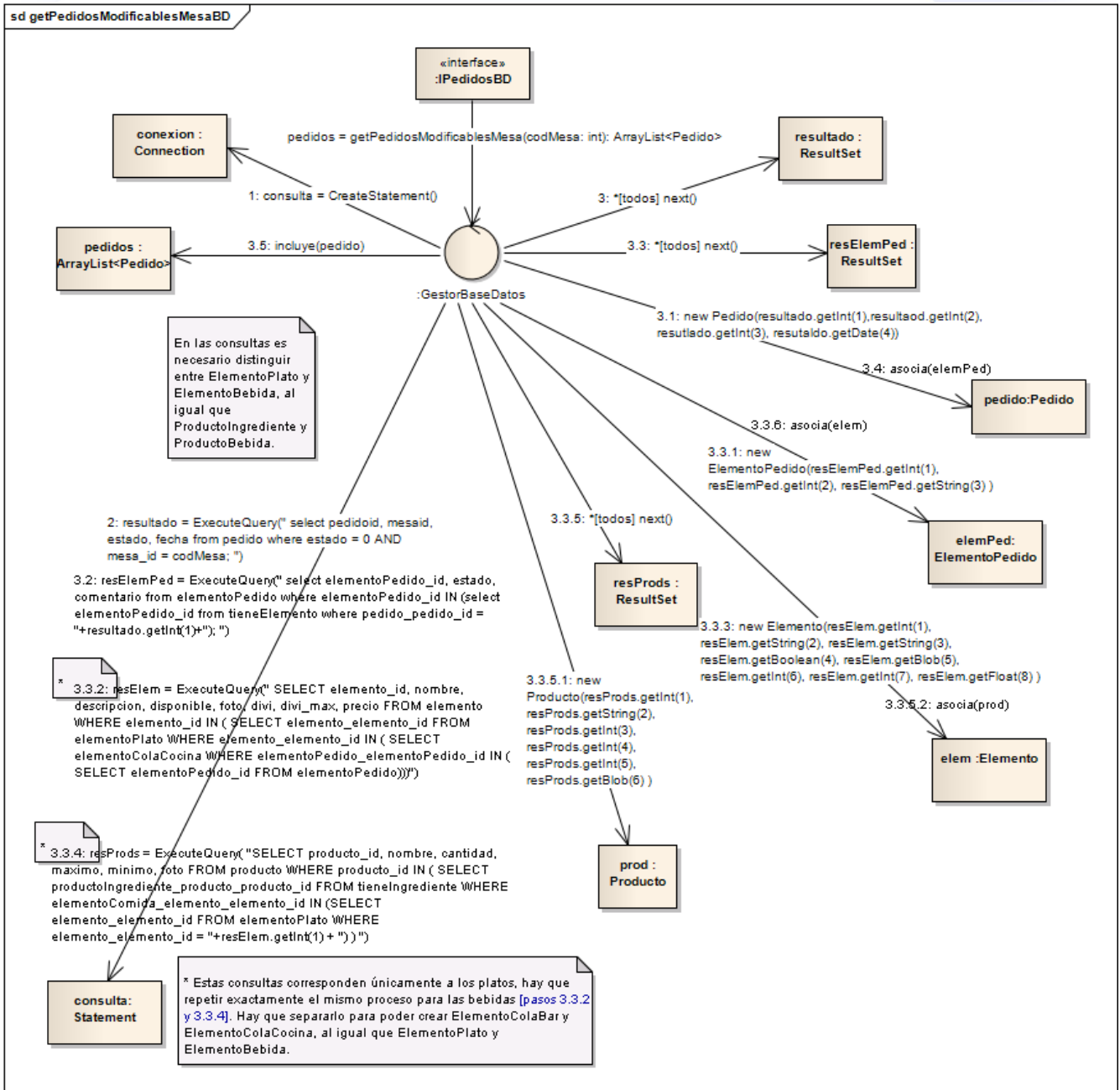
Obtiene los pedidos de una mesa que todavía se pueden modificar.



Realizado por Adrián Víctor Pérez Lopera.

getPedidosMesaModificablesBD

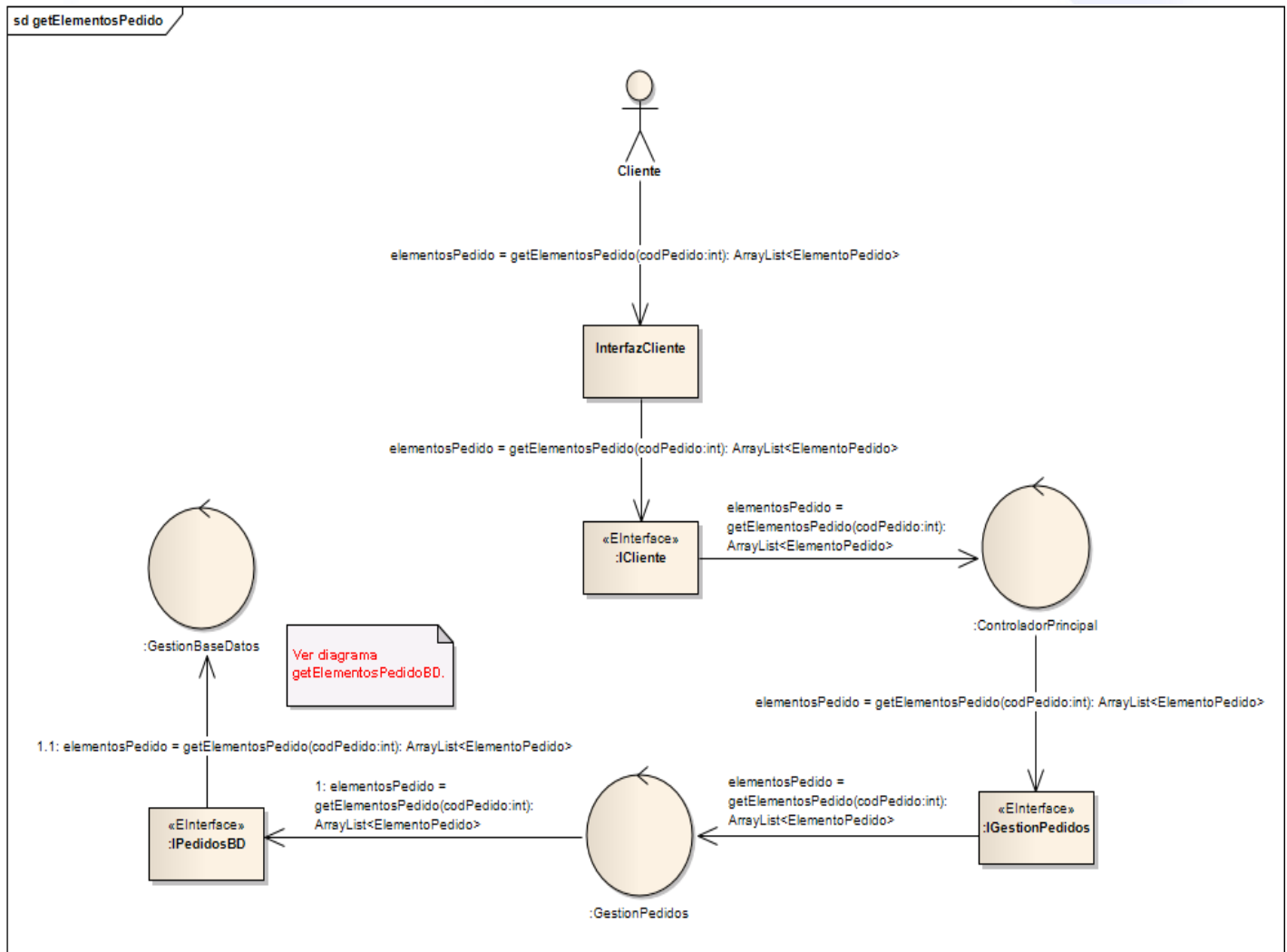
Obtiene los pedidos de una mesa que todavía se pueden modificar de la base de datos.



Realizado por Adrián Víctor Pérez Lopera.

getElementosPedido

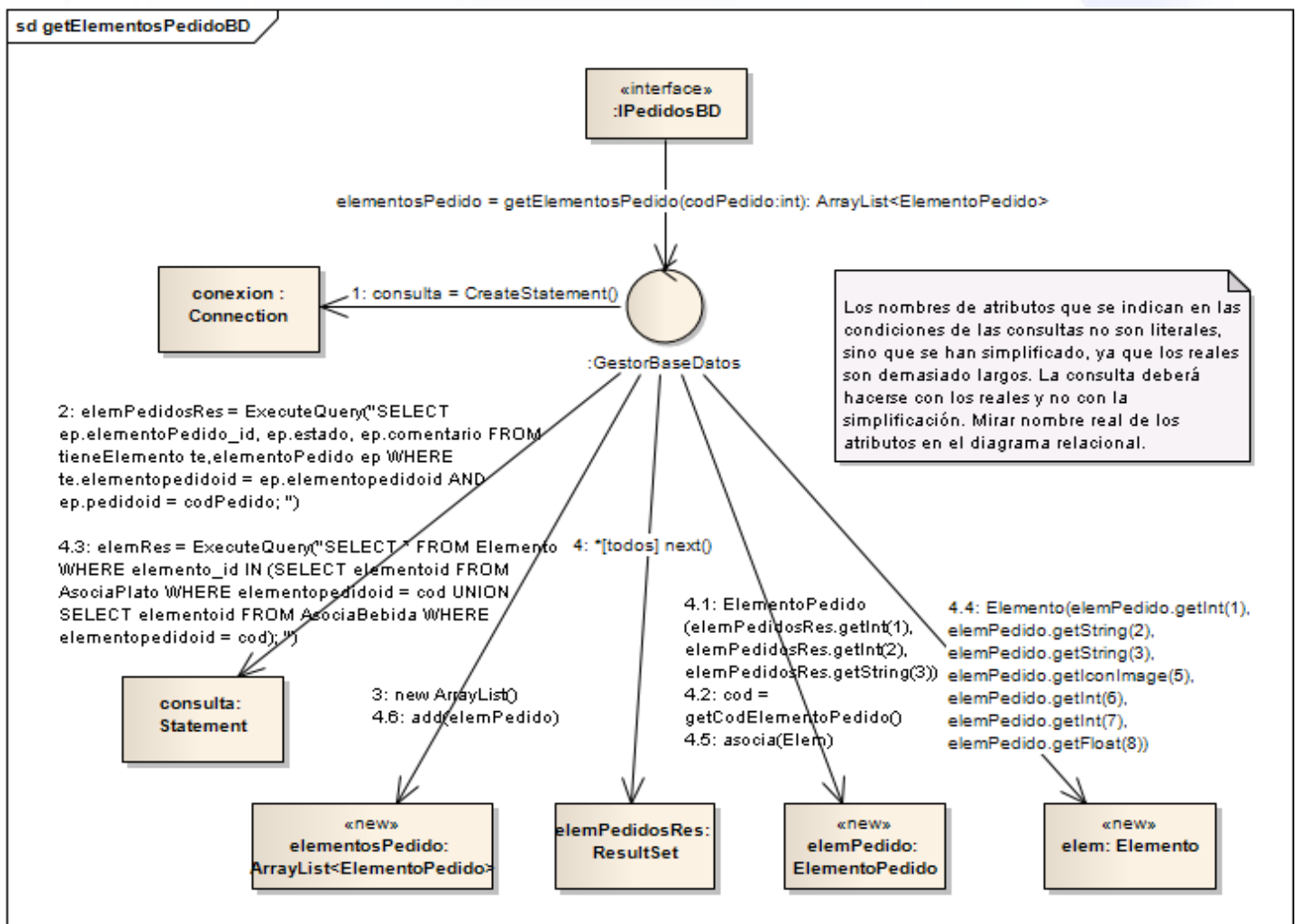
Obtiene los elementos de un pedido.



Realizado por Adrián Víctor Pérez Lopera.

getElementosPedido

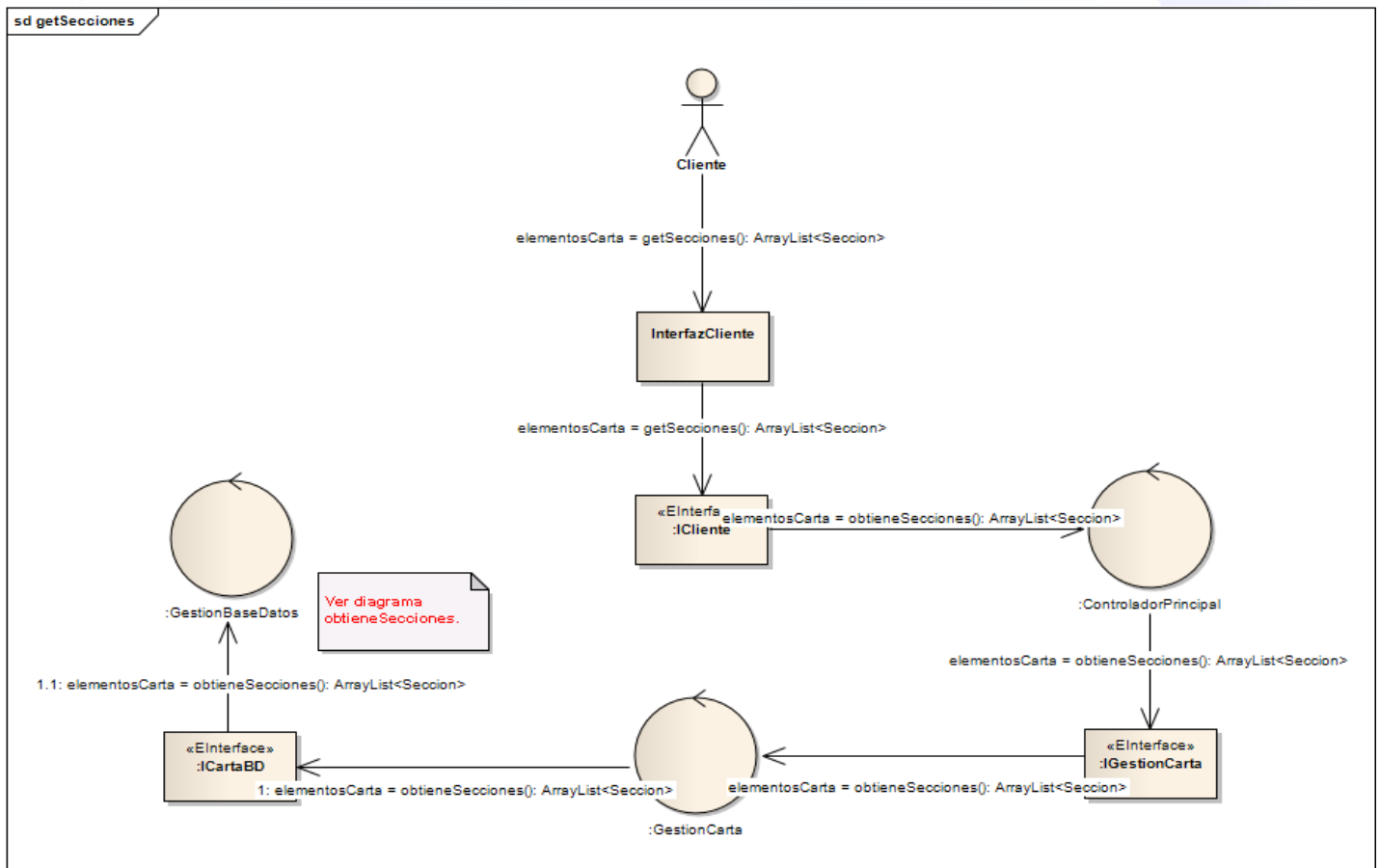
Obtiene los elementos de un pedido de la base de datos.



Realizado por Adrián Víctor Pérez Lopera.

getSecciones

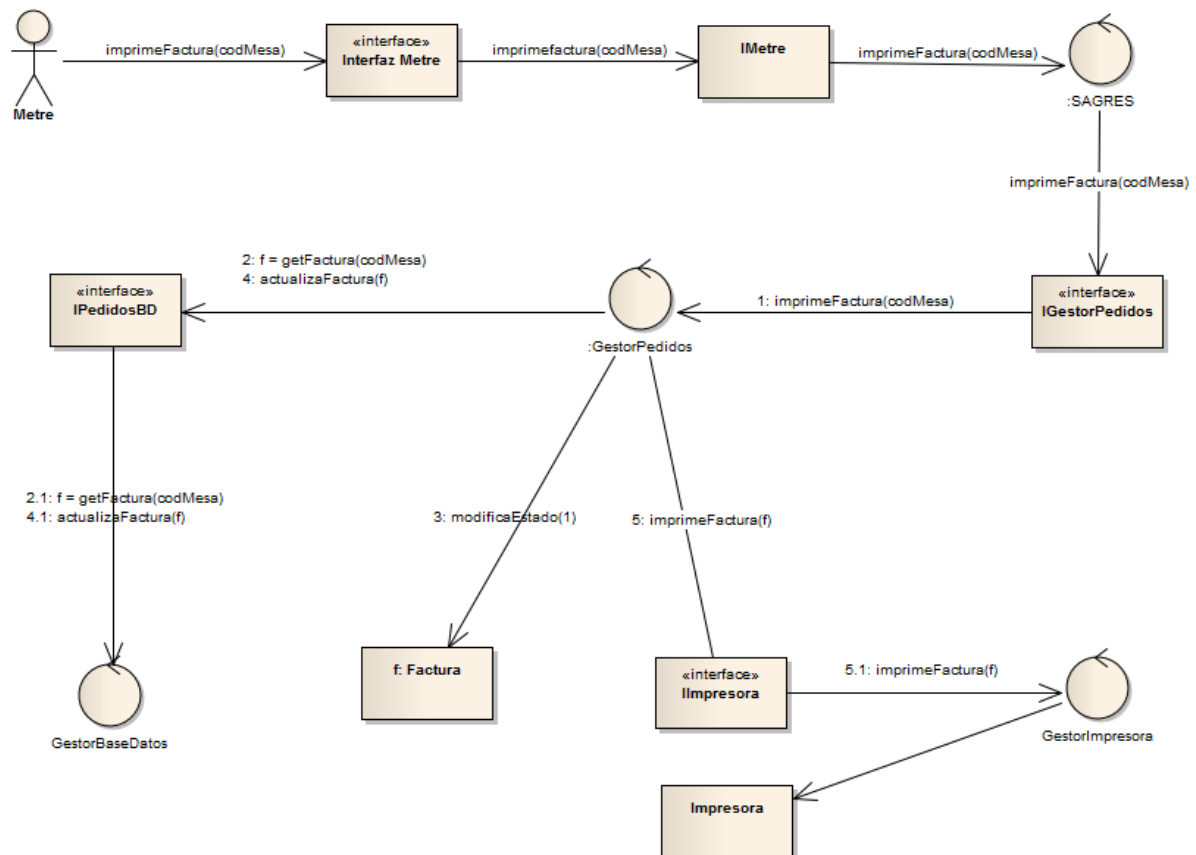
Obtiene las secciones de la carta.



Realizado por Adrián Víctor Pérez Lopera.

imprimeFactura

sd imprime Factura



Consultas SQL:

getFactura(codMesa):

1. SELECT factura_id,estado,fecha FROM factura,facturaPedido,pedido WHERE factura_id = factura_factura_id AND pedido_pedido_id = pedido_id AND mesa_id = "+codMesa+" AND estado = 1"; **devuelve la factura sin los pedidos.**

2. "SELECT * FROM pedido WHERE mesa_id = "+codMesa+" AND estado = 1"; >>**devuelve los pedidos en estado 1 de la mesa codMesa.**

2.1. mientras_hay_pedidos:

2.1.1. "SELECT elementoPedido_id,estado,comentario FROM tieneElemento,elementoPedido WHERE pedido_pedido_id = "+tablaPedidos.getInt(1)+" AND elementoPedido_elementoPedido_id = elementoPedido_id"; **devuelve los elementosPedido asociados al pedido.**

2.1.2. mientras_hay_elementos_asociados:

2.1.2.1.a es_una_bebida: "SELECT elementoBebida_elemento_elemento_id FROM asociaBebida WHERE elementoColaBar_elementoPedido_elementoPedido_id = codElemento.

2.1.2.1.b es_un_plato: "SELECT elementoPlato_elemento_elemento_id FROM asociaPlato WHERE elementoColaCocina_elementoPedido_elementoPedido_id = codElemento.

2.1.2.2: asociar elemento al pedido.

2.1.3: asociar pedido a la factura.

actualizaFactura(f):

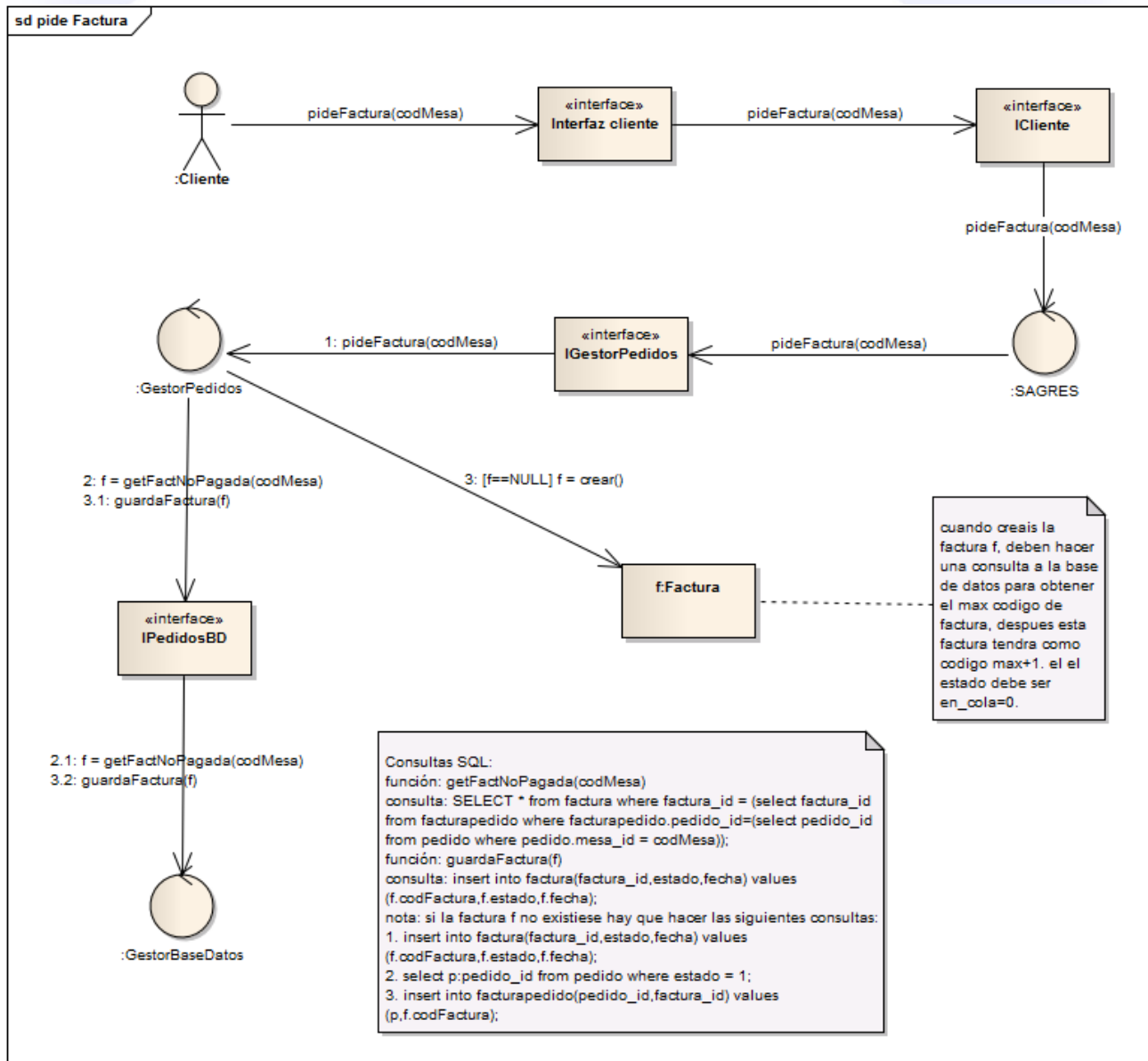
1."UPDATE factura SET estado=1 WHERE factura_id=f.getCodFactura()";

actualizaPedido(ped):

1."UPDATE pedido SET estado=? WHERE pedido_id=ped.getCodPedido()";

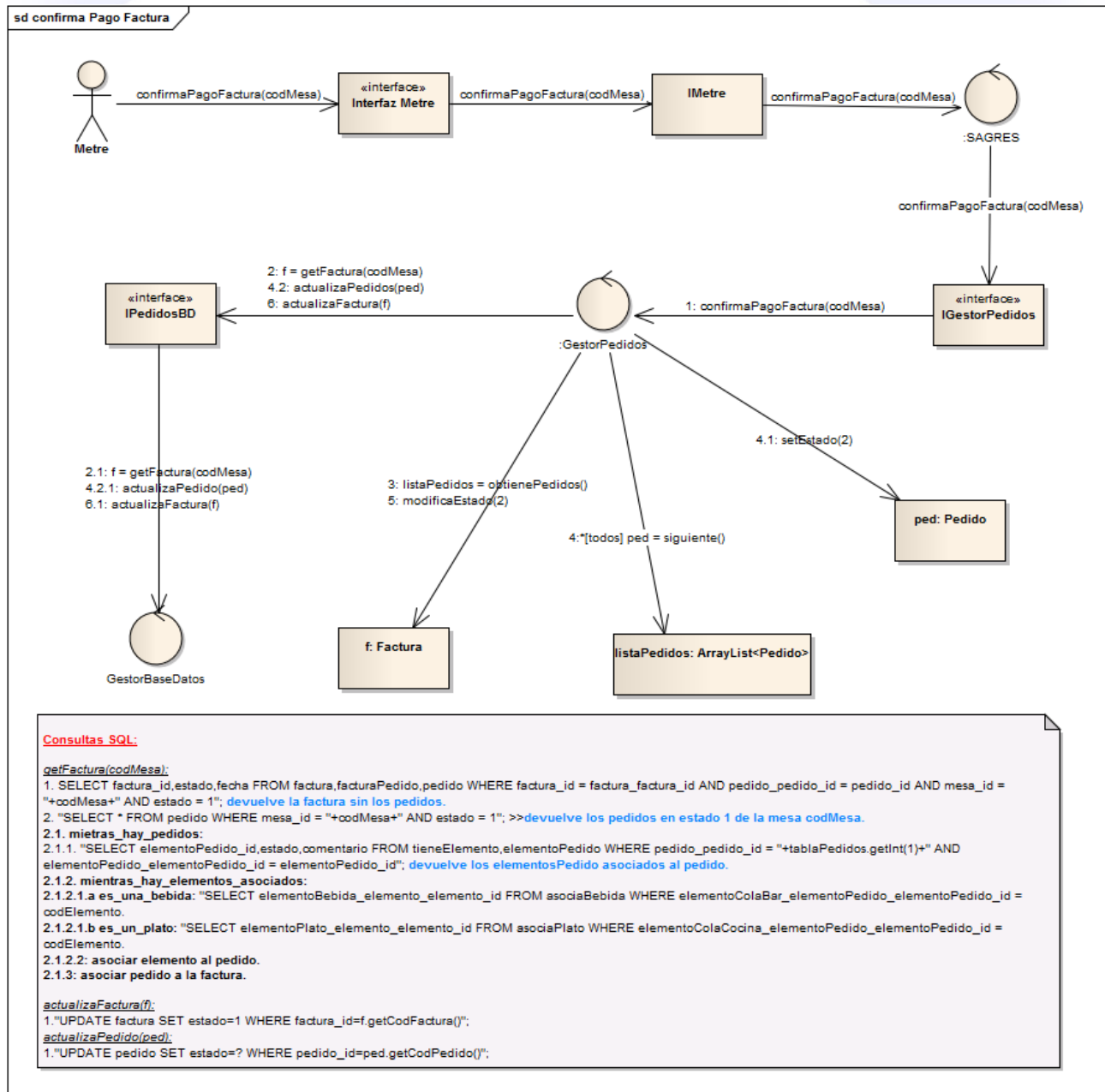
Realizado por Nabil

pideFactura



Realizado por Nabil

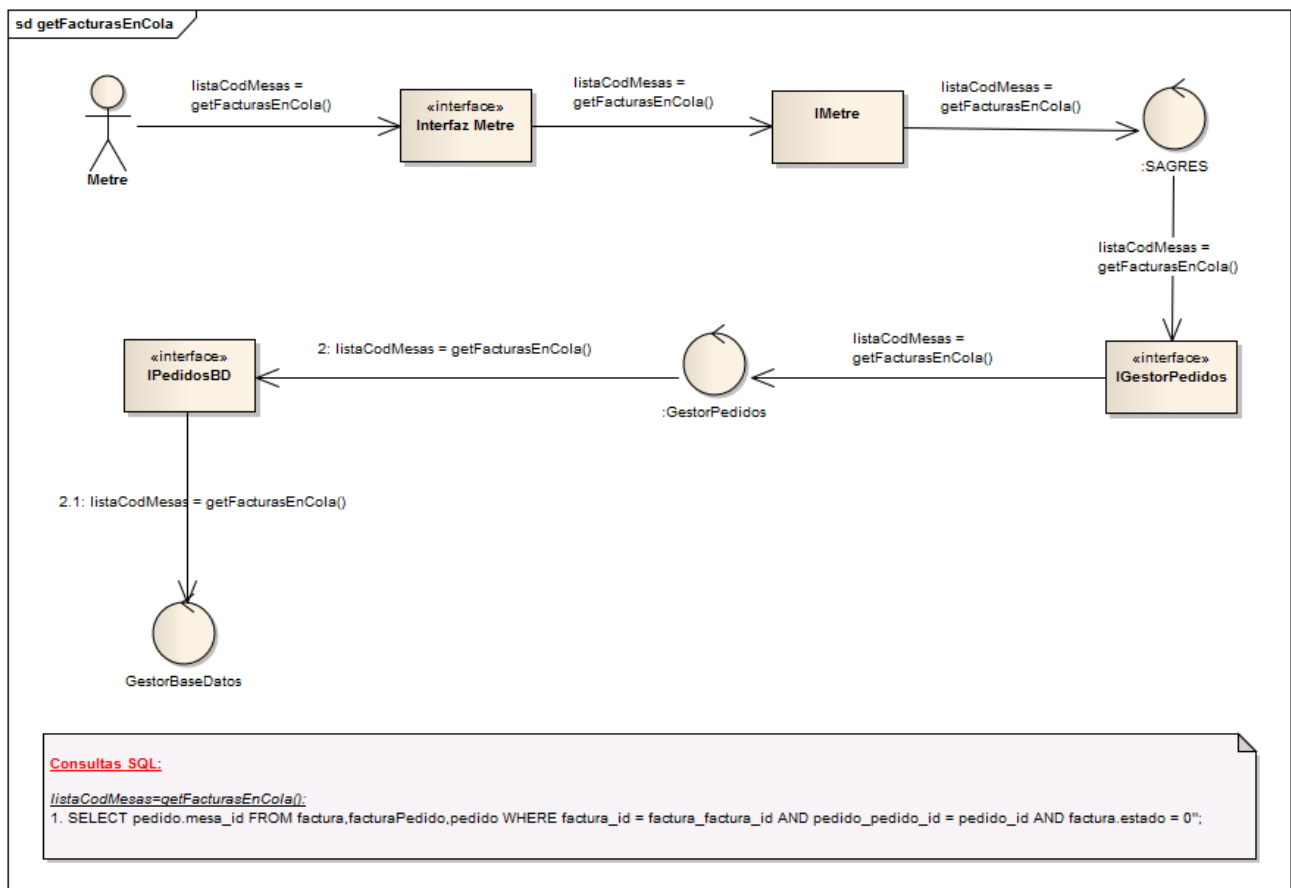
confirmaPagoFactura



Realizado por Nabil

getFacturasEnCola

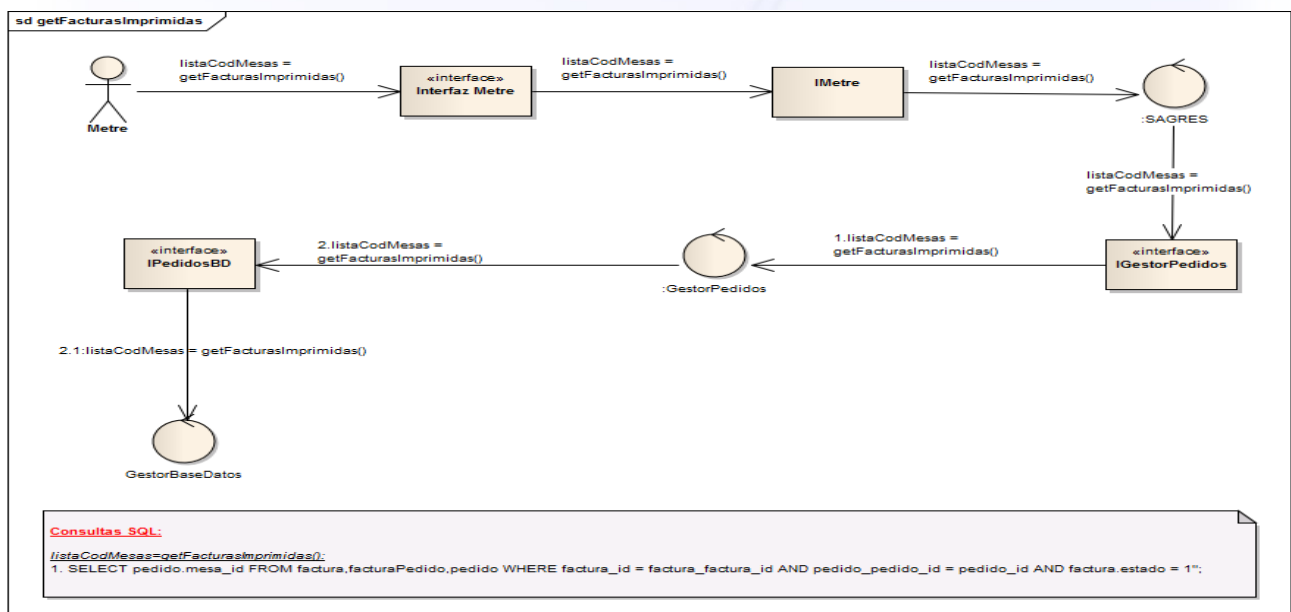
Es una funcionalidad que necesita el sistema para actualizar la cola de facturas pendientes de imprimir, esta función obtiene una lista de enteros que identifican a las mesas de clientes que piden una facturas.



Realizado por Nabil

getFacturasImprimidas

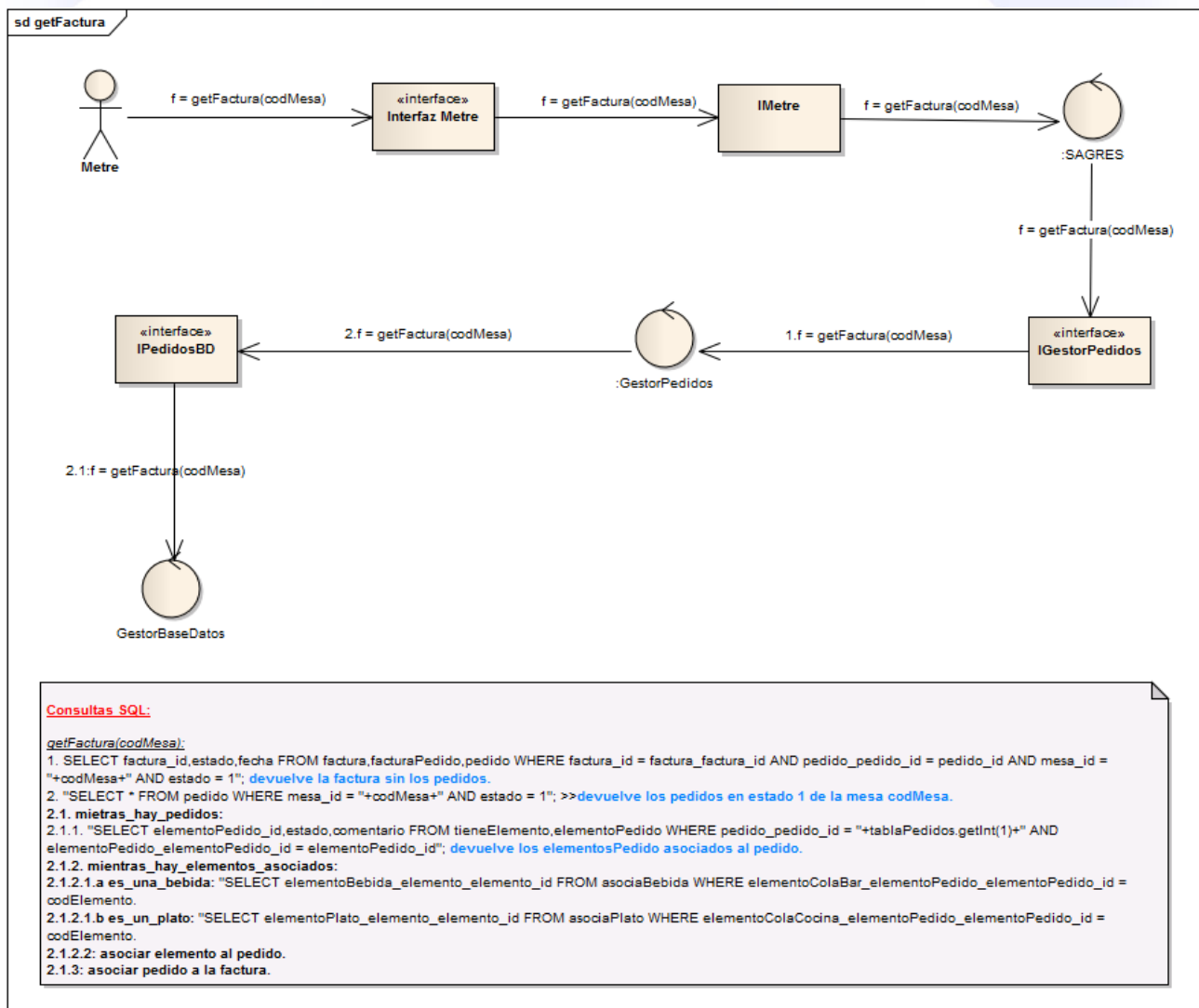
Es una funcionalidad que necesita el sistema para actualizar la cola de facturas imprimidas, esta función obtiene una lista de enteros que identifican a las mesas de clientes que sus facturas han sido imprimidas.



Realizado por Nabil

getFactura

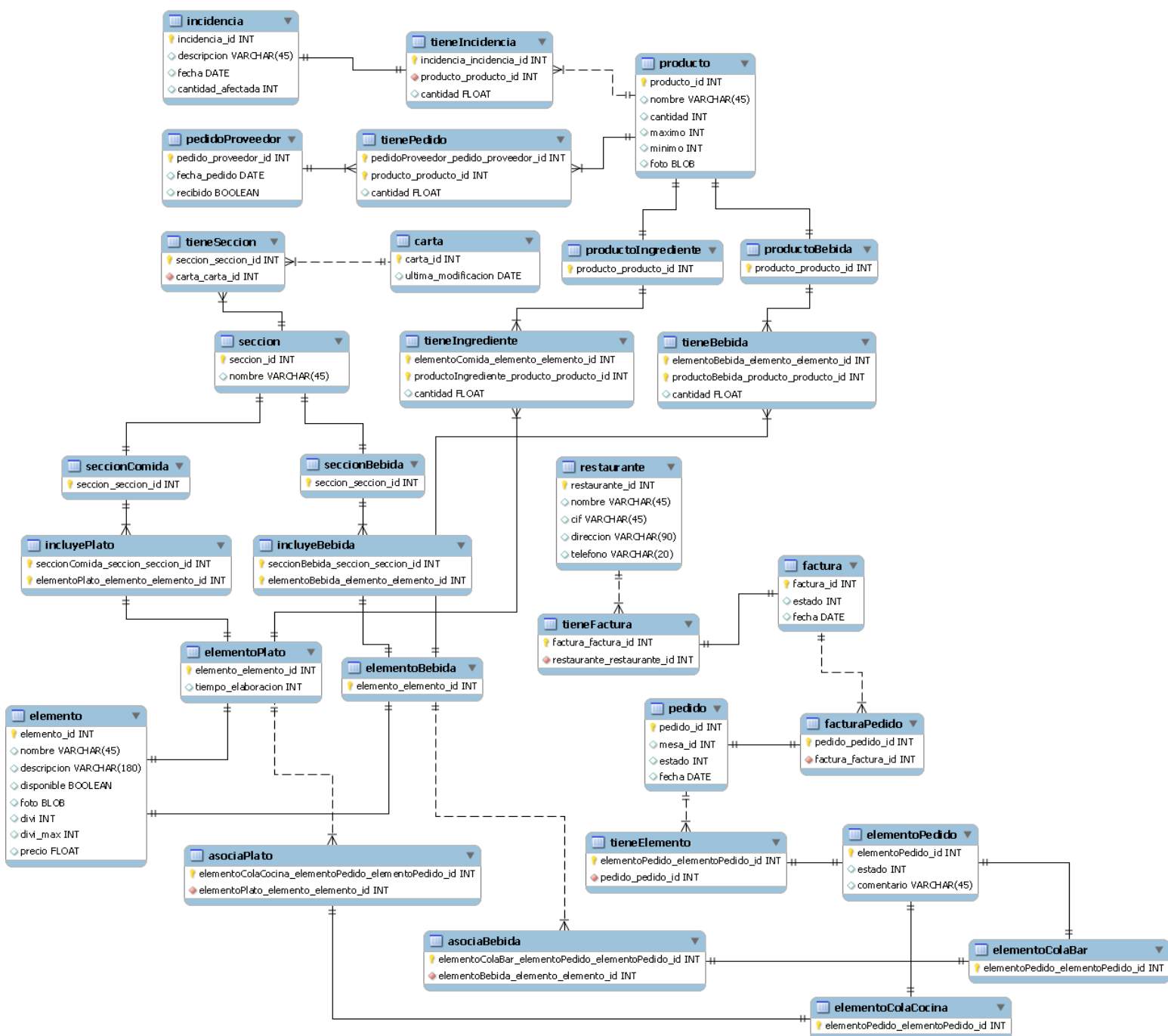
Es una funcionalidad que el sistema necesita para obtener una factura.



Realizado por Nabil

8. DISEÑO DE LA BASE DE DATOS

A continuación se muestra el diagrama relacional, que representa el conjunto de tablas utilizadas para construir la base de datos de todo el sistema actual. El proceso para obtener las tablas es sencillo. Mirando el diagrama de clases, se ha obtenido una tabla para cada clase o relación entre clases existente, introduciendo además en cada una como datos los atributos originales de la clase. En casos en los que ha sido necesario (como en el de las relaciones), se han añadido atributos nuevos para formar las claves primarias



Realizado por Adrián Victor Pérez Lopera

9. DISEÑO DE LAS INTERFACES DE USUARIO

La interfaz de usuario se ha realizado siguiendo los diseños realizados en la primera iteración. Se han utilizado tonos suaves de azules para dar una sensación agradable al usuario durante su utilización. Los diálogos y las ventanas son simples, tratando de dejar el máximo espacio libre para no estresar al usuario con demasiada información. Se ha seguido un coloreado plano junto con degradados suaves de azul a blanco en algunos puntos, para dar continuidad y espacio a cada escritorio.

Para comenzar, mostraremos el diseño reutilizado del diálogo de confirmación.

Confirmación Indica la acción que se va a confirmar

Añadir nuevo elemento a la carta

¿Confirma que desea añadir el siguiente elemento?

Sección :
Nombre:
Descripción:
Ruta a imagen:
Tiempo de elaboración:
Lista de productos asociados:
-
-
-

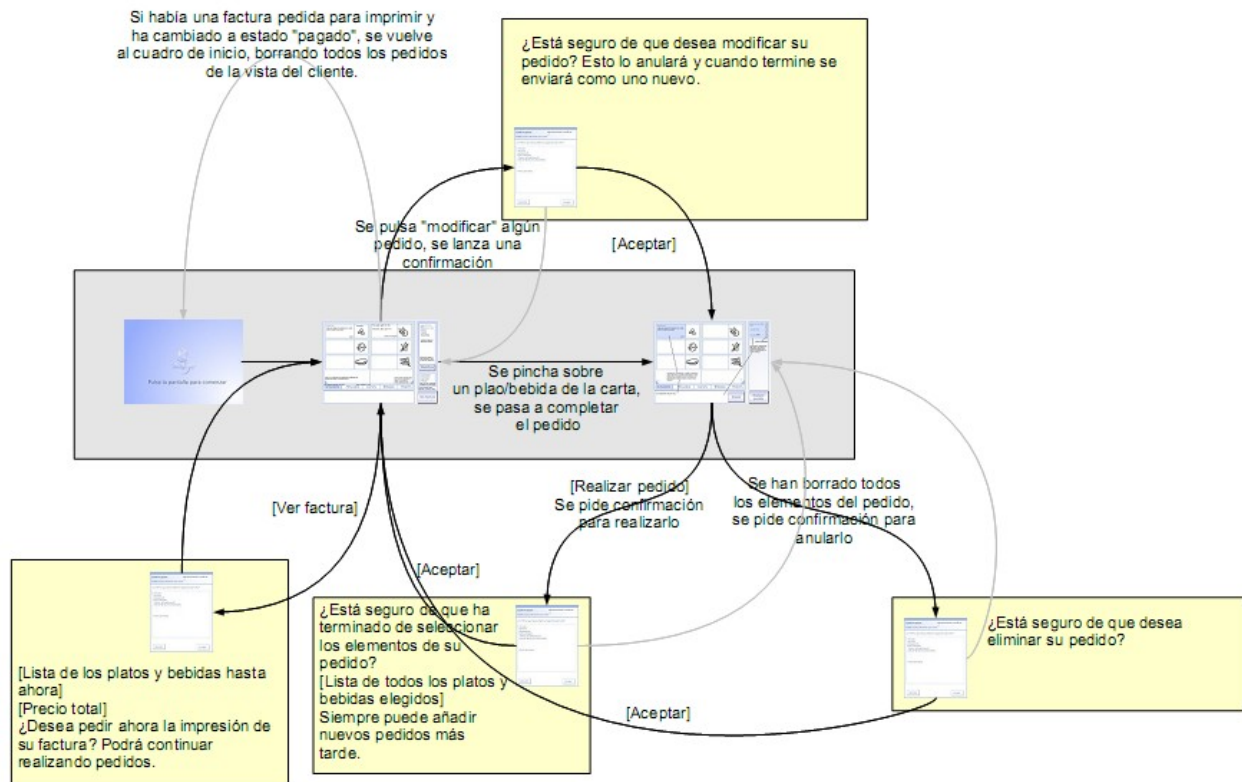
Precio de venta:

Cancelar Aceptar







A continuación mostraremos la línea de comunicación entre interfaces de usuario que sigue para cada usuario.

Interfaz de usuario de Cliente

A continuación se muestra la secuencia de interacción entre ventanas. Los cuadros en gris indican que todo ocurre en la misma pantalla (es decir, se cambia la imagen de la pantalla por otra). El resto son diálogos, los cuadros en amarillo indican lo que tienen que mostrar. Las flechas grises indican el regreso o bien la cancelación del diálogo.



Se puede ver una ventana limpia de información (salvo las anotaciones para el equipo de implementación). En esta interfaz se pretende mostrar únicamente la carta al cliente. En el lateral derecho se pueden ver los pedidos que actualmente se han realizado, si no se han comenzado a servir se podrán modificar. Los pedidos permanecen ahí hasta que se paguen.

<p>Aceitunas</p> <p>Aceitunas bañadas en aceite de oliva virgen extra de la sierra de Cazorla.</p> <p>2.00 €</p>	<p>(Fotografía)</p> 	<p>(Título, 14px, rgb(80, 98, 143))</p> <p>(Descripción, 10px, rgb(0, 0, 0))</p> <p>(Precio en la esquina)</p>	
			
			

(Esto es un multiPanel, los doblados de las esquinas son botones que llaman a multiPanel.next())

(color seleccion rgb(240, 240, 255))

(Pulsar sobre el doblado de la hoja, pasa a la siguiente página, si no hay más, no hay doblado de hoja)

(16px, rgb(80, 98, 143))

Entrantes	Pescados	Carnes	Bebidas	Postres
-----------	----------	--------	---------	---------

- Hamburguesa con queso.
- Macarrones con salsa bolognesa.
- CocaCola
- CocaCola
- Fanta naranja
- CocaCola light
- ...

(Listado de todos los elementos pedidos)

(Mientras el pedido no esté bloqueado, habrá un botón de "Modificar")

Modificar

- Perrito caliente
- Ensalada ligera
- Agua con gas
- Agua natural
- CocaCola

(Este ya se ha comenzado a preparar, no hay botón)

(Cuando el pedido está preparado, se quita de aqui)

Ver factura

Al pulsar sobre un elemento de la carta, se pasa a crear un nuevo pedido, en el que se va a la siguiente imagen (Igual que si se da a modificar un pedido y se acepta la confirmación).

Esta ventana representa modificar o crear un nuevo pedido. Si se va a modificar, aparecerá el cuadro de la derecha ya relleno con los productos. Si se va a crear nuevo, aparecerá vacío. Los productos se van añadiendo con el botón de “Añadir” o bien pulsando de nuevo sobre ellos. Seleccionarlos sirve para añadirles un comentario, tal como sucede en la imagen.

Se puede pasar de página en la carta pulsando en las esquinas de abajo, en el doblado de hoja.

The interface is divided into several sections:

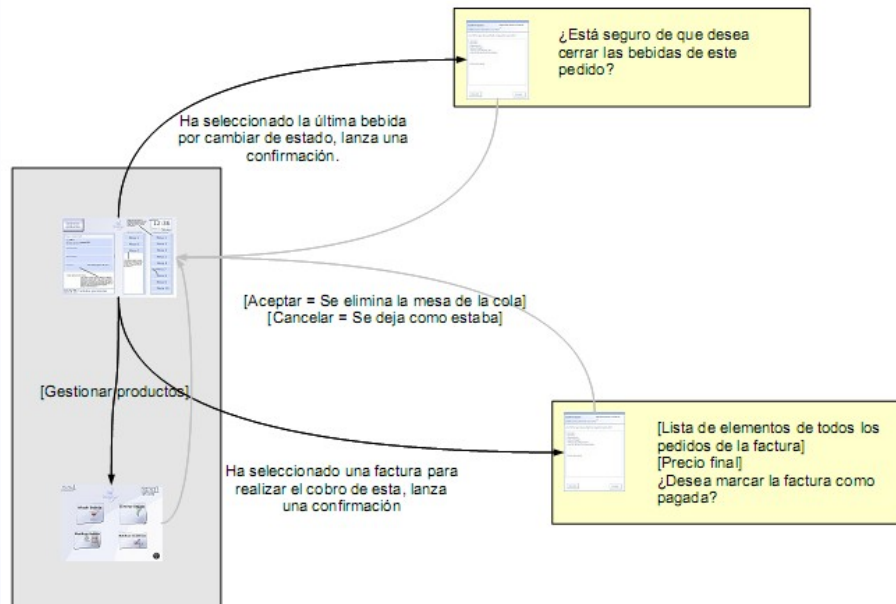
- Main Grid:** A 2x3 grid of items. The first item is "Aceitunas" (Olives) with a description "Aceitunas bañadas en aceite de oliva virgen extra de la sierra de Cazorla." and a price of "2.00 €". The other items are represented by icons: a hand, a leaf, and a fork.
- Right Sidebar:** A vertical list of items with their prices and a button to "eliminar el elemento" (remove the item). The items are: "Huevo duro con patatas fritas" (2.00 €), "Lechuga fresca" (1.00 €), and "Cocacola (10px)" (1.00 €). Below the list is a button "Realizar pedido" (Place order).
- Bottom Section:** A horizontal bar with tabs for "Entrantes", "Pescados", "Carnes", "Bebidas", and "Postres". Below the tabs is a text input field for comments, currently containing "Con bastante sal por fav|". To the right of the input field is a button "Añadir" (Add).

Annotations in the image provide additional context:

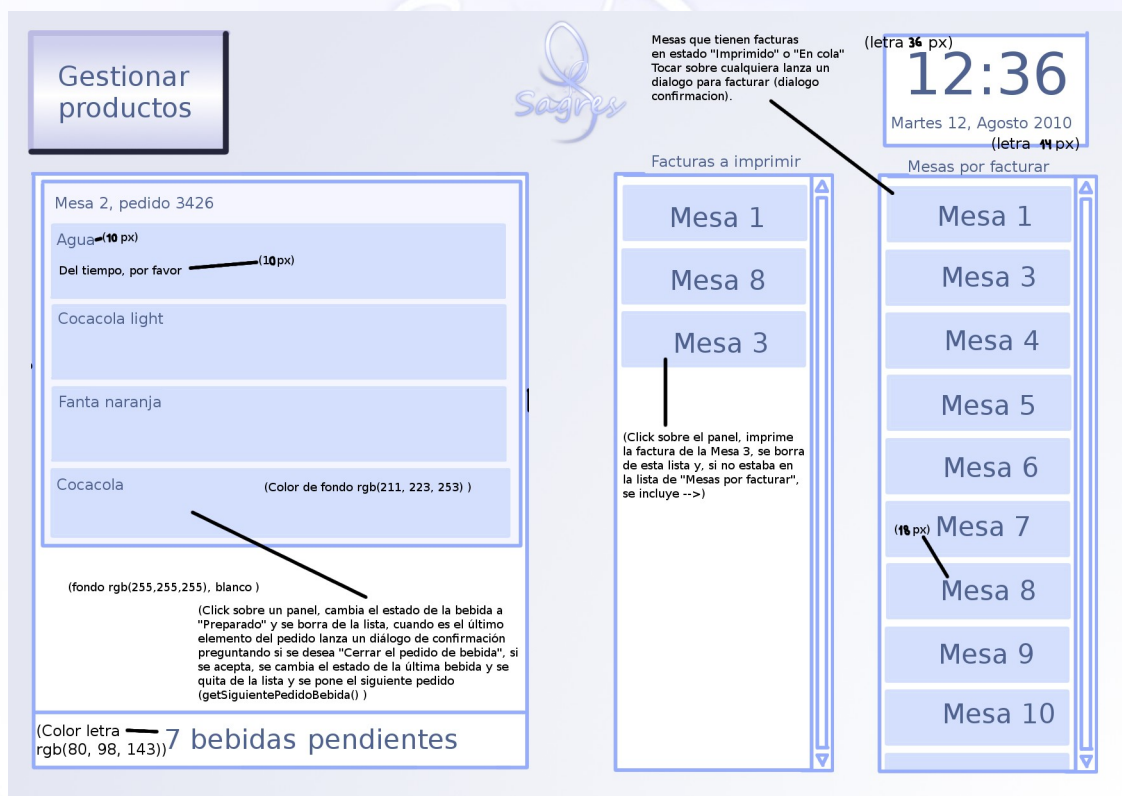
- A line points from the "Aceitunas" item to the comment field, with the text: "(Se selecciona un elemento, se añade el comentario abajo si se quiere y se pulsa el botón añadir, o bien se pulsa otra vez sobre el cuadro 'Aceitunas' (Los dos lo añaden al pedido actual en el cuadro de la derecha))".
- A line points from the "eliminar el elemento" button to the "Cocacola" item, with the text: "(Pulsa sobre un elemento de este cuadro y se activa, el comentario aparece en el cuadro de comentarios (Se puede modificar))".

Interfaz de usuario de Metre

A continuación se muestra, con las mismas reglas que en el caso anterior, cómo se pasa de una ventana a otra.



Aquí se muestra el menú principal del metre. La cola de bebidas muestra únicamente el pedido más antiguo con bebidas en estado “En cola”. Al pulsar sobre una bebida, esta se marca como “Preparado” y se elimina de la imagen. Cuando se han preparado todas, se carga el siguiente pedido.

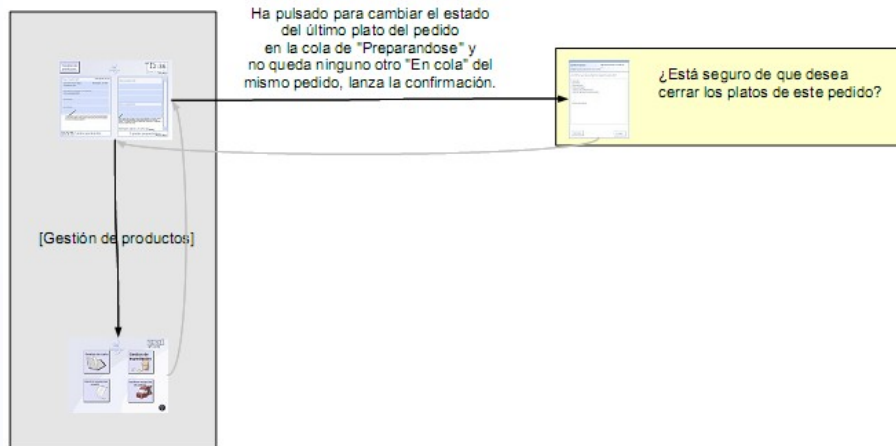


Las colas a la derecha muestran las mesas que han pedido imprimir una factura y las mesas que han pedido alguna factura y no han pagado aún. Con solo pulsar sobre una mesa en la cola de imprimir, se imprime y se quita de dicha cola. En su cola homóloga, al pulsar se pedirá confirmación para realizar el pago, mostrando un diálogo con los platos pedidos y el precio total, con las opciones de “cancelar”, “imprimir” o “aceptar”. Si se cancela se vuelve a la ventana anterior. “Imprimir” imprime la factura más reciente, sin cerrar el cuadro. “Aceptar” marca la factura como pagada y cierra el diálogo, eliminando además la mesa de la lista de mesas por facturar.

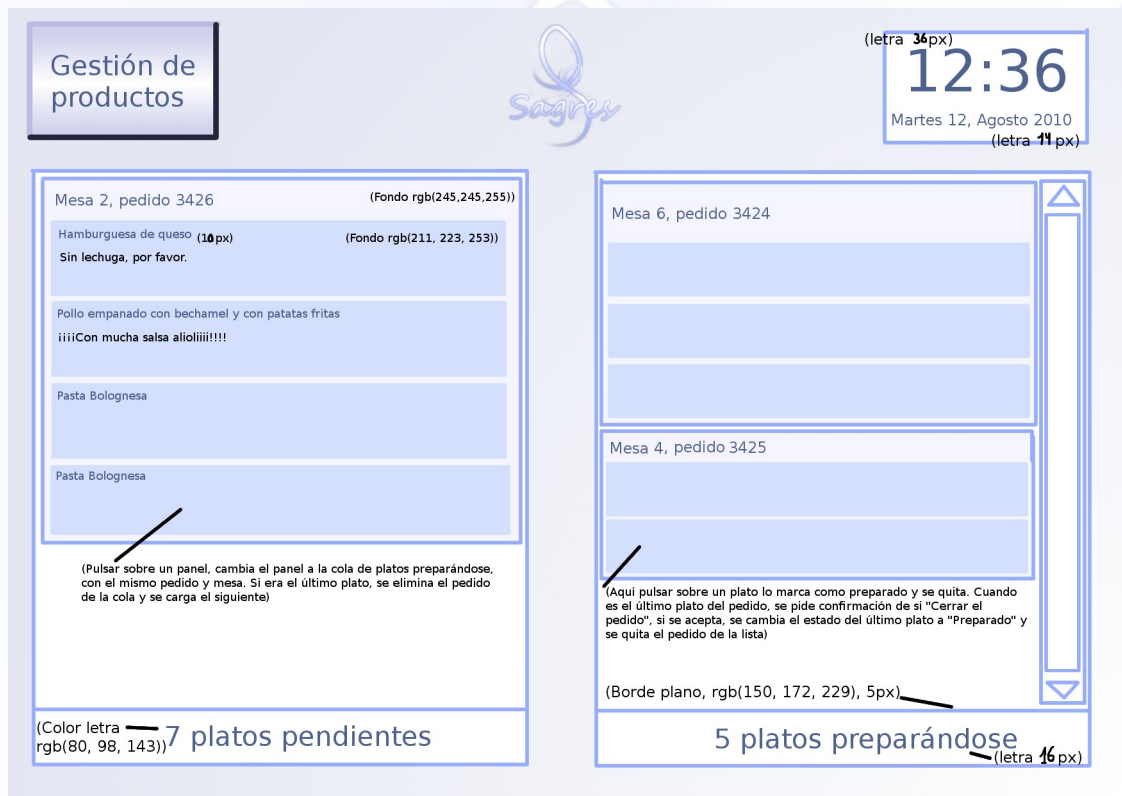


Interfaz de usuario de Cocinero

Al igual que sucede en los casos anteriores, se muestra cómo se pasa de una ventana a otra para el caso del cocinero.



Cuya interfaz principal es la siguiente:



En este caso, la cola de la izquierda muestra el pedido más antiguo con únicamente los platos en estado “En cola”. Cuando se pulsa sobre un plato, se marca como “Preparándose”, eliminándose de esta cola y pasando a la cola de la derecha. Cuando se marcan todos los platos, se carga el siguiente pedido por orden cronológico.

En la cola de la derecha se muestran todos los platos en estado “Preparándose” agrupados por pedido. El orden de los pedidos es el orden de llegada (el último pedido que entra, es el último en la cola). En este caso, cuando se pulsa en el último plato del pedido, se lanza una confirmación para cerrar el pedido de platos.

Formato de factura

Bienvenidos	
Duerme Mucho--Granada	
Elemento	Precio
Elemento	Precio
.	.
.	.
.	.
.	.
.	.
.	.
Elemento	Precio
	Total
fecha y hora	
Gracias por su visita	

Realizado por Nabil

hemos decidido que la factura debe tener el siguiente formato mostrado en la imagen:

- una cabecera que contiene el mensaje de bienvenida y el nombre del restaurante.
- Justo abajo la lista de elementos con su precio correspondiente, al final el total.
- Mas abajo la hora y fecha de impresión.
- Y al final el mensaje de “gracias por su visita”.

10. CONSIDERACIONES FINALES SOBRE EL DISEÑO

A continuación se detallan algunas consideraciones que se han notificado al equipo de implementación pero que requieren hacer cierto énfasis en el documento.

1. Todas las listas las cuales no se han especificado su tipo serán **por defecto** `ArrayList`. Utilizamos mucho este tipo de listas porque son las que mejor comportamiento presentan a la hora de recorrerlos iterativamente y, además, poder eliminar elementos en posiciones aleatorias sin tener que reestructurar el contenido. Estas dos operaciones mencionadas las utilizamos muy a menudo.
2. Las vistas están pensadas para que haya una hebra que periódicamente actualice los datos con los que se está tratando. Para ello se dispone de las funciones necesarias (como por ejemplo, `getSiguienteColaBar()` o bien `getPedidosCocinaPreparandose()`). En principio se ha pensado en un periodo de actualización de cinco segundos, tiempo suficiente para no “atosigar” demasiado a la base de datos pero también de mantener actualizados a los usuarios. Esto es susceptible a cambios una vez veamos cómo se comporta el sistema.
3. Se dispone de funciones que tal vez no sean necesarias para esta iteración (como `getPedidosCocinaPreparandose`, porque la vista puede cumplir su función de forma más sencilla), estas funciones permitirán al sistema recuperarse de la caída del terminal del metre, por ejemplo, manteniendo la consistencia y la persistencia de datos, cosa no contemplada para esta iteración.

Sagres

APÉNDICE 1.0

Fecha	02/05/10
Descripción del problema	
Impacto del problema	
Soluciones adoptadas	<ul style="list-style-type: none">• Se ha generado el documento de diseño.
Anexos a la versión	

Sagres

APÉNDICE 1.1

Fecha	07/05/10
Descripción del problema	<ol style="list-style-type: none"> 1. No se habían incluido todos los diagramas de transición entre ventanas en la interfaz de usuario. 2. Había errores en los diagrama de colaboración del subsistema de facturación. 3. Había error en el controlador de impresora. 4. No se había mostrado el formato de la factura. 5. Los diagramas de colaboración no detallaban lo suficiente la secuencia de acciones del sistema. 6. Faltaban detalles por especificar sobre la vista del sistema.
Impacto del problema	<ol style="list-style-type: none"> 1. No se podía conocer con claridad en qué momento se lanza un diálogo, o cambia la ventana. 2. Faltaban pasos a realizar en los diagrama de colaboración del subsistema de facturación. 3. La función <code>imprimeFactura(f,r)</code> no llevaba parámetros que dan los elementos de la factura y el total a pagar. 4. Imprecisión a la hora de reproducir el formato de las facturas. 5. La traducción entre el diseño y la implementación podría no ser directa. 6. Imposibilidad de conocer el comportamiento a bajo nivel de las vistas.
Soluciones adoptadas	<ul style="list-style-type: none"> • Se han incluido todos los diagramas y especificaciones en el documento. • Se han cambiado diagrama de colaboración “confirmaPagoFactura”, “imprime Factura” y se han quitado “nueva factura”, “imprimeFacturas”, “inicializar” • se ha cambiado la función <code>imprimefactura(..)</code> del controlador de impresora. • Se ha agregado un boceto explicativo de las facturas. • Se han detallado los diagramas de colaboración hasta el nivel de base de datos. • Se ha creado una nueva sección en la que se especifican algunos detalles comentados al equipo de implementación pero que no tienen apoyo de diagramas u otra forma de información.
Anexos a la versión	

APÉNDICE 1.2

<i>Fecha</i>	13/05/10
<i>Descripción del problema</i>	1. No estaban actualizados en el documento ciertos diagramas, como el diagrama de clases, la división del mismo en subsistemas y la identificación de las interfaces de los distintos subsistemas.
<i>Impacto del problema</i>	1. No se corresponde cierta información del documento con el sistema actual.
<i>Soluciones adoptadas</i>	<ul style="list-style-type: none">• Se han actualizado todos los diagramas desfasados.
<i>Anexos a la versión</i>	

Sagres

APÉNDICE 1.3

<i>Fecha</i>	18/05/10
<i>Descripción del problema</i>	1. El equipo de implementación informó sobre posibles mejoras en los diagramas de secuencia de “getSiguientePedidoCocinaEnCola”, “getSiguientePedidoBar” y “getPedidosCocinaPreparandose”. Obtener todos los pedidos para buscar el necesario en las operaciones era muy costoso y poco eficiente.
<i>Impacto del problema</i>	1. Baja eficiencia en el sistema y una gran complicación de código.
<i>Soluciones adoptadas</i>	<ul style="list-style-type: none">Se ha rediseñado la estructura de las operaciones, delegando las funciones que antes hacía el gestor de pedidos directamente al gestor de base de datos. Además se ha especializado la consulta para cada operación, en lugar de utilizar una genérica en las tres. Como consecuencia de esto, se ha borrado la operación “obtienePedidosNoFacturados”, ya que no se utiliza en el sistema. Estos cambios se han reflejado además en los diagramas de componentes.
<i>Anexos a la versión</i>	