3 DESARROLLO

3.1 ESTUDIO DE LA FORMA DE TRABAJO DEL OPERADOR

La idea de este apartado es desarrollar de acuerdo con la empresa de distribución, un breve estudio de campo sobre la forma de la forma de trabajar del operador encargado del *picking* de productos.

En una primera aproximación se puede ver que el proceso de *picking* es totalmente manual, ya que el operario se basa normalmente en una hoja de pedido impresa para ir recogiendo los productos del almacén y llevarlos a la zona de entrega. Esto puede provocar errores debido al factor humano de la operación. Por ejemplo tomar un producto de una ubicación incorrecta, o realizar recorridos más largos dentro del almacén. Un recorrido más largo implica que el operador tiene que gastar más tiempo realizando el *picking*, repercutiendo esto en la productividad y aumentando el coste operativo.

3.1.1 TAREAS DEL OPERADOR

A continuación se detallan los casos principales que habitualmente debe llevar a cabo un operador, indicando de forma esquemática la secuencia de pasos.

3.1.1.1 Picking basado en un pedido realizado por un cliente

Esta es una de las tareas más frecuentes que debe realizar un operador. El cliente ha solicitado un pedido y queda pendiente de servir. Un encargado o jefe de almacén revisa estos pedidos y confirma que dispone en almacén del stock que requiere el pedido. Una vez que se confirma el stock, encarga a un operador el *picking* de uno o más pedidos.

1. El jefe de almacén revisa los pedidos pendientes de servir y los reparte de forma impresa a los operarios según su criterio (por ejemplo la fecha de entrega).
2. El operario recibe su listado de pedidos
3. El operario toma el primer pedido de la lista
4. El operario se dirige a la ubicación concreta del almacén donde se encuentre uno de los productos que tiene que servir y toma las unidades indicadas en la hoja de pedido.
5. El operario deposita en la zona de recogida el material tomado
6. El punto 4 y 5 se repiten las ocasiones que sean necesarias hasta completar el pedido.
7. El pedido está listo para servir, el operario se sitúa en un TPV y escanea todos los códigos de barras de los productos recogidos, generando un albarán de salida y/o una factura de venta.
8. Los productos están listos para ser empacados o preparados para el transporte.
9. El operario toma el siguiente pedido de su lista y vuelve al punto 4 hasta que ya no le quedan más pedidos que servir.

3.1.1.2 Picking “ad-hoc”

En este caso el cliente se presenta directamente en el mostrador del almacén, sin haber realizado un pedido previo. Un operador atiende al cliente el cual le solicita los productos que desea y estos son servidos sobre la marcha por el operador.

1. El cliente se presenta en el almacén y es atendido por un operario
2. El operario toma nota de los productos que solicita el cliente

3. El operario se dirige a la ubicación concreta del almacén donde se encuentre uno de los productos que tiene que servir y toma las unidades indicadas
4. El operario deposita en la zona de recogida el material tomado
5. El punto 3 y 4 se repiten las ocasiones que sean necesarias hasta completar el pedido del cliente.
6. El pedido está listo para servir, el operario se sitúa en un TPV y escanea todos los códigos de barras de los productos recogidos, generando un albarán de salida y/o una factura de venta.

3.1.2 PROBLEMAS HALLADOS EN EL PROCESO DE PICKING

A continuación se describen brevemente los problemas en cuanto a eficacia a la hora de realizar el *picking* que se observan en los operadores. En particular puede darse uno o más de los siguientes problemas:

3.1.2.1 Errores en la identificación del producto tomado.

Esto puede venir originado por dos motivos:

- El operario se ha dirigido a una ubicación incorrecta y está tomando un producto que no está en el pedido
- El operario está en la ubicación correcta, pero en ella se ha colocado un producto de otra ubicación al reponerlo. Por tanto el operario toma el producto erróneo pensando que pertenece a esa ubicación.

3.1.2.2 Errores en el recuento de productos.

El operario se equivoca contando las unidades que debe servir en el pedido bien por exceso o por defecto.

3.1.2.3 Errores de transcripción

Una vez finalizado el *picking* el operario debe transcribir a la aplicación del ERP de la empresa los artículos y cantidades recogidas para poder generar un documento de salida de mercancía. Este proceso de transcripción puede dar lugar a errores.

3.1.2.4 Determinación de diferencias subóptima

El operario debe llevar un control totalmente manual sobre los artículos y cantidades que ha recogido y las que no. Normalmente esto se realiza de forma manual tomando notas sobre la propia hoja de pedido.

Esto puede retrasar el *picking* si el operario debe detenerse en algún momento a determinar las diferencias entre lo que ha recogido y lo que le queda por servir, en particular con pedidos grandes.

3.1.2.5 Recorrido subóptimo en un pedido

De la lista de ubicaciones que aparece en el pedido, el operador elige a su criterio el recorrido entre ellas pudiendo no ser éste el más óptimo. A mayor distancia recorrida transcurre mayor tiempo en completar la preparación del pedido y por tanto la productividad es peor y el coste operativo aumenta.

Cabe destacar que indicar de forma automatizada un recorrido óptimo a un operador es algo muy complejo y que depende de múltiples factores y en ocasiones un operador experimentado puede incluso ser más eficaz que un algoritmo mediocre.

3.2 APLICACIÓN DE CÓDIGOS DE BARRAS

Una vez estudiado la forma de trabajo del operador se observa la conveniencia de aplicar de forma efectiva una tecnología de autoidentificación como las que se han discutido anteriormente en la sección 2.1. Tras analizar las diferentes ventajas y desventajas de las distintas tecnologías se elige aplicar el código de barras. Los motivos para elegir esta tecnología frente a las otras son varios:

- El coste por etiqueta es muy bajo. En muchas ocasiones se puede utilizar el código de barras del proveedor que ya trae el producto en lugar de generar uno interno, si se registra correctamente en el ERP de la empresa.
- Dispone de un estándar estable
- Existen múltiples dispositivos disponibles en el mercado a un precio asequible para trabajar con códigos de barras
- Aunque el RFID se descarta principalmente por su costo, una de sus desventajas es que provoca interferencias cuando una etiqueta va adjunta a un objeto de metal. Dado que la empresa distribuye materiales eléctricos, se observa que en ciertas referencias podría llegar a haber problemas para su uso.

3.2.1 REUTILIZACIÓN DE LOS CÓDIGOS DE BARRAS DEL PROVEEDOR

El proveedor en muchas ocasiones nos proporciona los productos con el código de barras impreso en el embalaje del producto o en el propio producto por lo que se plantea utilizar en estos casos dicho código en lugar de generar uno propio.

Aunque se trata de un código de barras particular al proveedor en cuestión, a la empresa le resulta mucho menos costoso registrar este código de barras en su ERP y relacionarlo con la referencia en cuestión que tener que emitir etiquetas con nuevos códigos de barras para este producto y aplicárselo.

Inconvenientes en la reutilización de códigos de barra

- Diferentes proveedores pueden utilizar diferentes tipos de códigos de barras. Aunque el EAN-13 sea el más común podemos encontrar otros como EAN-8, UPC-A o el GS1-128 (alfanumérico) con lo que tenemos que asegurar su correcta lectura en cada ocasión. Los dispositivos lectores de códigos de barras normalmente vienen preparados para la lectura de estas diversas codificaciones.
- Normalmente los códigos de barras dentro de cada proveedor son estables pero podría darse el problema de que el proveedor modificara su código de barras. En este caso la empresa debe ser capaz de darse cuenta de este cambio y modificarlo en su ERP correspondiente.
- Una variante del problema anterior sería que un proveedor mantenga un código de barras para dos artículos diferentes. Esto es habitual si por ejemplo se descataloga un artículo para dar lugar a otro más moderno pero el proveedor decide respetar el código de barras. El problema para la empresa es que si tienen stock tanto del artículo descatalogado como del nuevo deben reetiquetar uno de los dos para evitar confusiones.
- También hay que tener en cuenta los factores logísticos a la hora de registrar el código de barras en el ERP. Algunos proveedores indican su código de

barras en un paquete de varias unidades donde estas unidades no van identificadas. Sin embargo otros proveedores identifican de forma individual cada unidad dejando el paquete sin identificar. Como la unidad logística de venta puede no coincidir con la de compra puede ser necesario tener que reetiquetar estos productos

- Otra situación que podría darse muy raramente sería el que dos proveedores utilizaran un mismo código de barras interno para identificar a dos productos diferentes. Este problema se detectaría relativamente fácil en el momento de registrar ambos códigos de barras en el ERP. La solución sería generar y aplicar a una de las dos referencias un código de barras propio de la empresa.

3.2.2 GENERACIÓN DE CÓDIGOS DE BARRA INTERNOS

En el caso de que un producto no venga preetiquetado con un código de barras de un proveedor, o incurramos en algunos de los casos previstos en la sección 0 la empresa debe ser capaz de generar un código de barras propio o interno.

Entre los pasos a seguir, se incluye la generación del código de barras, la asociación de dicho código con el producto dentro del ERP de la empresa, la impresión de un número de etiquetas de código de barras necesarias para identificar todos los productos, y su correcta aplicación sobre los mismos por parte de los operadores.

3.2.2.1 Formato

El formato elegido para la numeración de códigos internos es el EAN-13, pues tal y como hemos visto en la sección 2.2 es un formato estandarizado, conocido y ubicuo. En la sección 2.2.1.1 también se puede ver cómo los códigos EAN-13 que tienen un código GS-1 dentro del rango *200-299* se dedican a funciones internas de la empresa.

Se decide entonces utilizar concretamente el prefijo GS-1 con rango *290-299* permaneciendo el rango *200-289* sin uso alguno en lo concerniente al ámbito de este proyecto. Dentro del rango especificado se decide además utilizar concretamente el valor GS-1 *299* para indicar productos con un código de barras interno. Excluyendo dicho código GS-1 y el dígito de verificación, esto nos dejaría libres 9 dígitos (999.999.999 códigos válidos), capacidad de sobra para codificar un gran número de productos. El rango restante *290-298* se reserva para comandos de la aplicación, como se verá más adelante.

3.2.2.2 Modelo de etiqueta

Se acuerda con la empresa que se aprovechará la etiqueta para imprimir otros datos humanamente legibles para el operador que pueden ser de interés. Por tanto además del código de barras podremos encontrar:

- El nombre de la empresa, para identificar visualmente un código de barras interno de uno ajeno
- El código del producto dentro del ERP de la empresa
- La descripción del producto dentro del ERP de la empresa
- El nombre del proveedor principal del producto
- El código del producto en la Plataforma Electronet

3-1 Modelo de Etiqueta

En la imagen 3-1 podemos observar en la parte superior el código EAN en formato numérico, a la izquierda el nombre de la empresa, a la derecha el código interno del producto, en la parte inferior el proveedor y código de la Plataforma Electronet (primera línea) y a continuación la denominación del producto (segunda línea).

3.2.2.3 Guía para la generación

La empresa decide que la aplicación que debe generar los códigos de barras internos discutidos en la sección 3.2.2 va a ser un módulo dentro de su ERP. Por tanto aunque dicho módulo quedaría fuera del alcance de este proyecto, se va a definir a continuación a grandes rasgos cómo debería ser su implementación, junto con algunas recomendaciones.

- Debe existir una relación uno a varios entre el objeto de negocios que representa un producto dentro del ERP y un código EAN.
- Un código EAN debe existir de forma única, de forma que no pueda estar relacionado con más de un producto. Un EAN identifica unívocamente a un producto.
- Un artículo debe admitir un código EAN de cualquier formato o longitud.
- Si un artículo tiene relacionado un código en formato EAN-13, este debe ser validado según el algoritmo discutido en el Anexo D. Validaciones de otros formatos no están excluidas.
- Debe existir un dato modificable por el usuario de Factor Logístico asociado a la combinación producto-EAN, de cara a evitar inconvenientes como los discutidos en 3.2.1
- Debe programarse una interfaz de usuario cómoda y versátil para asociar y desasociar un producto con un código EAN, siempre que se cumplan todas las validaciones y restricciones anteriores.
- Se debe incluir una opción para autogenerar un código numérico EAN-13 interno para un producto concreto según las especificaciones discutidas en 3.2.2.1
- Debe existir una opción de sacar en formato impreso un número paramétrico de etiquetas según el formato discutido en 3.2.2.2
- Debe programarse una interfaz de usuario ágil y cómoda para asociar y desasociar un producto a un código EAN, siempre que se cumplan todas las validaciones y restricciones anteriores.

3.3 DISPOSITIVOS LECTORES DE CÓDIGOS DE BARRAS

En este apartado se comenta el modelo concreto del ordenador de mano industrial que se va a utilizar para efectuar la lectura del código de barras y su posterior interpretación. Los requisitos que se estudian sobre los terminales disponibles en el mercado son los siguientes:

- El terminal debe disponer evidentemente de lector de códigos de barras 1D. Se valorará que adicionalmente pueda leer Códigos 2D y Matrix.
- En cuanto a comunicaciones el terminal debe tener WiFi. Aunque la cobertura dentro del almacén no está asegurada al 100%, es conveniente que la aplicación se comunique de forma inalámbrica allí donde haya cobertura. Se valorarán otros dispositivos de comunicación tales como Bluetooth o 3G.
- Debe ser un terminal robusto y resistir en la medida de lo posible golpes y caídas
- Factor de forma y peso
- Coste del dispositivo

De acuerdo con la Empresa y tras evaluar algunos modelos, se decide que el terminal escogido será de la marca Datalogic, modelo Skorpio X3. Éste cumple sobradamente con los requisitos expuestos previamente.

En principio limitarnos a trabajar únicamente con un modelo de dispositivo de mano simplifica las labores de programación, pero nos crearía una dependencia excesiva de dicho modelo específico. Sin embargo hay que destacar que una aplicación desarrollada para un dispositivo que soporta .NET CF no es excesivamente dependiente del mismo ya que el CLR nos abstrae de la máquina subyacente. Únicamente ciertas librerías de acceso a hardware (por ejemplo acceso al Wifi o Bluetooth) requerirían ser reprogramadas en caso utilizar otro dispositivo. Esto por tanto nos da cierta flexibilidad en un futuro a la hora de seguir utilizando un dispositivo modelo Skorpio o buscar una alternativa cualquiera que soporte .NET CF.

3.4 DESARROLLO DE LA APLICACIÓN

3.4.1 ESTRUCTURA GENERAL

El grueso de la solución viene dado por el desarrollo de una aplicación que se ejecuta en el ordenador de mano industrial. Esta aplicación se encarga de leer e interpretar los códigos de barras que lee el operador y realizar los comandos subsiguientes tales como solicitar la recepción de un pedido, realizar una lectura de existencias o enviar el pedido una vez finalizado, conectándose normalmente a la red de área local a través de un enrutador inalámbrico. Asimismo del lado del servidor contaremos con un servicio web que orquestará las comunicaciones entre los dispositivos y el servidor ERP de la empresa

Señalar que aunque se ha procurado respetar en la medida de lo posible, dadas las limitaciones del entorno .NET CF no se ha podido desarrollar una aplicación que cumpla estrictamente con diversos patrones de programación como MVC ni tampoco una separación pura entre las capas de interfaz de usuario, lógica de negocios y acceso a datos. Esto es así porque .NET CF está orientado a desarrollar aplicaciones de negocios embebidas y de no muy alta complejidad con lo que las prácticas habituales de buena programación no aportan tanta ventaja en cuanto al mantenimiento del código. La ventaja de este entorno es que es más sencillo desarrollar una aplicación orientada a datos si utilizamos las herramientas estándar de dicho entorno.

3.4.1.1 Idea general del funcionamiento de la aplicación

La premisa de la aplicación siempre será ser una herramienta más del operador, nunca sustituir sus funciones. El dispositivo ayudará al operador a identificar los artículos servidos y vigilar que las cantidades sean las correctas.

Además se desea que el operador tenga que utilizar el teclado del dispositivo lo menos posible, para evitar errores en las pulsaciones. Por tanto los comandos habituales que un operador puede ejecutar en la pistola los llevará impresos en formato de código de barras en una hoja o tarjetero que debe llevar encima mientras realiza el proceso de *picking*. El propio código del operador se codificará en un código de barras de tal forma que el operador leerá este código con el dispositivo para identificarse.

Cuando un operador tenga que realizar un *picking* de un pedido recibirá la hoja de pedido tal y como venía haciendo hasta ahora. Sin embargo a este impreso se le habrá añadido un código de barras que codifique la serie y el número del pedido. Una vez seleccionado el pedido del que se va a realizar el *picking* el operador debe descargar en el dispositivo el detalle de las cantidades pedidas de cada artículo de dicho pedido. Este proceso se puede repetir para varios pedidos de forma que el dispositivo de mano almacene la información de dichos pedidos y el operador pueda realizar el picking de una vez.

Seguidamente realizaría el *picking* guiándose con su hoja de pedido, con la diferencia de que para cada artículo seleccionado el operador realizará una lectura de su código de barras con el dispositivo. Una vez terminado el *picking* el operador envía las lecturas al servidor y puede realizar un documento de venta directamente desde esta información.

En resumen estos serían los pasos a seguir:

1. El operador recibe sus hojas de pedido y selecciona de cuales de ellas va a realizar el *picking*.
2. Lectura del código de barras de identificación del operador
3. Lectura del código de barras impreso en la hoja del pedido para señalar el pedido en curso.
4. Lectura del código de barras del comando "descargar pedido" para descargar en el dispositivo las cantidades solicitadas (en zona con cobertura)
5. Se repiten los pasos 3-4 para cada pedido adicional que quiera servir simultáneamente.
6. Realización del *picking* leyendo el código de barras de cada artículo con el dispositivo. Para alternar lecturas entre pedidos se leerá el código de barras impreso en la hoja de pedido para seleccionar el pedido en curso.
7. Una vez terminado el *picking* se realiza la lectura del comando "cargar pedido" (en zona con cobertura).
8. En el ERP de la aplicación se localiza la lectura cargada y se da la opción de generar un documento de venta a partir de los datos enviados.

En el caso de un *picking* sin pedido la secuencia sería similar salvo que el operador leería un código de barras genérico "*picking* sin pedido" de su tarjetero en lugar del código de la hoja de pedido y se omitiría el 4º paso de "descargar pedido" ya que no hay información en el servidor sobre las cantidades a descargar.

Ventajas del picking con el dispositivo de mano

El dispositivo de mano va a avisar de diferentes formas al operador de diversas formas activas y pasivas de si está realizando correctamente el *picking*.

- Al realizar la lectura de un artículo el dispositivo muestra en pantalla datos relevantes para el operador, como el código lógico y descripción del mismo. Estos datos deberían coincidir con los datos que tiene el operador en la hoja de pedido impresa, en caso contrario estaría leyendo un artículo erróneo.

- Si tiene que leer múltiples cantidades de un mismo artículo el operador puede revisar la cantidad servida que se irá acumulando en la pantalla del dispositivo cada vez que lea el código de barras del artículo en lugar de contar por sí mismo.
- Si el operador recoge más cantidad de la pedida el dispositivo avisará mediante una alerta acústica una vez superada esta cantidad.
- El operador puede invocar en cualquier momento un comando de "diferencias" que le dará un resumen por pantalla de los artículos que lleva servidos y los que aún están pendientes
- Al finalizar el *picking* el operador genera un documento de venta directamente desde la lectura realizada, por tanto no hay error a la hora de transcribir esta información
- Un operador puede ahora combinar más fácilmente el *picking* de múltiples pedidos o pedidos ad-hoc ya que el dispositivo de mano carga la información relevante de dichos pedidos.
- Los problemas analizados en 3.1.2.5 relativos al recorrido no van a ser considerados dentro del alcance de este proyecto debido a la complejidad de implantar un almacenamiento de los productos en base a la ubicación. Esta será una probable línea futura de mejora de este proyecto.

3.4.1.2 Codificación EAN-13 de los comandos del dispositivo

Tal y como se ha visto en 2.2.1.1 un código EAN-13 tiene un prefijo GS-1 que normalmente identifica el país del artículo. Sin embargo para nuestros comandos internos vamos a utilizar el prefijo 200-299 que están destinados para funciones internas y por tanto no debería colisionar con ningún código de barras de ningún artículo existente. Concretamente nos vamos a ceñir al rango 290-298, ya que como se ha visto en 3.2.2.1 hemos reservado el prefijo 299 para codificar artículos con un EAN interno.

- Codificación de comandos invariables
- Respetando el prefijo GS-1 *290-298* el resto de datos salvo el dígito de verificación puede elegirse arbitrariamente, por tanto relacionamos aquí los códigos escogidos para cada comando.

Nombre del comando	Código EAN13	Representación impresa
Descargar documento	2920000000009	
Cargar documento	2930000000008	

Picking sin pedido	2970000000004	
Pantalla de Resumen	2960000000227	
Anular última lectura	2940000000229	
Borrar operación en curso	2940000000007	

3-2 Codificación de comandos EAN-13

- Codificación del comando de Asignación de Operador

El código de barras para identificar el operador que está utilizando el dispositivo será variable pues en el propio código irá codificado el código lógico del operador. La forma de codificar elegida será basándonos en el prefijo *298*, seguidamente 9 dígitos que codifican el código lógico del operador y terminando con el dígito de verificación calculado según se puede ver en el Anexo D.

Por ejemplo el operador con código lógico 81220 se codificaría así:

- Codificación del comando de Selección de Pedido

En la hoja de pedido impresa se ha agregado un código de barras que utiliza un operador para indicar que ése es el pedido en curso. De forma similar al punto anterior y teniendo en cuenta que un pedido se identifica por dos claves numéricas *Serie* y *Número*, éstas se codificarán en el propio código de barras. La forma de codificar elegida sería utilizando el prefijo *291*, seguidamente 3 dígitos para codificar la Serie, 6 dígitos para codificar el Número y finalizar con el dígito de verificación calculado según se ha visto en el Anexo D.

Por ejemplo el pedido con Serie 81 y Número 57254 se codificaría así:

3.4.1.3 Interfaz de usuario de la aplicación .NETCF

El interfaz de usuario que se pretende diseñar busca varios objetivos:

- Debe presentar la mayor información posible útil para el operador de un vistazo, evitando tener que navegar por distintas pantallas para recabar dicha información.
- La entrada de datos debe ser simple. Un dispositivo de mano admite entrada por varios canales tales como teclado físico, pantalla táctil (a través de teclado en pantalla o de elementos del interfaz de usuario) y el propio escáner de códigos de barras. Para simplificar la entrada de datos se pretende que ésta venga dada en la medida de lo posible a través del escáner de códigos de barras.
- Aunque el dispositivo muestra información en pantalla, dado que éste cuenta con un altavoz se desea que haya confirmaciones auditivas. Esto va a permitir a un operador trabajar con el dispositivo de mano sin tener que mirar continuamente la pantalla.

3.4.1.4 Modo conectado y desconectado

Dado que en el almacén es problemático obtener una cobertura inalámbrica total además de que se desea que el trabajo del operario no se vea interrumpido por problemas derivados de trabajar continuamente en modo conectado, la aplicación debe detectar esta situación y ser capaz de conmutar de un modo conectado a uno desconectado y viceversa. Se utilizará para ello una caché de datos en el propio dispositivo donde se almacenarán las lecturas para permitir seguir trabajando cuando no haya conexión y se volcará dicha caché hacia el servidor una vez se halle en una zona con cobertura.

3.4.1.5 Almacenamiento de datos local

Para evitar la pérdida de datos durante una lectura en caso de rotura del dispositivo, finalización de carga de la batería o simplemente para sobrevivir a reinicios voluntarios o involuntarios tanto de la aplicación como del dispositivo se hace necesario contar con una pequeña base de datos en el dispositivo de mano para almacenar localmente la caché de lecturas. La aplicación cliente del dispositivo de mano se conectará a esta base de datos local donde se almacenarán las cantidades leídas de cada producto y sus respectivos códigos de barras.

3.4.1.6 Servicio Web

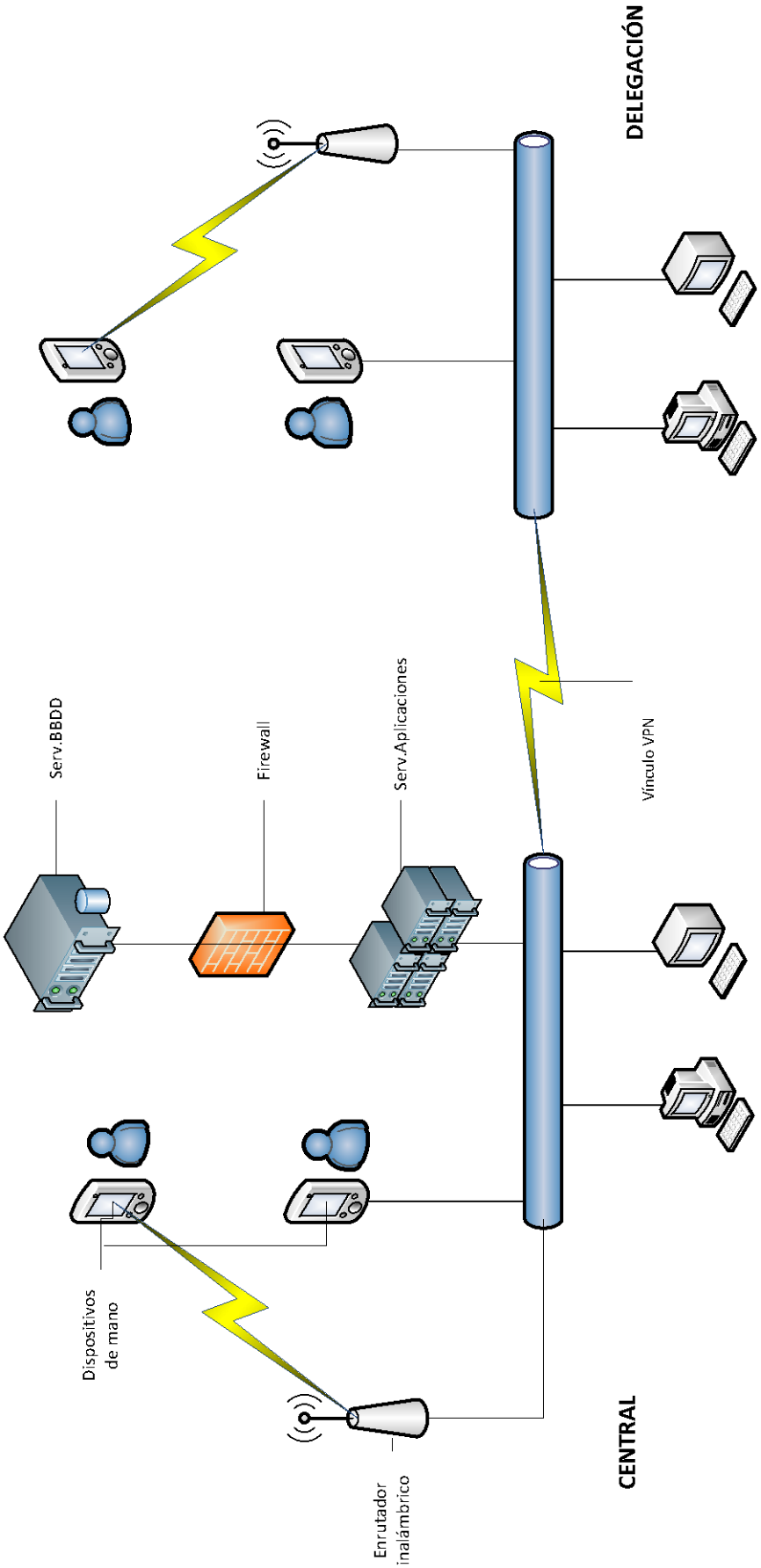
El cliente del dispositivo de mano se debe comunicar con el ERP de la empresa para recuperar los pedidos que se van a leer y enviar los datos resultantes de las diversas lecturas. Sin embargo nos encontramos el problema de que el ERP no tiene un interfaz para que dispositivos móviles accedan al mismo. Una opción sería que los dispositivos actuaran directamente contra la base de datos del ERP pero esto no parece una solución viable ya que entre otras cosas dicha base de datos está protegida con un cortafuegos al que sólo puede acceder en principio el servidor de aplicaciones del ERP.

La solución propuesta se trata de implementar un servicio web dentro de la red de área local de la empresa que actuará a la vez como servidor de los dispositivos y como cliente del ERP a través de un interfaz de programación que éste provee. Por tanto la aplicación del ordenador de mano debe ser capaz de conectarse bajo determinados comandos a este servicio web.

3.4.1.7 Diagrama de Red y despliegue físico

En la figura 3-3 se representa la red de la empresa con sus diferentes elementos. Los dispositivos de mano se conectarán a dicha red a través de un enrutador inalámbrico o directamente a través de una base de conexión. El servicio web de los dispositivos se implantará dentro del servidor de aplicaciones de la empresa ya que es quien tiene acceso al servidor de base de datos a través del firewall.

La empresa además cuenta con distintas delegaciones pero estas se encuentran unidas a la red principal a través de un vínculo VPN. Por tanto y en lo que cabe para el alcance de este proyecto consideraremos que dichas delegaciones se hallan dentro de la red principal y cada dispositivo se comportará de forma similar ya esté situado en la central o en una delegación.



3-3 Diagrama de Red

3.4.2 DISEÑO DE LA APLICACIÓN .NET CF

3.4.2.1 Lecturas y Lotes de lecturas

Definimos una Lectura como una n-tupla de valores entre los que hallamos:

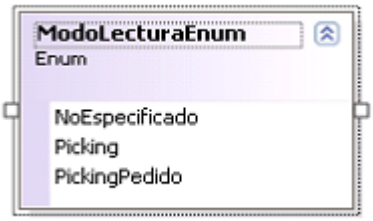
- Identificativo de Lectura
- Identificativo del Lote al que pertenece la lectura
- Código del Artículo Leído
- Cantidad Leída
- Cantidad Pedida
- Otros valores informativos (EANLeído, Stock, Descripción del Artículo)

Se elige como tipo de datos para almacenar el identificativo de la lectura y del lote un tipo *guid.*

Un lote de Lecturas será simplemente un conjunto de lecturas con el mismo identificativo de Lote. La propiedad *IdLectura* de la instancia *SkorpioCommand* almacena el valor *guid* del lote de lectura seleccionado actualmente. Las lecturas pertenecientes al lote en curso serán aquellas que tengan su valor *IdLote* coincidente con la propiedad *IdLectura* de la instancia SkorpioCommand.

3.4.2.2 Modo de lectura

Se debe definir también el modo de lectura actual del dispositivo. Esto se hace en el tipo enumerado *ModoLecturaEnum* definido en el archivo *ModoLecturaEnum.vb* (figura 3-4)



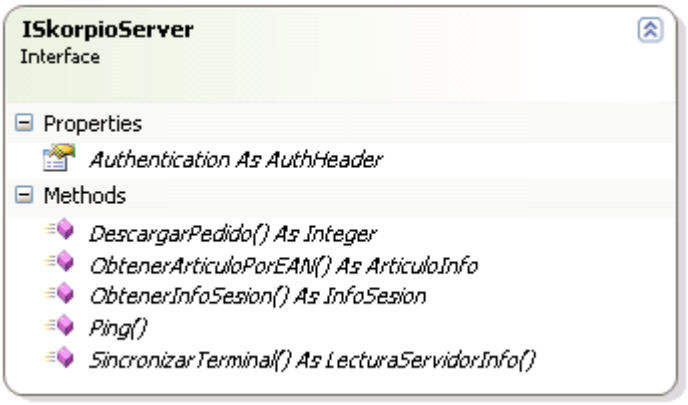
3-4 Enumeración ModoLecturaEnum

En este proyecto se van a detallar sobre todo los modos de lectura *Picking* y *PickingPedido* y por ello es importante diferenciar el flujo de datos en un caso y en otro. Mientras que con el modo de lectura *Picking* el lote de lecturas nace directamente en el dispositivo (ya que no hay un pedido previo), con el modo *PickingPedido* el lote de lecturas nace en el ERP, a través de un pedido de un cliente. Este lote de lecturas nace con valores en las cantidades pedidas y cero en las cantidades leídas y se recibirá en el dispositivo cuando el operador lo solicite con el comando adecuado.

El emplear códigos de tipo *guid* para los lotes de lecturas nos permite que no haya conflictos entre ellos independientemente de si se generan en el servidor o en el dispositivo.

3.4.2.3 Cliente del Servicio Web

En el archivo *ISkorpioServer.vb* se define el interfaz *ISkorpioServer*. Este interfaz debe ser implementado por aquella clase que haga de cliente del servicio web. Puede verse en la figura 3-5 un esquema de dicho interfaz.



3-5 Interfaz ISkorpioServer

La propiedad *Authentication* contiene una instancia del tipo *AuthHeader* que a su vez contiene las propiedades *IdTerminal, IdOperador* y *Password.* Estos tres datos representan las credenciales con las que el dispositivo de mano va a presentarse ante el servicio web cada vez que llame a cualquiera de los métodos disponibles.

El método *ObtenerInfoSesion* devuelve algunos datos de interés para ser mostrados en el dispositivo, tales como el código lógico del almacén o el nombre del operador.

El método *DescargarPedido* solicita al servicio web que descargue un pedido concreto pasado como parámetro desde el ERP a la tabla de lecturas del servidor, preparándolo así para su envío al dispositivo de mano.

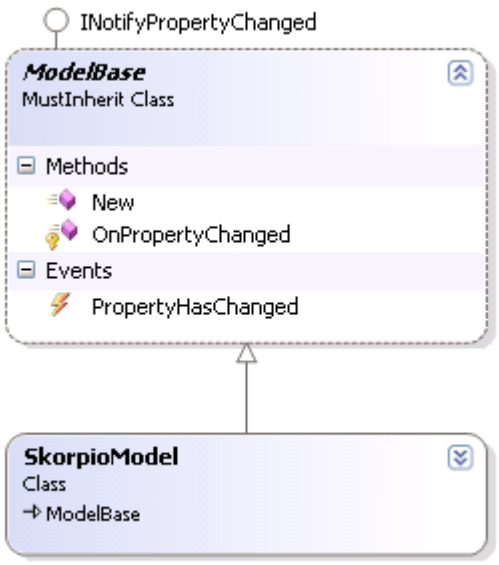
El método *SincronizarTerminal* tiene la doble función de enviar al servidor las lecturas finalizadas y de recibir del servidor los lotes de pedidos pendientes de leer. Por lo general la sincronización se produce siempre en un único sentido (envío o recepción) sin embargo el método soporta ambos sentidos de forma simultánea.

El método *Ping* se utiliza para producir una llamada al servicio web sin efecto alguno, sin embargo resulta útil para inicializar el servicio web si este está en reposo o para evitar que precisamente entre en reposo si no vamos a realizar llamadas al servicio durante un largo tiempo.

3.4.2.4 Modelo

Se crea una clase denominada *SkorpioModel*, que a su vez hereda de la clase abstracta *ModelBase* la cual implementa notificación de cambio de propiedades. La clase *SkorpioModel* representa un modelo de datos de un dispositivo de lectura y contiene propiedades que se van a enlazar con la parte visual.

El esquema 3-6 muestra la clase *SkorpioModel*.



3-6 Modelo

Las propiedades más relevantes de esta clase se detallan a continuación:

- **AltavozActivado (Boolean)**: Si tiene valor *true* se produce respuesta a través del altavoz del dispositivo
- **DescripcionDocumento (String)**: Muestra información sobre el documento en curso.
- **OperadorAsignado (Integer)**: Código lógico del operador que está utilizando el dispositivo.
- **DescripcionOperador (String)**: Muestra información sobre el operador que está utilizando el dispositivo
- **DescripcionTerminal (String)**: Muestra información sobre el propio dispositivo
- **IdLectura (Guid)**: Identificativo del lote de lecturas en curso.
- **IdLecturaContado (Guid)**: Identificativo del lote de lecturas para el modo de lectura *picking*
- **IdTerminal (String):** Devuelve el número de serie del dispositivo
- **Log (String)**: Contiene el histórico de mensajes producidos por la aplicación
- **ModoLectura (ModoLecturaEnum)**: Indica el modo de lectura activo
- **ServerProxyFactory**: Contiene una expresión lambda que de ser invocada devuelve una clase que implemente el interfaz *ISkorpioServer* tal y como se ha detallado en el punto 3.4.2.3. Esta instancia se utiliza para invocar el servicio web. Esta propiedad tiene de forma predeterminada el valor de una expresión lambda que al ser invocada llama al método *SkorpioWSFactory* del módulo *DefaultWebService*. Este método va a seguir un patrón de *Lazy Load*, es decir se creará la instancia de la clase *SkorpioServer.SkorpioWS* en la primera llamada y será esta instancia la que se devuelva en sucesivas llamadas.
- **TieneWifi (Boolean)**: Si tiene valor *true* significa que el dispositivo se encuentra en una zona donde hay disponible cobertura de red inalámbrica.
- **UltimaLectura (Guid):** Almacena el identificativo de la última lectura realizada

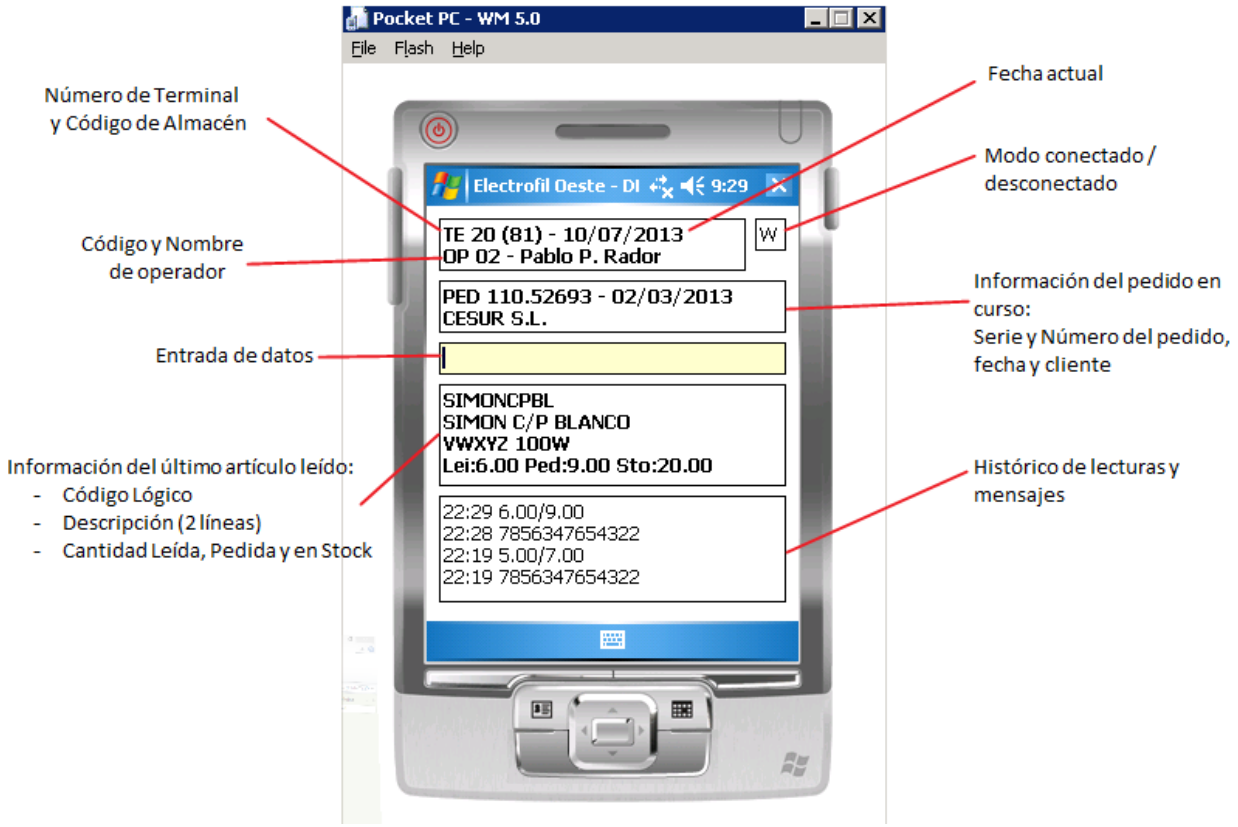
Los métodos más relevantes de esta clase se detallan a continuación:

- **AppendLog:** Anexa a la propiedad *Log* el texto especificado como parámetro

- **BeepError:** Si la propiedad *AltavozActivado* tiene valor *true*, genera una serie de varios pitidos consecutivos.
- **BeepOk:** Si la propiedad *AltavozActivado* tiene valor *true*, genera un pitido.
- **GetServerProxy:** Este método invoca la expresión lambda almacenada en la propiedad *ServerProxyFactory* devolviendo por tanto una instancia que implemente el tipo *ISkorpioServer*,

3.4.2.5 Formulario Principal

Teniendo en cuenta los puntos discutidos en 3.4.1.3 se utiliza el diseñador de Windows Forms para dibujar la pantalla principal de la aplicación, la cual se almacena en el archivo **MainForm.vb.** La representación visual de dicho archivo se muestra en la imagen 3-7.



3-7 Interfaz principal de la aplicación

- **Número de Terminal:** Se muestra el código lógico asociado al dispositivo de mano.
- **Código de Almacén:** Muestra un código lógico asociado al almacén donde está trabajando el dispositivo
- **Fecha Actual**: Muestra la fecha del día en curso.
- **Código y Nombre del operador:** Descripción del operador que está utilizando el dispositivo.

- **Información del pedido en curso:** Muestra datos de interés de un pedido que se está sirviendo, como el número de pedido, la fecha del pedido y el nombre del cliente.
- **Entrada de datos:** Este es el único campo donde se hará lectura de comandos por el usuario, normalmente códigos de barras.
- **Información del último artículo leído:** Tras realizar una lectura de un artículo, aquí aparecerá el código lógico del mismo, su descripción y las cantidades leídas acumuladas, el total de unidades que hay que servir, y la cantidad en stock actualmente.
- **Histórico de lecturas y mensajes:** En este recuadro se muestra un histórico de los comandos de entrada y los mensajes de salida

Para crear los enlaces entre los componentes visuales y el modelo de datos se agrega un componente de tipo *BindingSource* a dicha pantalla, indicando el valor *SkorpioModel* en su propiedad *DataSource.* Seguidamente se enlazan individualmente cada campo dibujados en el formulario con su respectiva propiedad de la clase *SkorpioModel*:

- La caja de texto *OperadorTextbox* se enlaza con la propiedad *DescripcionOperador*
- La caja de texto DescripcionDocumentoTextbox se enlaza con la propiedad DescripcionDocumento
- La caja de texto DescripcionArticuloTextbox se enlaza con la propiedad DescripcionArticulo
- La caja de texto *LogTextbox* se enlaza con la propiedad *Log*

Señalar que las cajas de texto *WifiInfoTextbox* y *LecturaTextbox* no están enlazadas a datos. En el primer caso se va a actualizar por código mientras que en el segundo caso no es necesario este enlace ya que sólo se va a usar para recabar datos.

3.4.2.6 Procesado de códigos EAN

El procesado del campo de entrada de datos lo realiza el método privado *ProcesaCodigoEAN*. Este método básicamente se limita a analizar el parámetro de entrada que corresponde al código leído y trata de identificarlo con alguna de las constantes de códigos de barras que representan comandos de la aplicación tal y como se definieron en 3.4.1.2. Si se identifica correctamente alguno de estos códigos se crea el comando correspondiente y se ejecuta, indicando los parámetros si fuera necesario.

Si el código comienza por alguno de los prefijos de comandos variables: *298* para selección del operador o *291* para selección de pedido se analiza los números sucesivos descodificando el código del operador o la serie y número del pedido respectivamente. Una vez descodificado se utilizan estos datos como argumentos a la hora de invocar el comando correspondiente.

Señalar que también se admiten algunas variaciones para seleccionar un comando, por ejemplo “contado” en lugar de “2970000000004”. Esto simplifica las pruebas en el emulador.

3.4.2.7 Comprobación de la conectividad

El modelo *SkorpioModel* define una propiedad *TieneWifi* que consultan diversos comandos para determinar si existe cobertura inalámbrica o no. Para que esta propiedad tenga un valor correcto al iniciar la aplicación se inicia también un hilo donde se verifica

la conectividad por una técnica de muestreo, es decir se verifica cada vez que pasa un intervalo de tiempo concreto.

El intervalo de tiempo de muestreo se define en el fichero de configuración con la etiqueta *CheckWifi*.

La comprobación de conectividad se realiza de forma muy sencilla utilizando una librería propia del SDK del dispositivo Skorpio. Concretamente hay que instanciar una clase *datalogic.wireless.OutOfRange* y verificar su propiedad *isAssociated* tal y como se muestra en el siguiente código:

```
Private _CheckWifiThread As System.Threading.Thread = Nothing
Private _OutOfRange As datalogic.wireless.OutOfRange = Nothing

'Inicializa el hilo de comprobación de Wifi
Private Sub CheckWifi()
  If Me.ModoEmulador Then
    Me.TieneWifi = True
  Else
    _OutOfRange = New datalogic.wireless.OutOfRange

    'Obtener valor de polling del fichero de configuración, predeterminado a 5"
    Dim Int = 5000
    Dim cw = MobileConfiguration.Settings("CheckWifi")
    If IsNumeric(cw) Then Int = CInt(cw)

    While True
      System.Threading.Thread.Sleep(Int)
      Me.TieneWifi = _OutOfRange.IsAssociated
    End While
  End If
 End Sub
```

3.4.2.8 Pantalla de Resumen

Un comando permite al operador abrir una pantalla emergente *ResumenForm.vb*. Esta pantalla únicamente muestra un texto donde se resumen todas las cantidades servidas y todas las cantidades pendientes de servir del lote de lectura en curso.

Esta consulta se realiza en el código usando Linq que nos permite fácilmente filtrar el conjunto de datos, ordenarlo, etc.

3.4.2.9 Pantalla de entrada de artículo desconocido

En algunos casos el operador tendrá que enfrentarse a la identificación de un artículo que la aplicación no ha podido identificar, normalmente porque el EAN no se encuentra codificado en el ERP de la empresa.

En esta situación al operador le aparecerá la pantalla *InputForm.vb* informándole del problema y le solicitará un texto descriptivo o un comentario para que al realizar el

proceso de integración sea más fácil identificar el artículo. El no encontrar el EAN en la base de datos no impide proseguir con el pedido en curso pero el dispositivo de mano no podrá en este caso aportar información sobre el artículo tal como la descripción o el stock disponible en el almacén.

La pantalla utilizada para recabar esta información se muestra en la figura 3-8.

3-8 Entrada de Artículo desconocido

3.4.2.10 Comandos

Se ha decidido estructurar la funcionalidad de la aplicación mediante el patrón de diseño *Command* y *Composite*. Esto nos permite encapsular la funcionalidad requerida por la aplicación en clases discretas que no tienen relación entre sí. Esto va a facilitar ciertas tareas como por ejemplo invocar comandos independientemente de la forma que se haga (a través de una tecla, de un código de barras, de un botón, por código…). Otra ventaja de estructurar la aplicación en comandos es que facilita la realización de programas de pruebas pues ante una entrada determinada al ejecutar un comando podemos esperar una salida concreta.

Se puede encontrar una explicación más detallada de los patrones de diseño *Command* y *Composite* en el anexo E. La aplicación implementa este patrón en las clases *CommandBase, CommandInvoker, SkorpioCommand* y *CompositeSkorpioCommand.*

- **Clase CommandBase**

La clase *CommandBase* es una clase abstracta que únicamente cuenta con el método *Execute*, método que se debe sobrescribir en la clase concreta que herede de ella desarrollando aquí las acciones que va a efectuar este comando. Este método será invocado por una clase *CommandInvoker*.

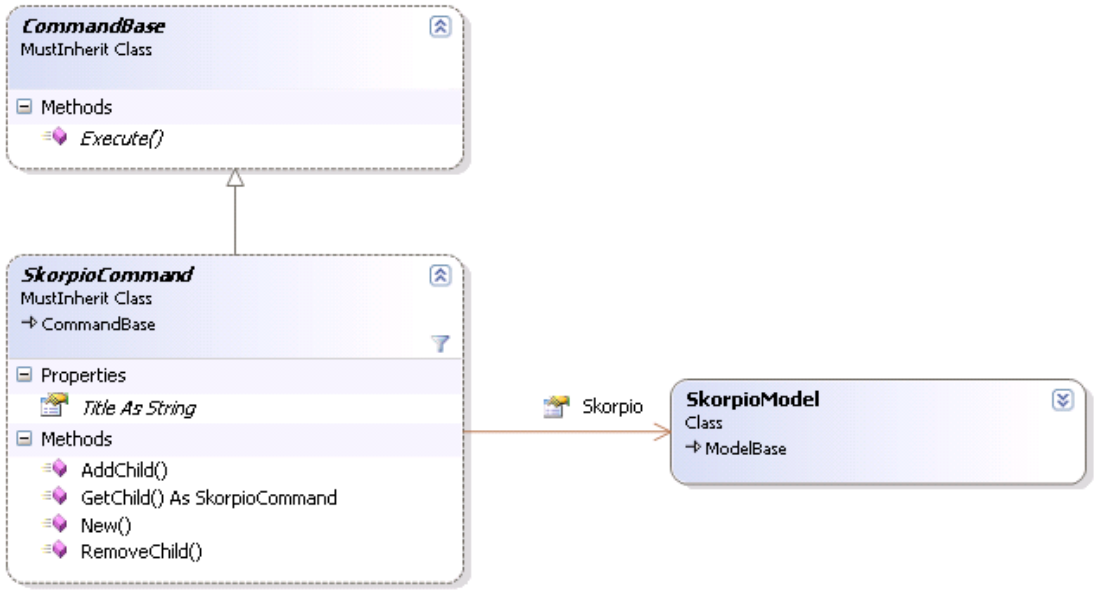
- **Clase CommandInvoker**

La clase *CommandInvoker* contiene un método Execute la cual se limita a ejecutar el comando especificado por el parámetro. Una clase invoker podría realizar tareas como acumular los comandos ejecutados (por ejemplo para poder deshacerlos posteriormente) o puede actuar de buffer (por ejemplo si se solicita la ejecución de comandos cuando el receptor aún no está listo).

- **Clase SkorpioCommand**

En el archivo *SkorpioCommand.vb* se define la clase abstracta *SkorpioCommand* que a su vez hereda de la clase también abstracta *CommandBase.* La clase *SkorpioCommand* agrega una propiedad *SkorpioModel* que va a contener la instancia del modelo sobre la que se van a ejecutar los comandos. Otra propiedad es *Title*, una cadena descriptiva del comando en sí. Los métodos *AddChild, RemoveChild* y *GetChild* se implementan vacíos, y sirven para dar soporte al patrón *Composite* como veremos en el siguiente punto.

La figura 3-9 muestra un esquema de esta clase.



3-9 Implementación del patrón Command

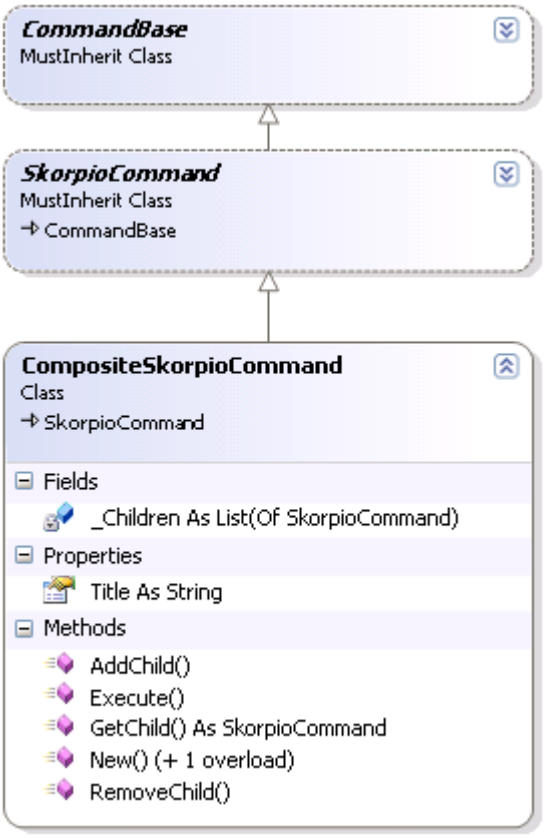
- **CompositeSkorpioCommand**

La clase *CompositeSkorpioCommand* está diseñada para soportar el patrón *Composite*. De esta forma comandos más complejos pueden ser compuestos a partir de otros más sencillos.

Esta clase contiene una variable interna *_Children* de tipo *List(Of SkorpioCommand).* Esta lista acumula una colección de instancias de *SkorpioCommand.*

A través de los métodos *AddChild* y *RemoveChild* se pueden agregar y eliminar comandos a esta lista. Seguidamente el método *Execute* se sobrescribe para recorrerse la lista interna de comandos y ejecutarlos secuencialmente.

La figura 3-10 muestra un esquema de esta clase.



3-10 Implementación del patrón Composite

- **Comandos implementados para la aplicación**

A continuación vamos a detallar los comandos que se han implementado para la aplicación. Especificamos en cada caso la intención del comando, los parámetros esperados por el mismo y el detalle de su ejecución. En el caso de los parámetros se omitirá el primero de ellos que será siempre el Receptor del comando, de tipo *SkorpioModel*.

Se pueden omitir del detalle ciertas explicaciones más o menos triviales como puede ser validación de los parámetros y mensajes de información para el usuario.

3.4.2.10.1 AsignarOperadorCommand

- Objetivo

El objetivo de este comando es modificar el operador que está utilizando actualmente el dispositivo.

- Parámetros

CodigoOperador (Integer): El código lógico de operador que se desea asignar a la instancia *SkorpioCommand*

- Detalle

Este comando simplemente asigna a la propiedad *OperadorAsignado* de la instancia *SkorpioCommand* el código lógico de operador especificado como parámetro.

A continuación se hace una solicitud al método *ObtenerInfoSesion* del servicio web. Este método devuelve a nivel informativo (si se obtienen en el servidor) los datos de nombre del operador, código lógico de terminal y código lógico de almacén.

Con estos datos se rellenan las propiedades *NombreOperador* y *DescripcionTerminal* de la instancia *SkorpioCommand*. Ambas propiedades están enlazadas a campos dibujados en la pantalla principal de la aplicación y por tanto el usuario puede visualmente confirmar su nombre de operador, código lógico de terminal y de almacén.

3.4.2.10.2 MostrarInformacionCommand.vb

- Objetivo

El objetivo de este comando es mostrar información de identificación del dispositivo (como el número de serie) en la ventana de depuración

- Parámetros

Ninguno

- Detalle

Este comando se limita a anexar información diversa en el log usando el método *AppendLog* de la instancia *SkorpioCommand*, como los valores de *IdTerminal* (número de serie físico del terminal) y *DescripcionTerminal* (código lógico de terminal y código lógico del almacén)

3.4.2.10.3 PingCommand.vb

- Objetivo

Este comando sirve para levantar o mantener vivo al servicio web si es necesario

- Parámetros

Ninguno

- Detalle

La implementación de este comando se limita a invocar el método *Ping* del servicio web. Este método no hace nada pero el invocar este comando hace que el servicio web se inicialice si no lo estuviera.

3.4.2.10.4 CambiarModoLecturaCommand.vb

- Objetivo

Este comando permite alternar entre los distintos modos de lectura, tales como *PickingPedido* y *Picking*

- Parámetros

ModoLectura (*ModoLecturaEnum)*: Parámetro que indica el modo de lectura deseado.

- Detalle

El comando asigna a la propiedad *ModoLectura* de la instancia *SkorpioCommand* el valor del parámetro *ModoLectura*, cambiando así efectivamente el modo de lectura que está utilizando el operador. Adicionalmente se ponen a valores predeterminados algunos

valores de la instancia *SkorpioCommand*, tales como *UltimaLectura, DescripcionArticulo, SerieDocumento, NumeroDocumento, FechaDocumento, ClienteDocumento*.

Si el modo de lectura seleccionado es de tipo *Picking* entonces se verifica si es la primera vez que se selecciona este modo para crear un nuevo registro en la tabla Lotes, dado que se trata de un lote que se crea en el dispositivo (a diferencia de un lote de un pedido). Se generaría entonces un nuevo guid que se almacenaría en la propiedad *IdLectura*, suficiente para permitir ya la lectura de artículos.

3.4.2.10.5 AsignarPedidoCommand.vb

- Objetivo

Selecciona un pedido para posteriormente trabajar con él.

- Parámetros

Serie (*Integer*), Numero *(Integer)*: La Serie y el Número del pedido que se desea seleccionar

- Detalle

Este comando selecciona el valor *PickingPedido* en la propiedad *ModoLectura* de la instancia *SkorpioCommand.* También asigna los parámetros de entrada a las propiedades *SerieDocumento* y *NumeroDocumento* respectivamente.

A continuación se busca en la tabla de Lotes si ya existe un pedido con la serie y número seleccionados. Si se encuentra se recupera de dicha tabla el identificativo de lectura que se asignará a la propiedad *IdLectura*, además de otros datos de interés (fecha del pedido y cliente del pedido) que se mostrarán en la pantalla de la aplicación.

En caso de no encontrar el pedido en la tabla de lotes (normal si se lee por primera vez) lo se informa al usuario en el registro de que el pedido no está descargado.

3.4.2.10.6 PrepararPedidoCommand.vb

- Objetivo

Indica al servidor que debe preparar el pedido seleccionado para su descarga en el dispositivo en la próxima sincronización.

- Parámetros

Ninguno

- Detalle

En primer lugar hay que validar si la propiedad *ModoLectura* tiene valor *PickingPedido* terminando el comando en caso contrario, ya que este es el único modo de lectura en el que tiene validez este comando.

A continuación se invoca el método *DescargarPedido* del servicio web indicando como parámetros del mismo la Serie y Numero del pedido a descargar, los cuales vamos a tomar de las respectivas propiedades *SeriePedido* y *NumeroPedido* de la instancia *SkorpioCommand.* Existe un tercer parámetro boolean que fuerza la preparación del pedido aunque ya esté preparado en el servidor para ser enviado

El valor de retorno es un integer que nos indica el resultado de la operación según la tabla 3-11.

Valor	Significado
0	El pedido se descargó con éxito por primera vez
1	El pedido ya se había descargado y se sobreescribió
-1	El pedido no existe
-2	El pedido ya está preparado para su descarga
-3	El pedido ya está totalmente servido
-4	El pedido está anulado

3-11 Valores de retorno de DescargarPedido

3.4.2.10.7 SincronizarCommand.vb

- Objetivo

Enviar al servidor las lecturas validadas y recibir del mismo las nuevas lecturas pendientes.

- Parámetros

Ninguno

- Detalle

Este comando llama al método *SincronizarTerminal* del servicio web, enviando como argumento las lecturas realizadas como un array de objetos de tipo *SkorpioServer.LecturaTerminalInfo*, detallado en la figura 3-12.

3-12 Clase LecturaTerminalInfo

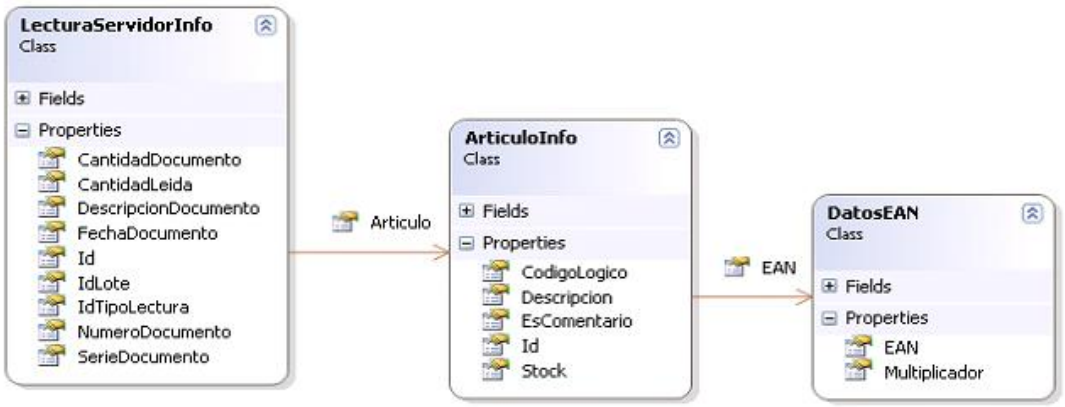
Este array se construye seleccionando con una expresión linq todas las líneas de la tabla de lecturas cuyo campo *PendienteEnvio* tenga valor true y mapeando cada columna relevante de la tabla a una propiedad.

```
'preparar array de datos a enviar
Dim datosPendientes = (From l In datosLecturas _
                                    Where l.PendienteEnvio = True).ToArray

Dim q = (From l In datosPendientes _
    Join Cab In datosCabLecturas On Cab.Id Equals l.IdLote _
    Select New SkorpioServer.LecturaTerminalInfo With { _
      .Id = l.Id, _
      .IdLote = l.IdLote, _
      .CantidadLeida = l.CantidadLeida, _
      .EANLeido = l.EANLeido, _
      .IdTipoLectura = Cab.IdTipo, _
      .SerieDocumento = Cab.SerieDocumento, _
      .NumeroDocumento = Cab.NumeroDocumento, _
      .DescripcionArticulo = l.DescripcionArticulo}).ToArray
```

Las líneas seleccionadas en *datosPendientes* se marcarán con *PendienteEnvio* a False una vez terminado el envío.

A continuación se recibe e interpreta el resultado del método web, que resulta ser un array de tipo *LecturaServidorInfo* como el descrito en la figura 3-13.



3-13 Estructuras devueltas por el servidor

El proceso que se lleva a cabo es tal como sigue:

- Se agrupa el array de datos recibidos por el campo *IdLote* para identificar cada lectura recibida
- Para cada lote de lecturas se realiza lo siguiente:
    - Si el lote ya existiera en la tabla del dispositivo se elimina el lote del dispositivo ya que vamos a actualizarlo con una versión más reciente del servidor.

- Se insertan un registro con los datos de cabecera (*IdLote, IdTipoLectura, SerieDocumento, NumeroDocumento*…) en la tabla de Lotes
- Para cada línea de la agrupación se realiza lo siguiente:
  - Se inserta una línea en la tabla Lecturas con los datos descargados (IdArticulo, Descripcion, CodigoLogico, CantidadDocumento, Stock…)
  - Para cada línea del array *EAN* se inserta en la tabla de EAN un registro con el código EAN, el código de artículo y el multiplicador. Si ya existiera el EAN en la tabla para este artículo se elimina previamente.

3.4.2.10.8 LecturaArticuloCommand.vb

- Objetivo

Resolver la lectura de un código EAN en un código concreto de un artículo

- Parámetros

EANLeido (*String)*: Código EAN leído por el dispositivo

- Detalle

Este comando intenta resolver el código EAN del parámetro en un artículo concreto en varios intentos. Primero intentará resolverlo en la tabla de lecturas para la lectura en curso ya que es probable que el EAN leído por el operador se encuentre ya en esta tabla. Por ejemplo si realizamos una segunda lectura de un mismo artículo, el EAN leído ya se hallará en esta tabla.

En caso de no encontrarlo en esta tabla, se hará una segunda búsqueda para los EAN alternativos de las lecturas en curso, ya que es posible que un mismo artículo tenga varios EAN. Por ejemplo en el caso del modo de lectura *PickingPedido,* el operador ya se habrá descargado en el dispositivo todos los EANs de los artículos incluidos en el pedido.

En caso de no encontrarlo tampoco en la tabla de EAN estaríamos ante el caso de una nueva lectura. El comando creará un registro de la tabla de Lecturas con valores predeterminados y con el código del artículo que sigue siendo necesario resolver. Para resolverlo se invoca una función privada para el comando *ResuelveArticulo* que intentará resolverlo en el servidor remoto.

Si el artículo no se ha hallado a través de la función *ResuelveArticulo* se asumirá que se trata de un artículo nuevo, no catalogado o que no fue posible identificar. En este caso se solicitará al operador un comentario o descripción que le será de utilidad para reconocer el artículo cuando descargue la información de la pistola en el servidor. El código del artículo estará en blanco en este caso y tendremos que resolver el artículo a través del EAN almacenado una vez se descarguen los datos al servidor y generemos un documento TPV.

Finalmente en el registro de Lectura obtenido o creado se incrementarán el valor del campo *CantidadLeida* en tantas unidades como indique el campo multiplicador que viene arrastrado por el EAN (normalmente de valor 1). En caso de que la *CantidadLeida* supere el valor del campo *CantidadDocumento* se dará un aviso al operador ya que estaría superando la cantidad objetivo.

El diagrama de flujo de la figura 3-14 representa este comando.

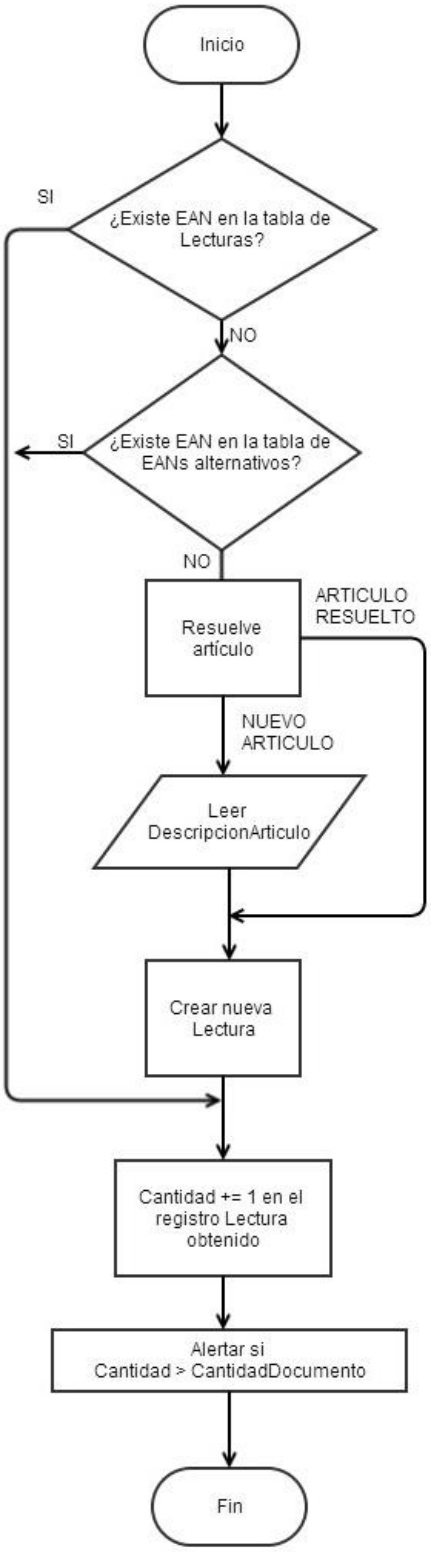
Función ResuelveArticulo:

En esta función se realiza en primer lugar una nueva búsqueda pero para toda la tabla EAN pudiendo ocurrir que estuviera presente el código buscado debido a que se encontró durante una lectura de otro tipo (por ejemplo se resolvió durante una lectura *PickingPedido*).

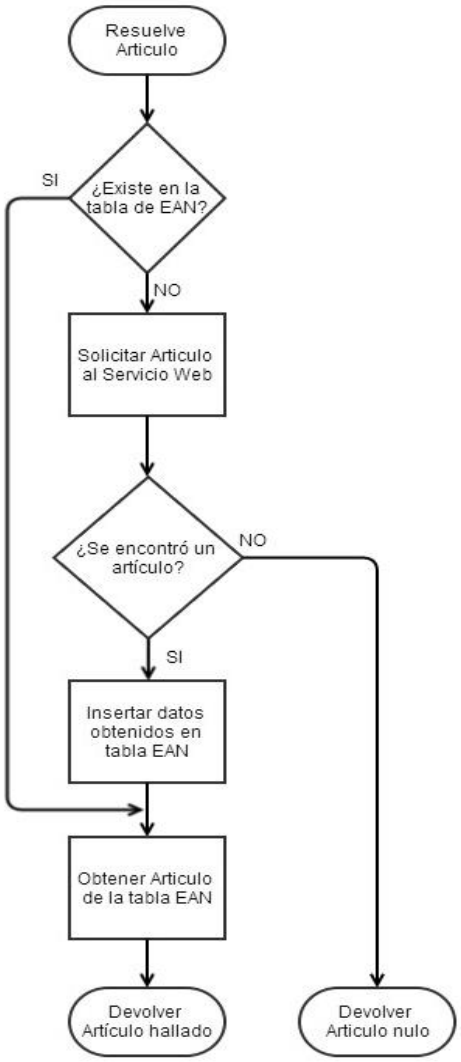
El siguiente paso sería solicitarlo al servicio web (si tenemos conexión). En caso de encontrar el artículo a través de su código EAN el servidor nos enviará toda la información del mismo e incluso el resto de códigos EAN del mismo artículo, para almacenarlos en la tabla EAN.

Finalmente se devolverían los datos del artículo obtenido hallándolos de la tabla de EAN.

El diagrama de flujo de la figura 3-15 representa esta función.



3-14 Diagrama de flujo del comando LecturaArticulo



3-15 Diagrama de flujo de la función ResuelveArtículo

3.4.2.10.9 IncrementarUnidadesCommand.vb

- Objetivo

Incrementar o decrementar una lectura ya realizada una cantidad de unidades determinada

- Parámetros

IdLectura (*guid)*: Identificativo de la lectura a modificar.

Cantidad (*decimal)*: Cantidad (con signo) a incrementar en el registro de lectura. Se asume que la primera unidad ya está incrementada.

- Detalle

En primer lugar el comando localiza el registro de la tabla Lecturas correspondiente al parámetro *IdLectura*. Una vez localizada esta lectura se toma del registro el valor del campo *MultiplicadorLeido*, siendo ésta la cantidad que se va a incrementar en el campo *CantidadLeida* del mismo registro, tantas veces como el parámetro *Cantidad* indique menos uno. Es decir:

```
Dim Multi = fLectura.MultiplicadorLeido
Dim IncCantidad = (Cantidad - 1) * Multi
fLectura.CantidadLeida += IncCantidad
```

Es necesario restar uno al argumento *Cantidad* porque normalmente este comando se ejecutará después de un comando *LecturaArticulo*, el cual ya habrá acumulado una unidad. El efecto es que así el operador puede leer un artículo y a continuación tecleará las unidades finales deseadas en lugar de tener en cuenta la unidad de la primera lectura.

3.4.2.10.10 ValidarDocumentoCommand.vb

- Objetivo

Marcar un lote de lecturas como finalizado y listo para ser enviado al servidor

- Parámetros

Ninguno

- Detalle

Este comando se recorre todas las líneas de la tabla de Lecturas que tengan el campo *IdLote* igual al valor actual de *IdLectura* de la instancia *SkorpioCommand.* A cada línea procesada le asigna el valor *PendienteEnvio* = True, dejando preparada esta línea para ser enviada por el comando SincronizarCommand.

3.4.2.10.11 BorrarDatosCommand.vb

- Objetivo

Eliminar todas las lecturas de un lote de lecturas. También se puede utilizar este comando para eliminar todas las lecturas

- Parámetros

IdLecturaABorrar (*guid*): Código de la lectura a eliminar.

BorrarTodo (*Boolean)*: Si vale True se eliminarán todas las lecturas de la tabla

- Detalle

Este comando selecciona todas las líneas de la tabla de lecturas donde el valor del campo *IdLectura* corresponda con el parámetro *IdLecturaABorrar*. Si el parámetro *BorrarTodo* value True se seleccionarán todas las líneas independientemente del valor de *IdLectura.*

Cada línea procesada es eliminada de la tabla.

De forma similar se eliminan los datos de las tablas EAN y Lotes, bien selectivamente para la lectura indicada como argumento o bien en su totalidad si se solicita borrarlo todo.

Finalmente se inicializan a su valor por defecto algunas propiedades de la instancia *SkorpioCommand*, como *IdLectura, ModoLectura, SerieDocumento* y *NumeroDocumento.* Esto deja preparado el dispositivo para iniciar una nueva lectura.

3.4.2.10.12 AnularLecturaCommand.vb

- Objetivo

Cancelar la última lectura del operador restando para ello la última cantidad leída con el dispositivo

- Parámetros

Ninguno

- Detalle

En primer lugar el comando localiza el registro de la tabla Lecturas correspondiente a la última lectura. Para ello la instancia *SkorpioCommand* tiene una propiedad *UltimaLectura* con el identificador de la misma.

Una vez localizada esta lectura, se toma del registro el valor del campo *MultiplicadorLeido*, siendo esta la cantidad que se va a decrementar del campo *CantidadLeida* del mismo registro.

Finalmente se inicializa el valor de *MultiplicadorLeido* a 0, con lo que este comando no tendría efecto de ser ejecutado una segunda vez consecutiva.

3.4.2.10.13 DescargarPedidoCommand.vb

- Objetivo

Solicita al servidor que prepare un pedido concreto para su descarga, sincroniza el dispositivo y lo selecciona para su lectura.

- Parámetros

Ninguno

- Detalle

Este es un comando compuesto de otros comandos que ya se han detallado. En concreto este comando se limita a ejecutar el comando *PrepararPedidoCommand* (3.4.2.10.6), el comando *SincronizarCommand* (3.4.2.10.7) y finalmente *AsignarPedidoCommand* (3.4.2.10.5).

El efecto que produce es que habiendo seleccionado un pedido este se solicita al servidor y se descarga inmediatamente, dejándolo listo para su subsiguiente lectura.

3.4.2.10.14 CargarPedidoCommand.vb

- Objetivo

Valida la lectura en curso y la sube inmediatamente al servidor. Seguidamente elimina la lectura del dispositivo.

- Parámetros

Ninguno

- Detalle

Este es un comando compuesto de otros comandos que ya se han detallado. En concreto este comando se limita a ejecutar de forma consecutiva los comandos *ValidarDocumentoCommand* (3.4.2.10.10), *SincronizarCommand* (3.4.2.10.7), *BorrarDatosCommand* (3.4.2.10.11) y finalmente *CambiarModoLectura (3.4.2.10.4*).

El efecto que produce es que teniendo seleccionada una lectura en el dispositivo ésta se valida para ser enviada al servidor, a continuación se envía inmediatamente y finalmente se borra del dispositivo.

3.4.2.11 Pruebas automáticas

Siempre es conveniente disponer de pruebas automatizadas de las aplicaciones ya que estas nos aseguran un mínimo de calidad en las mismas. Aunque es relativamente complejo probar al 100% una aplicación siempre es mejor tener algún tipo de prueba que ninguno. En el caso que nos ocupa encontramos que en .NETCF no encontramos frameworks reconocidos de pruebas como nUnit o similares ya que dada la naturaleza embebida del sistema de desarrollo no es de esperar que se realicen aplicaciones muy grandes o complejas.

Dado que se ha orientado la aplicación a la ejecución de comandos podemos considerar crear pequeños programas que realicen pruebas funcionales en base a ejecutar comandos y a verificar la respuesta. Para simplificar las pruebas de la aplicación y aislarla del servicio web de la parte servidora, se puede crear una clase que implemente el interfaz *ISkorpioServer* y sustituirla en el modelo *SkorpioModel*. De esta forma cada vez que se invoque un método web se invocarán los métodos de esta clase los cuales tendrán una implementación de acuerdo con el juego de pruebas que se esté ejecutando en ese momento.

En resumen los pasos a seguir para realizar una prueba serían los siguientes:

- Declarar una instancia de *SkorpioModel*
- Declarar una instancia de *CommandInvoker*
- Asignar la propiedad *Skorpio.ServerProxyFactory* a una expresión lambda que devuelva una instancia de la clase falsa *ISkorpioServer*.
- Instanciar un comando y ejecutarlo con el Invoker
- Analizar resultados

A continuación se muestra un ejemplo en código del esquema anterior:

```
Dim Skorpio As New SkorpioModel
Dim Invoker As New CommandInvoker

'Sustituir Servicio Web
Skorpio.ServerProxyFactory = Function() New TestLecturaMultiple_ServerMockup

'Crear e invocar comando de asignar operador
Dim Ope = 3
Dim cmd1 As As New AsignarOperadorCommand(Skorpio, Ope)
Invoker.ExecuteCommand(cmd1)
'Verificar que el código del operador es correcto
IsTrue(Skorpio.OperadorAsignado = Ope)

'Crear e invocar comando de asignar el pedido 200.1
Dim cmd2 As New AsignarPedidoCommand(Skorpio, 200, 1)
Invoker.ExecuteCommand(cmd2)
'Verificar que el pedido se asignó correctamente
IsTrue(Skorpio.SerieDocumento = 200)
IsTrue(Skorpio.NumeroDocumento = 1)

'Descargar el pedido
Dim cmd3 As New DescargarPedidoCommand(Skorpio)
Invoker.ExecuteCommand(cmd3)
```

3.4.2.12 Otras clases de utilidad

Como se ha visto en 2.4.3, la BCL (*Base Class Library)* del .NET Compact Framework carece de algunas clases con las que se está acostumbrado a trabajar en el

motor de ejecución .NET completo. Afortunadamente la extensibilidad del lenguaje nos permite reimplementar estas clases de utilidad para trabajar con ellas más cómodamente.

3.4.2.12.1 Clase MobileConfiguration

Esta clase nos permite trabajar de forma sencilla con un fichero de configuración XML *app.config*. Esta clase procesa el contenido de dicho fichero y acumula en una colección interna los pares clave-valor hallados dentro del nodo <appSettings>

Un ejemplo de archivo de configuración se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="Emulador" value="1"/>
                <add key="CheckWifi" value="5000"/>
        </appSettings>
</configuration>
```

En el ejemplo anterior se podría utilizar la clase MobileConfiguration para recuperar el valor cuya clave de configuración es "CheckWifi" de esta forma:

```
Dim cw = MobileConfiguration.Settings("CheckWifi")
```

Destacar que se ha implementado el soporte en esta clase para aplicaciones multihilo utilizando la palabra reservada de VB.NET *SyncLock*.

3.4.2.12.2 Clase InputContext

Esta clase anula el desplegable de autocompletar de una caja de texto que por defecto incluye .NET CF. Aunque este desplegable es útil para la mayoría de situaciones la experiencia demuestra que es molesta para la finalidad de la aplicación y es necesario anularla. Al no tener parámetros accesibles por el programador para ello, esta clase la anula a través de llamadas directas al sistema (P/Invoke)

3.4.3 DISEÑO DE LA BASE DE DATOS SQL SEVER CE EMBEBIDA

La base de datos del dispositivo es muy sencilla, consta de tan sólo tres tablas y sus respectivas claves primarias y foráneas. Los tipos de datos disponibles en Sql Server CE varían respecto otras bases de datos (incluyendo Sql Server) y aunque como hemos visto en 2.5.2 están algo limitados, son suficientes para nuestros propósitos.

La base de datos se ha diseñado en el propio editor Visual Studio 2008, agregando un archivo *LocalDB.sdf* y usando la ventana *Server Explorer* para agregar tablas y definir sus campos usando el diseñador visual.

3.4.3.1 Tabla Lotes

Cada registro de esta tabla representa un Lote de lecturas. La clave primaria de esta tabla es el campo *Id* de tipo *uniqueidentifier*, lo que nos permite tanto generar registros directamente en esta tabla como insertarlos de un medio externo asegurándonos de que no habrá colisión en la clave.

El campo *IdTipo* contendrá un valor numérico según el tipo de lectura del registro, los valores posibles se resumen en la figura 3-16.

Valor	Tipo de Lectura
0	No especificado
1	Picking
2	Picking de Pedidos

3-16 Valores de IdTipo

El resto de campos de datos sirven para identificar el documento sobre el que se está efectuando la lectura: *SerieDocumento, NumeroDocumento, FechaDocumento* y *DescripcionDocumento*

3.4.3.2 Tabla Lecturas

Cada registro de esta tabla representa una lectura individual de un artículo.

La clave primaria de esta tabla es un campo *Id* de tipo *uniqueidentifier*. Es importante destacar el campo *IdLote* también de tipo *uniqueidentifier* el cual es una clave foránea que referencia la columna *Id* de la tabla Lotes.

Encontramos en esta tabla campos que contienen información necesaria para identificar el artículo: *IdArticulo*, *DescripcionArticulo*, *EsComentario*, *CodigoLogico* y *Stock.*

Finalmente tenemos una serie de campos que contienen las cantidades leídas y objetivo: *CantidadLeida, CantidadDocumento* así como otros que dejan almacenada información sobre la última lectura: *EanLeido, MultiplicadorLeido, UltimaCantidadLeida.*

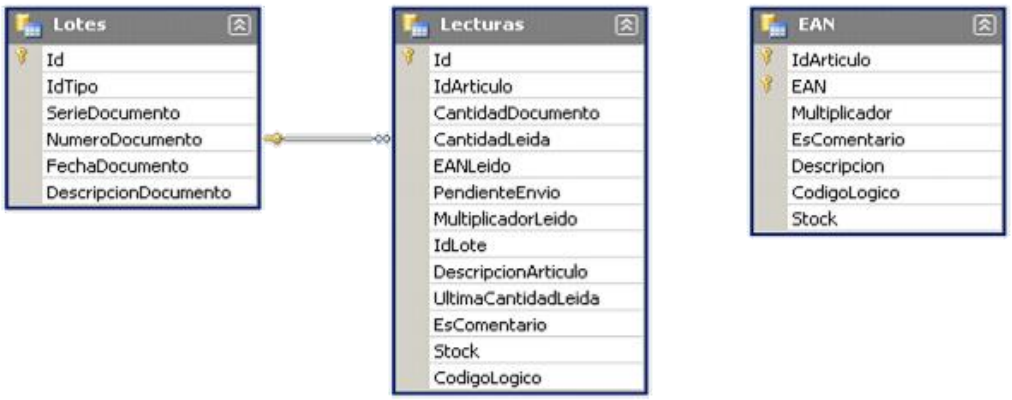
El último campo de interés es un campo binario *PendienteEnvio* que valdrá 1 cuando la lectura esté pendiente de ser enviada al servicio web y 0 en caso contrario

3.4.3.3 Tabla EAN

Esta tabla hace las veces de caché de la tabla completa de artículos y códigos EAN. Cada vez que se descarguen nuevos datos de artículos se almacenarán en esta tabla permitiendo volver a recuperar estos datos de forma más ágil que si se consultara el servicio web en cada ocasión.

La clave primaria de esta tabla está compuesta por dos campos *IdArticulo* y *EAN*, ya que para un mismo código interno de artículo podemos tener varios códigos EAN como ya se ha discutido anteriormente.

Los campos de datos son información del artículo como: Descripcion, CodigoLogico, EsComentario, Multiplicador y Stock.



3-17 Diagrama de la base de datos embebida

3.4.4 DISEÑO DEL SERVICIO WEB

El servicio web que se va a desarrollar necesita acceder a la base de datos del ERP de la empresa. Para ello el servicio web referencia una serie de librerías propias del ERP de la empresa basadas en el framework Csla.NET que nos van a permitir acceder a los datos necesarios para el funcionamiento de la aplicación en la parte servidora. En particular estas librerías nos dan acceso a tablas de datos del ERP como la de pedidos de cliente, la de artículos y códigos EAN.

Físicamente el servicio web se ejecutará en un servidor web IIS que estará escuchando las peticiones de todos los dispositivos. Este servidor a su vez tiene acceso directo a la base de datos de la empresa pero sin embargo no va a ser necesario interactuar directamente con comandos SQL con este servidor. Las librerías del ERP nos permiten interactuar con los datos a alto nivel utilizando entidades y objetos de negocio que representan los datos. Lo único que es necesario realizar en el código antes de utilizar estas librerías sería iniciar sesión en el ERP como un usuario más.

El grueso del servicio web se concentra en el archivo SkorpioWS.asmx, en la clase *SkorpioWS* que hereda de la clase de la BCL *System.Web.Services.WebService*. Esta clase además se decora con el atributo *<WebService>* con lo cual la plataforma ASP.NET realiza el trabajo por nosotros de generar una especificación WSDL y de publicar este servicio web. Dentro de esta clase encontraremos los métodos que van a invocar los dispositivos los cuales se decorarán de forma similar, en este caso con el atributo *<WebMethod>*.

3.4.4.1 Cabecera SOAP Authentication

Se define una clase *Authentication* que se va a mandar como cabecera SOAP. Esta clase contiene dos propiedades *IdTerminal* e *IdOperador* que van a identificar respectivamente el número de serie del terminal y el código lógico del operador que está invocando el método web. Basta decorar los métodos web con el atributo *<SoapHeader("Authentication")>* para indicar que se espera esta cabecera. En cada

método invocado se validará que *IdTerminal* e *IdOperador* tengan valores válidos antes de proceder con el método.

3.4.4.2 Método Web Ping

Como ya se ha comentado en 3.4.2.10.3, este método está vacío. Únicamente se invocará para forzar al servicio web a responder y asegurarnos así de que está levantado.

3.4.4.3 Método Web DescargarPedido

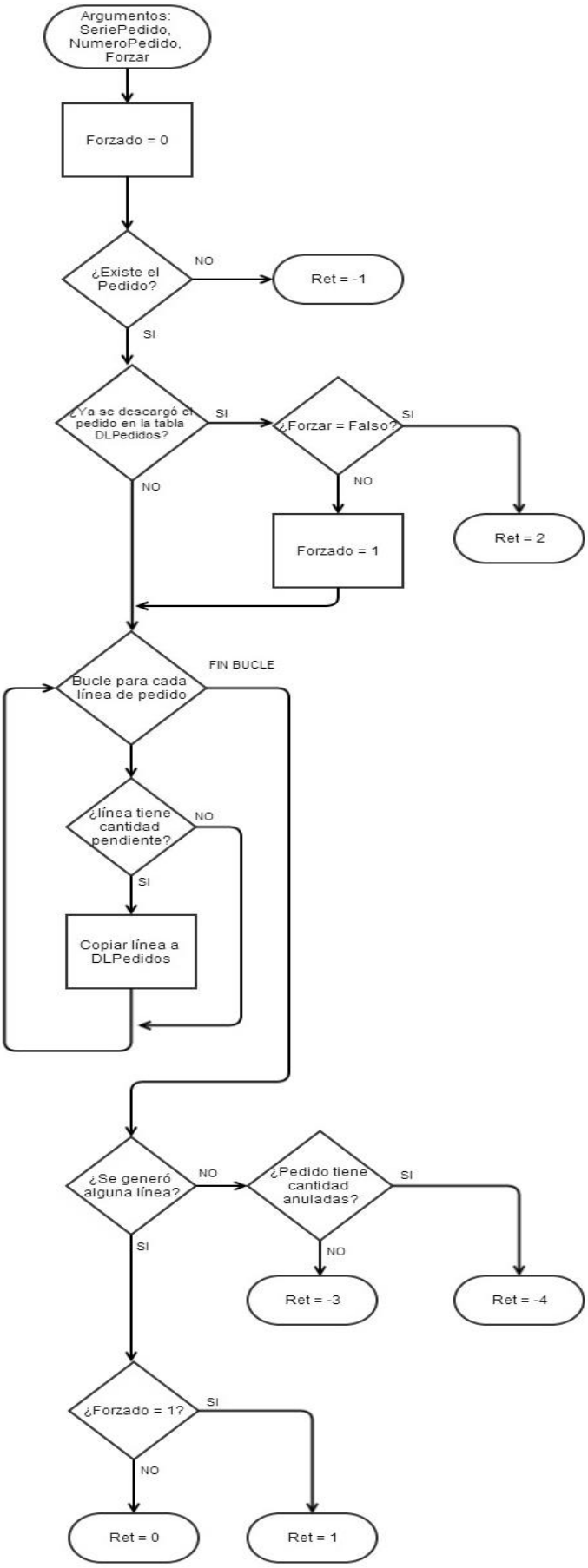
Este método se utiliza para transferir un pedido desde el ERP de la empresa, concretamente se transfieren únicamente las líneas con alguna cantidad pendiente de servir y cualquiera de tipo comentario. El destino de la copia es la tabla DLPedidos, que es la base de sincronización entre el dispositivo y el servidor de base de datos.

El método recibe como parámetros la serie y número del pedido, así como un valor boolean *Forzar*. Este valor indica si se fuerza la descarga en la tabla DLPedidos en caso de que el pedido ya estuviera descargado. El método devolverá un valor entero según la siguiente tabla, donde un valor menor que cero significa un error. La tabla 3-18 muestra un resumen de los posibles valores de retorno.

Resultado	Significado
0	El pedido se descargó con éxito por primera vez
1	El pedido se volvió a descargar con éxito
-1	El pedido solicitado no existe
-2	El pedido está totalmente preparado
-3	El pedido no tiene cantidades pendientes
-4	El pedido no tiene cantidades pendientes y tiene cantidades anuladas

3-18 Valores de retorno posibles del método DescargarPedido

La figura 3-19 muestra un diagrama de flujo de este método web.



3-19 Diagrama de flujo del método DescargarPedido

3.4.4.4 Método Web ObtenerArticuloPorEAN

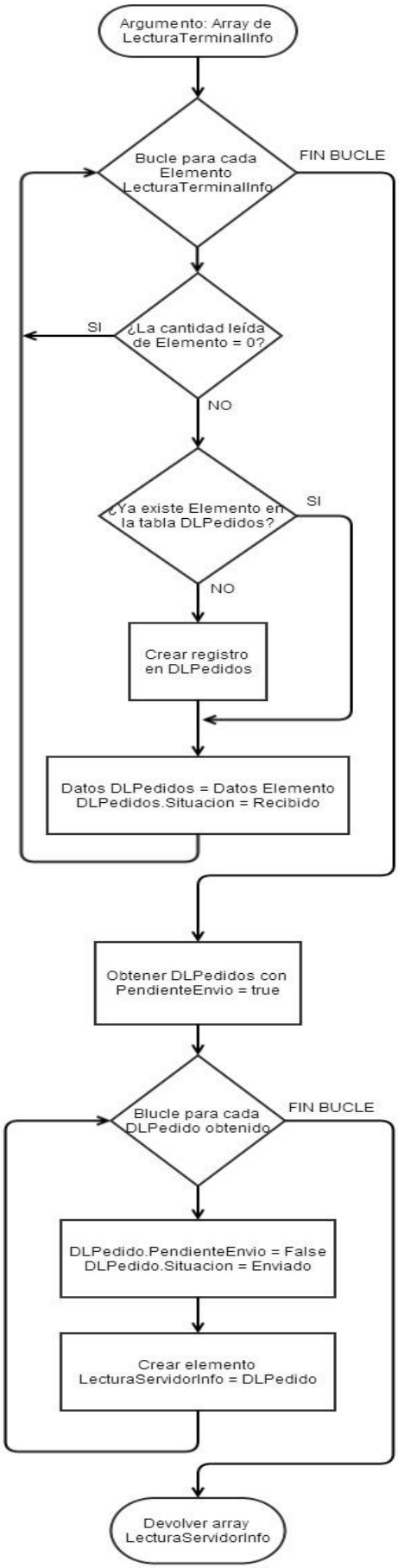
Este método devuelve información de un artículo desde el ERP de la empresa a partir del código EAN especificado como parámetro. Utilizando los objetos de negocios del ERP se localiza el artículo a partir del EAN y se devuelve una estructura de tipo *ArticuloInfo*.

3.4.4.5 Método Web SincronizarTerminal

Este método recibe un array de estructuras de tipo *LecturaTerminalInfo* conteniendo las lecturas del operador. Estos datos se guardarán en la tabla DLPedidos creando o actualizando los registros ya existentes.

De forma análoga se examinan los registros de la tabla *DLPedidos* que cumplan la condición *PendienteEnvio = true* e *IdTerminal* igual al enviado como argumento en la cabecera SOAP *Authentication*. Para cada registro obtenido se construye una estructura *LecturaServidorInfo* con los datos de la tabla *DLPedidos*. El método web modificará el valor del campo *PendienteEnvio* a *false* y finalmente devolverá este array de datos al dispositivo cliente.

La figura 3-20 muestra un diagrama de flujo de este método web.



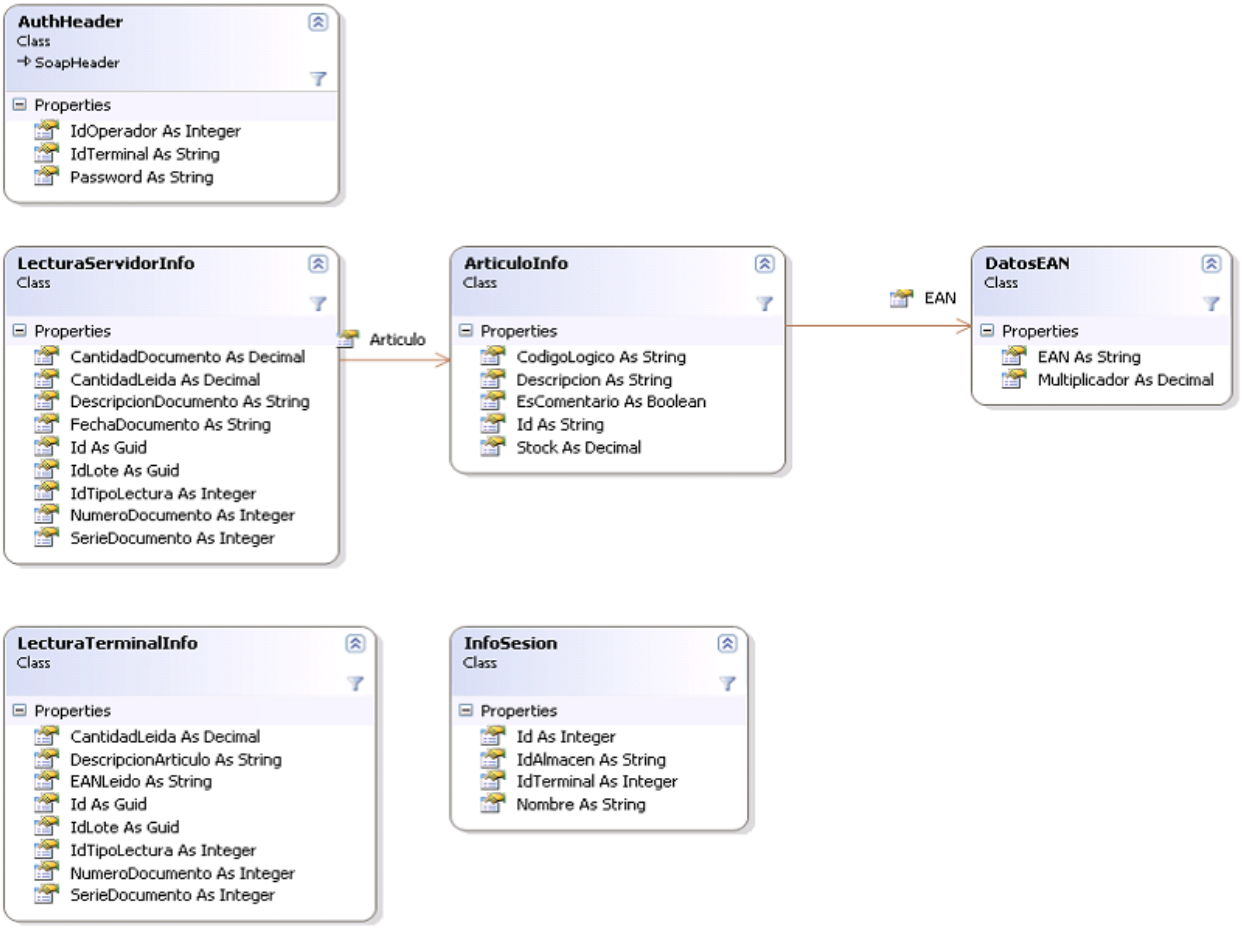
3-20 Diagrama de flujo del método SincronizarTerminal

3.4.4.6 Método Web ObtenerInfoSesion

Este método simplemente devuelve detalles del operador que está usando actualmente el dispositivo. A partir de los datos contenidos en la cabecera SOAP *Authentication* se devuelve una estructura *InfoSesion*. Esta estructura contiene algunos datos de utilidad como el código lógico de almacén, el código lógico del terminal y el nombre descriptivo del operador.

3.4.4.7 Clases de transferencia de datos

A continuación se detallan las clases de transferencia de datos utilizadas en el servicio web. Estas clases no tienen métodos relevantes y se utilizan únicamente para indicar las estructuras de datos a enviar o recibir de los métodos web explicados en los puntos 3.4.4.1 a 3.4.4.6.



3-21 Clases de datos usadas por el Servicio Web

3.4.5 SEGURIDAD EN LAS COMUNICACIONES INALÁMBRICAS

El entorno del almacén en el que se utilizan los dispositivos incluye una red inalámbrica que está ubicada en una zona privada a la empresa (el almacén) e integrada con la red local habitual que utiliza la empresa, por tanto consideramos que en principio

los dispositivos que se conectan son conocidos y es seguro enviar y recibir información de ellos.

Además el servicio web nos hace de mediador con la base de datos por tanto aun consiguiendo un dispositivo un acceso no legítimo a la red de la empresa, el intruso sólo podría invocar los métodos web disponibles y en ningún caso podría acceder directamente al servidor de base de datos. Por estos motivos aquí comentados en principio no deberíamos necesitar excesivas medidas de seguridad.

Sin embargo se sugieren por su simplicidad de implementación dos factores de seguridad a nivel de red como son una clave WPA/WPA2 y/o un filtrado por dirección MAC, ambos configurables en el enrutador inalámbrico.

La implementación de uno o ambos métodos queda a cargo de cada delegación pues dependiendo del caso puede ser más conveniente un método u otro.

3.4.6 CONFIGURACIÓN DEL DISPOSITIVO DE MANO

El dispositivo de mano requiere una configuración inicial para funcionar correctamente en la aplicación pues ésta determina el comportamiento a la hora de interpretar los códigos EAN.

En particular es necesario configurar dentro de la opción del sistema Settings los siguientes parámetros:

- Parameters > Reader Parameters > UpcEan > Ean13 > CheckEvaluation > Enabled
- Parameters > Reader Parameters > UpcEan > Ean13 > CheckTransmission > Enabled

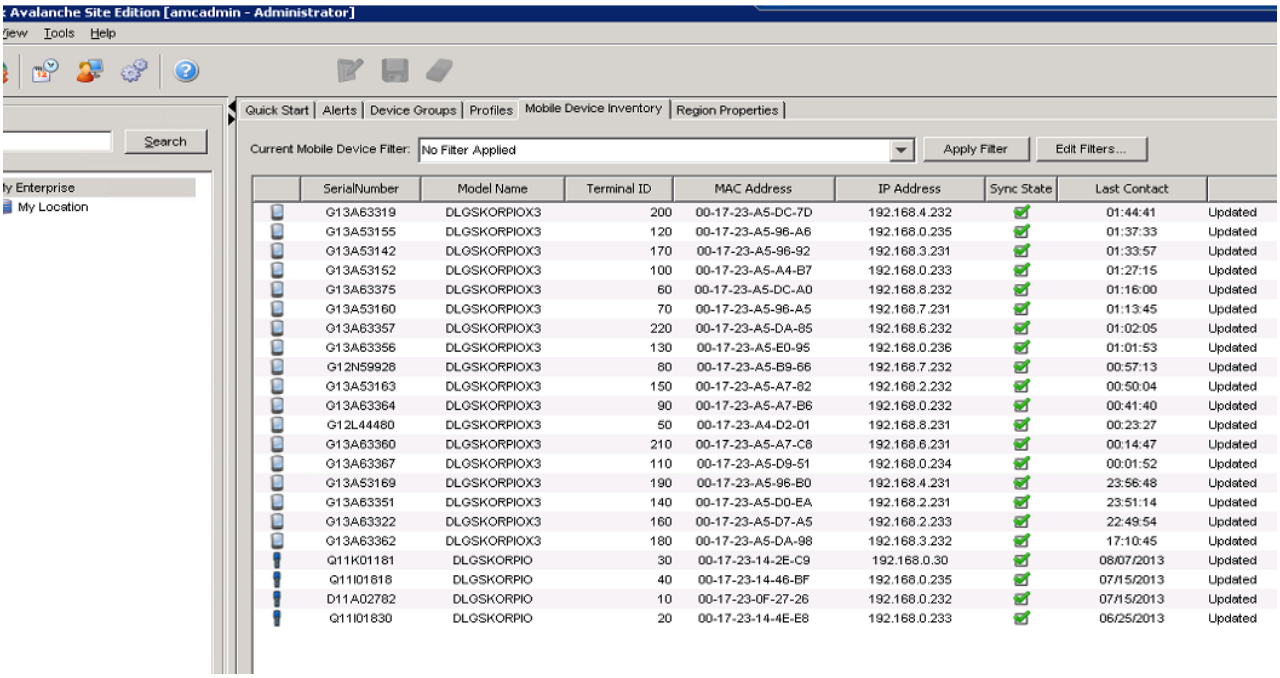
Estos parámetros fuerzan que se valide el dígito de verificación de cada Ean13 leído y además que el propio dígito se vuelque como el 13º dígito tras el código de barras (por defecto el dígito está excluido de la lectura).

3.4.7 DIFUSIÓN DE ACTUALIZACIONES

Dado que la empresa pretende contar con un gran número de dispositivos de mano tanto en la central como en las sucursales, es necesario plantearse cómo distribuir una actualización de la aplicación a cada dispositivo y gestionar dicha actualización. Afortunadamente la propia empresa que proporciona los dispositivos Skorpio nos ofrece una solución software para este fin: Wavelink Avalanche.

El instalar y configurar este software queda fuera del alcance de este proyecto sin embargo sí vamos a comentar los pasos a seguir para difundir una nueva actualización de la aplicación a cada dispositivo de mano para dejar constancia de lo sencillo que resulta difundir cambios.

- Se prepara la nueva versión de la aplicación, para ello se reúnen en una misma carpeta todos los binarios necesarios.
- Iniciar sesión en la aplicación Wavelink Avalanche
- Ir a la pestaña “Profiles”, localizar el paquete (package) anteriormente instalado y quitarlo pulsando el botón “Remove Package”
- Pulsar el botón “Install package” y seguir el asistente. Indicar la ruta de la nueva versión, dar un nombre description y activar el paquete.
- En la pestaña “Mobile Device Inventory” seleccionar uno o varios dispositivos a actualizar y pulsar el botón “Update Now”



3-22 Interfaz del Wavelink Avalanche