Universidad de Lleida
Escuela Politécnica Superior
Ingeniería Técnica en Informática de Gestión

Trabajo de final de carrera

# Sistema de notificación y consulta de alertas de Dynamics AX2012 para aplicación en Windows Phone mediante protocolo push

Autor: Mariano Gracia Carrera
Directora: Marta Oliva Solé
Septiembre 2014

# Contenido

Índice de ilustraciones

# Introducción

La motivación para la creación del proyecto aparece a raíz de la necesidad de la petición de un cliente a IFR (partner oficial del ERP Microsoft Dynamics AX). Este cliente (que ya dispone de AX 2009) solicita poder enviar un SMS desde un puesto de trabajo, en una fábrica de tratamiento de aluminio, a un técnico para avisar de una avería. Este SMS debía ser configurable y se tenía que procesar desde el mismo ERP. Pese a que al final no se llevó a cabo el desarrollo, se decidió aprovechar la idea para una nueva solución utilizando la versión de AX 2012 y las ventajas que éste ofrece.

El desarrollo pretende poder comunicar una alerta a un Smartphone. Esta alerta debe generarse en el ERP Dynamics AX 2012 y una aplicación sobre Windows Phone debe recibirla, avisar al usuario, y permitirle consultarla en detalle: quién la ha generado, dónde se ha producido, qué la ha provocado y marcarla como leída.

Es importante que el desarrollo de esta funcionalidad quede completamente integrada en el ERP, es decir, debe parecer que la nueva funcionalidad no sea un nuevo desarrollo. Para poder alcanzar dicho objetivo se ampliará la funcionalidad estándar de alertas de Dynamics AX 2012. Esta funcionalidad estándar permite a los usuarios configurar alertas que luego se envían a otros .

La alerta puede estar relacionada con cualquier tabla o campo de una tabla del ERP, por ejemplo: se genera una alerta cada vez que se modifica el límite de crédito de un cliente para avisar un comercial, o en el caso del cliente nombrado en la introducción de la memoria, en el cual, se debería haber configurado la alerta de manera que cada vez que se insertara un registro en una tabla de incidencias, se hubiera enviado ésta a un técnico.

Las alertas son notificadas a los usuarios en el propio ERP con un pop-up, o bien mediante correo electrónico, en este proyecto se decide ampliar la funcionalidad con una tercera opción, enviar la alerta a una aplicación de Windows Phone.

La opción escogida para poder enviar la alerta a una aplicación desarrollada para Windows Phone es usando el protocolo push (utilizado en mensajería instantánea).

Para poder enlazar el ERP con la aplicación de Windows Phone se necesita un servicio para las notificaciones mediante protocolo push, en este caso se ha decidido usar el Microsoft Push Notification Service (MPNS).

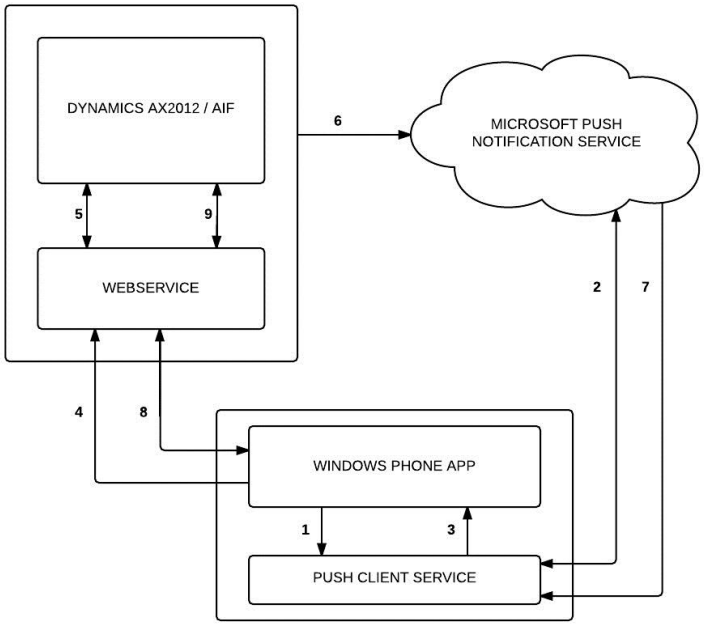
La APP solicita al servicio de MPNS un Uniform Resource Identifier (URI) que actuará como un identificador del terminal móvil. Este URI se ha de notificar al ERP de manera que quede vinculado a un usuario. A partir de este momento el ERP y la aplicación móvil quedan vinculadas. Una vez la alerta se genera en Dynamics AX se envía la notificación, usando el URI notificado previamente, al usuario configurado.

Una vez la alerta llega al Smartphone, el usuario ha de ser capaz de consultar los datos en detalle de la alerta, para poderlos consultar desde la APP (todo esto se puede ver resumido en el diagrama de infraestructura de la Ilustración 1), se ha utilizado el Application Integration

Framework (AIF) incluido en AX Dynamics 2012. Esta funcionalidad permite comunicarse con el ERP desde aplicaciones externas (en este caso la APP).

El AIF permite programar un servicio de Windows que se montará sobre un webservice que será utilizado por la APP para consultar los datos de la alerta.

Para poder llevar a cabo este intercambio de datos se usará un certificado para cifrar las comunicaciones y el usuario de la APP deberá identificarse mediante un usuario y password.




1. La APP solicita un canal push “uniform resource identifier” (URI).
2. El servicio de cliente push negocia con MPNS y éste le devuelve el URI.
3. El servicio de cliente push le devuelve el URI a la APP.
4. La APP se identifica con el webservice para enviarle el URI.
5. El webservice se comunica con Dynamics AX mediante el Application Integration Framework (AIF) para asociar el URI al usuario de la APP.
6. Dynamics genera una alerta para el usuario y envía una notificación push.
7. MPNS envía la notificación a la APP.
8. La APP consulta las alertas identificándose nuevamente contra el webservice.
9. El webservice se comunica con el AIF para la consulta de datos solicitada por el usuario de la APP.

**Ilustración 1. Diagrama de infraestructura**

# 1. Dynamics AX 2012

Dynamics AX 2012 es un sistema de planificación de recursos empresariales, o ERP (Enterprise Resource Planning). No se va a explicar ahora cuál es el funcionamiento de este sistema, pero sí que es necesario aclarar ciertos aspectos que ayudarán a entender mejor las decisiones que se han tomado para el desarrollo del proyecto.
Dynamics AX 2012 es un software de tipo cliente servidor que funciona conjuntamente con SQL Server en un entorno Windows. La mayor carga de procesamiento sucede en el servidor, que se ejecuta como un servicio de Windows que se llama axapta object server (AOS), el número de AOS es variable, este servicio es necesario para que los clientes se puedan ejecutar.
Dynamics AX es muy flexible a la hora de realizar modificaciones y adaptaciones para necesidades no soportadas por el estándar. Éstas se realizan con un lenguaje propio, X++, en un entorno que está incluido dentro del propio cliente de Dynamics AX, aunque X++ soporta llamadas al framework de .NET, esta funcionalidad es limitada.
El acceso a datos es particular en Dynamics AX, ya que se permiten crear objetos tabla que se sincronizan automáticamente en SQL (creando el objeto análogo), una vez creado el objeto se puede usar lenguaje Transact-sql (aunque limitado) en el lenguaje X++ para acceder a datos.

## *1.1. Alertas.*

Para entender correctamente el desarrollo del proyecto y las dificultades que éste presenta hay que entender la funcionalidad de alertas en Dynamics AX y cómo se usaran en la aplicación móvil.
La funcionalidad de las alertas fue integrada por primera vez en la versión de Dynamics AX 4.0 y desde entonces su funcionalidad no ha cambiado significativamente. Éstas permiten al usuario hacer un seguimiento de eventos que se pueden asociar a la modificación de un campo de una tabla en particular, o a la inserción o borrado de un registro. Una vez el evento sucede la alerta se envía al usuario en forma de pop-up en la propia aplicación de Dynamics AX o por correo electrónico.
El usuario puede configurar cada cuantos minutos quiere que se le actualice la lista de alertas.
La creación y configuración de las alertas es sencilla, sobre cualquier formulario al cual tenga acceso el usuario, se puede configurar con tan sólo situándose sobre un campo sobre el cual se desee hacer un seguimiento y seleccionando la opción “Crear regla de alertas…”(Ilustración 2)

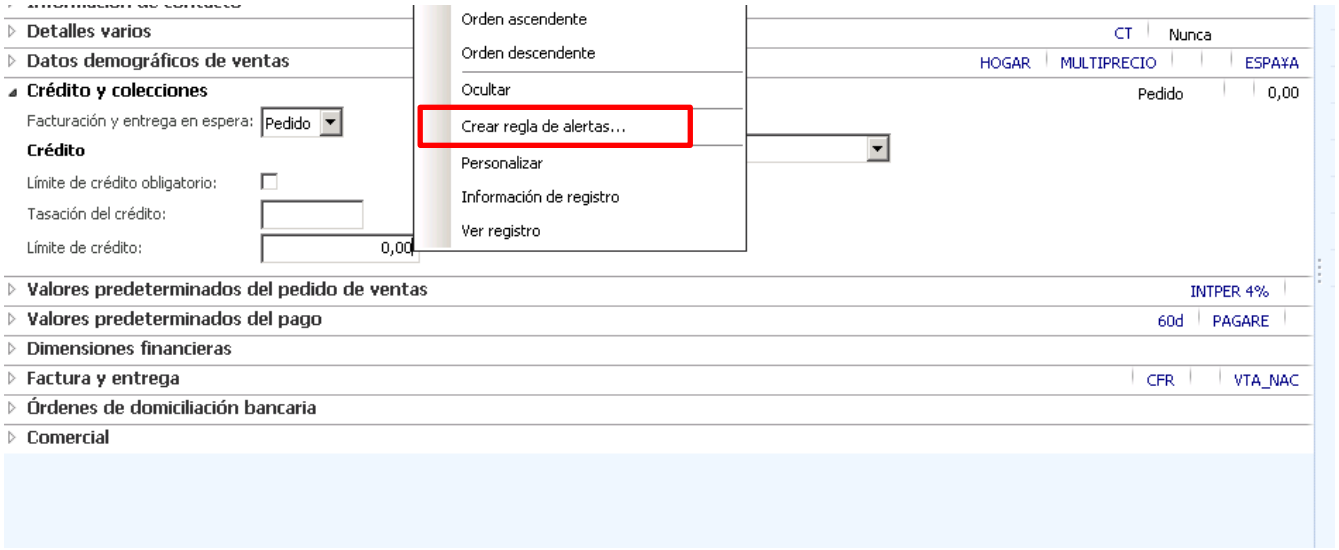



Ilustración 2. Crear regla de alertas

A partir de entonces se abre un formulario de configuración (Ilustración 3) donde se propone crear la alerta para el campo designado

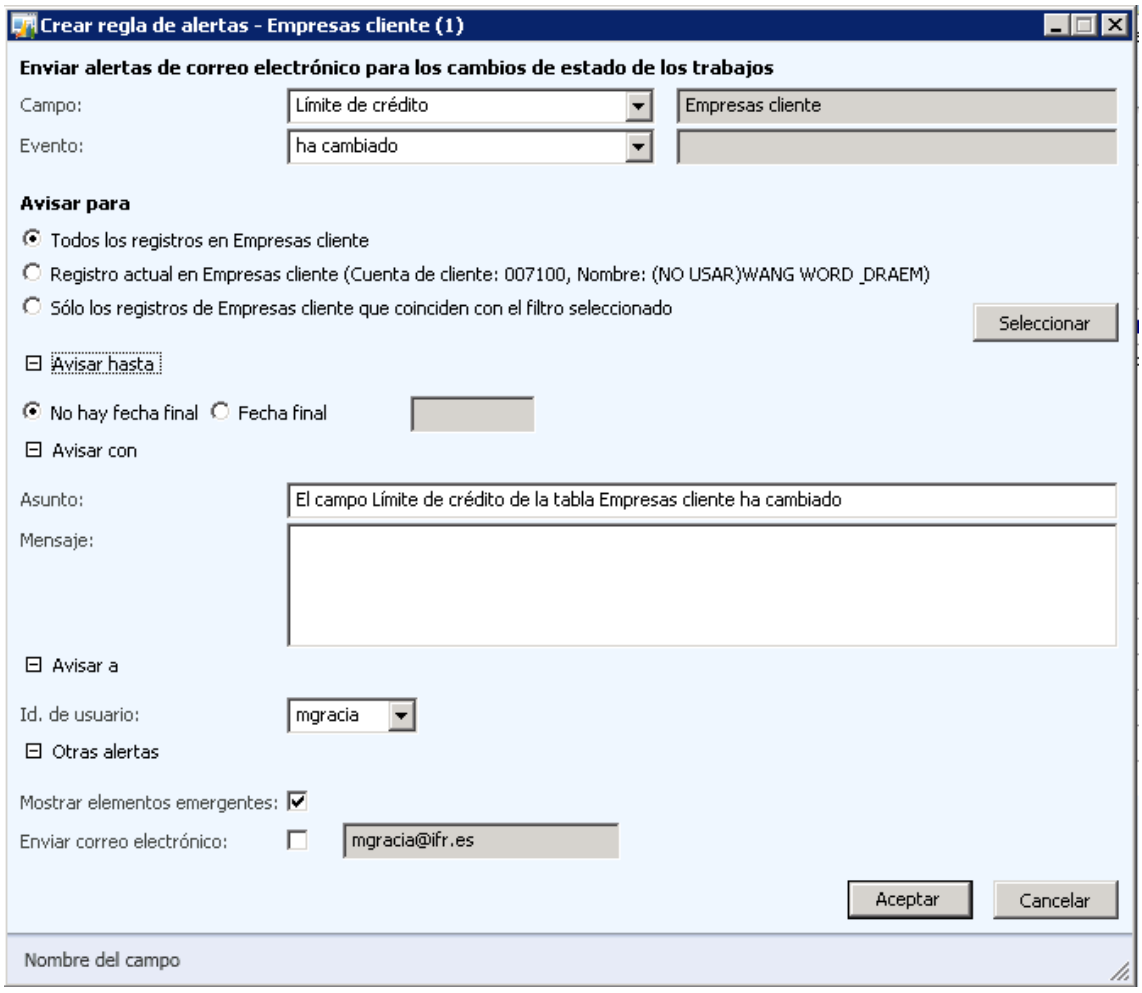



Ilustración 3. Configurar alerta

Es en este formulario donde el usuario puede configurar lo siguiente:

1. Campo: variable sobre la cual se quiere realizar la alarma. Se puede modificar y seleccionar cualquier otro campo del registro sobre el cual se quiere hacer el seguimiento.
2. Evento: suceso que ha de disparar la alerta. Es una lista fija con varias opciones:
    i. Se ha creado el registro. (no se tiene en cuenta el valor de campo)
    ii. Se ha eliminado el registro. (no se tiene en cuenta el valor de campo)
    iii. Se ha modificado el valor del campo.
    iv. Se ha aumentado el valor. (en caso de que campo sea te tipo numérico)
    v. Se ha disminuido el valor. (en caso de que campo sea te tipo numérico)
3. Avisar para: permite especificar si la alerta se configura para todos los registros de la tabla a configurar o el registro actual sobre el que está configurando la alerta o usando un filtro mediante el uso de una consulta.
4. Avisar hasta: permite definir una fecha límite en la cual la alerta dejará de ser efectiva.
5. Asunto: campo de texto corto configurable por el usuario, que se usa en el campo asunto en caso de que se envíe la alerta por correo electrónico.
6. Mensaje: campo de texto libre para que el usuario pueda configurar un mensaje en detalle que quiera enviar.
7. Avisar a: usuario de Dynamics AX a quién va dirigida la alerta, desafortunadamente, el sistema no permite enviar la alerta a más usuarios o grupos de usuarios.
8. Método de aviso: modalidad que se va usar para enviar la alerta
    i. Elemento emergente, pop-up que aparece en la aplicación de Dynamics AX cuando se produce el evento.
    ii. Correo electrónico, se envía un e-mail al usuario (si está configurado).

Tal y como se ha explicado en la introducción de la memoria y referente a este último punto, se pretende ampliar la funcionalidad de las alertas añadiendo un método de aviso más, enviar la alerta a una aplicación de Windows Phone.

En Dynamics AX, el usuario puede consultar una lista de alertas (Ilustración 4) que el sistema filtra para que sólo se vean aquellas que van dirigidas a él. La lista muestra varios datos, entre los cuales los más importantes son:

9. Estado de la alerta: permite ver si se ha leído o no.
10. Asunto: el texto corto configurado en la alerta.
11. Notificado para: texto que informa de forma automática Dynamics AX indicando qué registro ha sido afectado, dando información clarificadora al usuario.
12. Fecha: fecha y hora en la cual se ha producido el evento que ha disparado la alerta. Esta puede diferir de la fecha y hora en la que se hace la notificación al usuario.

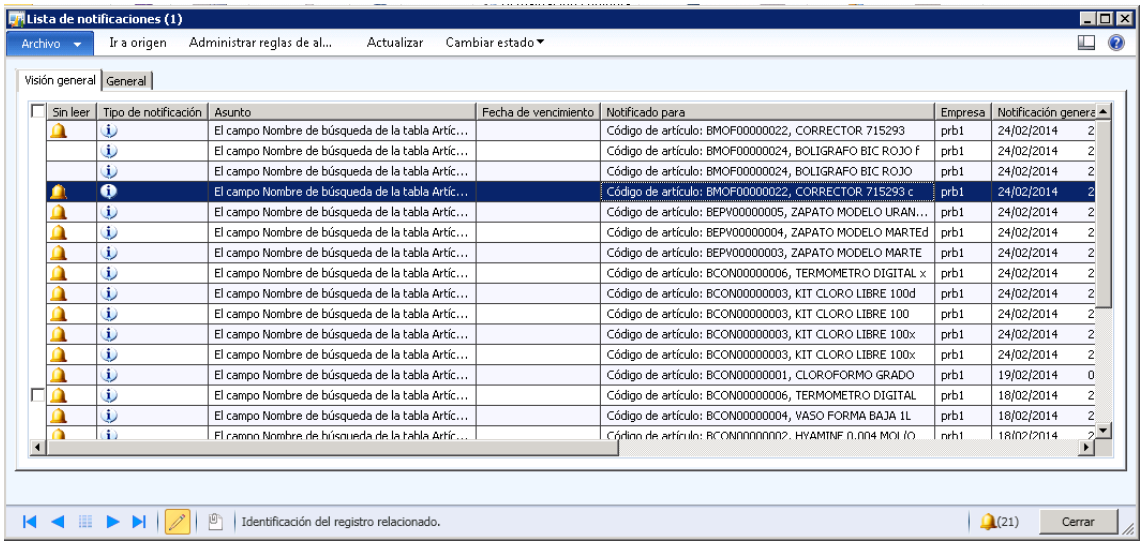



Ilustración 4. Lista de alertas

Y examinarla en detalle (Ilustración 5)

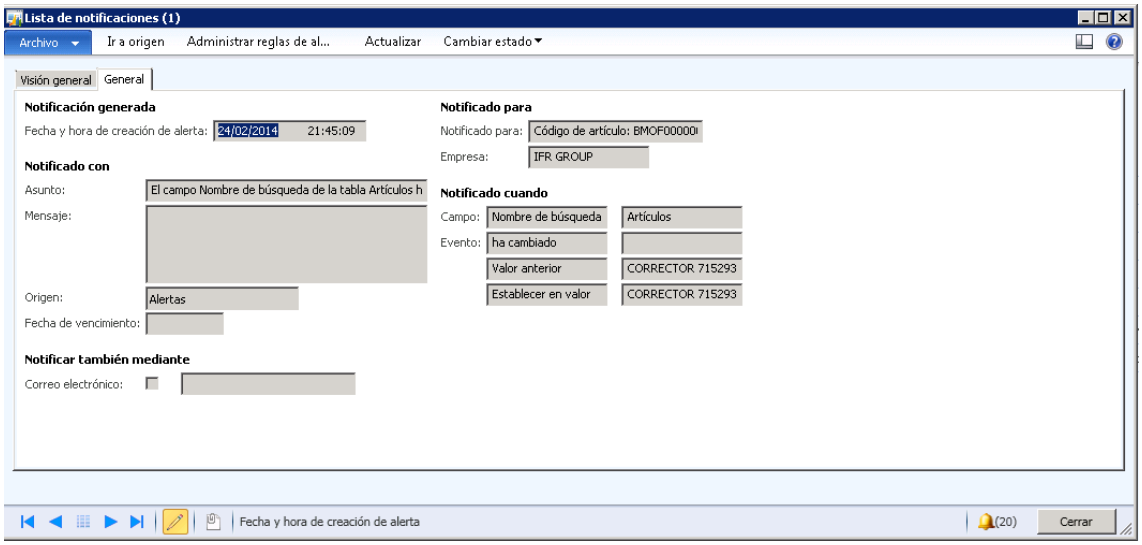



Ilustración 5. Detalle de la alerta

Donde aparte de ver los mensajes previamente configurados, el usuario puede consultar el evento que ha hecho dispararse la alerta. En caso que el evento sea producido por la modificación del valor de un campo del registro, se informa al usuario cuál era el valor anterior y posterior al suceso.

Es muy interesante la funcionalidad “Ir a origen” ya que permite al usuario navegar al formulario maestro donde encuentra el registro que generó el evento, esto es posible únicamente si la tabla tiene asociado un formulario para permitir su navegabilidad.

### *1.2. Como se presentará la información de la alerta en Windows Phone.*

La lista de alertas mostradas deben ser aquellas que están asociadas al usuario y pendientes de lectura. Por lo tanto el usuario se deberá identificar en la aplicación de Windows Phone para solicitar sus alertas.

Esta parte es muy importante, ya que determinará la manera en que se desarrollará la aplicación.

La información se deberá categorizar por tabla, mostrando en cada categoría otra lista donde se identifique claramente qué registro se ha visto afectado y una descripción corta de la misma.

La forma de informar qué registro ha generado la alerta habrá de ser una de manera con la que el usuario se sienta cómodo, algo con lo que trabaje normalmente, ha de ser algo conciso y directo que le permita relacionar la alerta con una entidad (ya sea un cliente, proveedor, artículo, etc…), ya que en la pantalla del terminal móvil no se dispondrá de mucho espacio para visualizar información.

¿Qué identifica un cliente, un proveedor o un artículo?, en todos estos casos tenemos un campo que además de ser identificador, es clave primaria del registro. Como ejemplo, un comercial está acostumbrado a llevar una cartera de clientes e identificar unos códigos con los que trabaja a diario, lo mismo podemos decir de un gestor de compras o de un responsable de almacén.

Por lo tanto, para mostrar la lista se mostrará un campo que dé una idea de qué registro ha provocado el evento y el campo asunto de la alerta.

Una vez mostrada la alerta, el usuario deberá ser capaz de consultar la alerta en detalle, mostrando toda la información relacionada.

Hay cuatro tareas a realizar, primero localizar el registro que provocó la alerta, segundo, poder identificar un valor que lo haga lo más identificable posible para poder montar la lista, tercero, poder consultar el detalle de la alerta y cuarto, poder marcarla como leída.

### *1.3. El origen de los datos en la alerta.*

Para poder montar la lista categorizada de las alertas (cada tabla representará una categoría, y de dentro de esta, una lista de las alertas asociadas) primero hay que averiguar cómo Dynamics AX relaciona una alerta con el registro que ha provocado el evento.

Las alertas generan tres registros en dos tablas relacionadas:

1. EventInbox, contiene los datos que informan al usuario de la alerta, básicamente mensaje, asunto, evento. El campo que relaciona la alerta con la tabla que generó el evento es AlertTableId.
2. EventInboxData, se generan dos registros:
    a. El primero contiene un array donde Dynamics AX guarda la información necesaria para informar al usuario de los cambios que generaron la alerta, el valor que tenía antes y después un campo en el momento del evento, o si, por ejemplo, se ha creado un registro nuevo o se ha borrado.
    b. El segundo contiene un array donde guarda información sobre el registro para poder navegar con la funcionalidad "Ir a origen". El nombre del campo asociado al evento, y el formulario. Dado que con la funcionalidad explicada no sólo se abre el formulario relacionado, sino que además el cursor se sitúa sobre el campo que generó el evento.

### *1.4. El reto de identificar los valores que definen un registro.*

Ya hemos indicado anteriormente que la aplicación en Windows Phone mostrará una lista categorizada por tablas que provocaron el evento de la alerta y que la información de esta ha de identificar el registro de forma clara para el usuario.

Dynamics AX tiene una estructura en la base de datos con unas 3000 tablas, por lo tanto, estamos hablando de miles de tablas sobre las que se puede configurar una alerta. Además están las personalizaciones que se pueden hacer sobre la aplicación creando nuevas estructuras de tablas, formularios, clases etc…

La solución que se presenta en este proyecto ha de ser capaz de presentar la información sin que se tengan que hacer futuras modificaciones porque se hagan nuevos desarrollos para la aplicación o porque se migra la versión actual de Dynamics AX a una superior.

La razón es muy importante, y es que, crear una aplicación sobre un ERP que lo usan unas 19000 empresas en todo el mundo (1) no puede estar sujeta a continuas modificaciones para su correcto funcionamiento.

Se puede pensar entonces que la solución planteada anteriormente, para identificar un registro, pasa por utilizar la clave primaria, no siempre es la solución óptima, puede ser que un registro tenga como clave primaria un índice compuesto por varios campos, lo que complica la presentación de datos en la aplicación de Windows Phone.

Todo esto nos lleva al inicio de la presentación del problema: ¿cómo presentar los datos en la lista?, ¿cómo se identifica el registro?, ¿cómo se evita tener que hacer continuas modificaciones?. Las distintas alternativas mostradas en este proyecto darán una idea de cuál es la más óptima y el porqué.

### *1.5. Condiciones que ha de cumplir la solución a escoger para el acceso a datos.*

Para poder llevar a cabo el proyecto, ha de ser posible consultar datos generados en Dynamics AX 2012 desde otra aplicación externa y también tiene que ser posible actualizar datos, ya que, aparte de consultar las alertas generadas, éstas han de ser marcadas como leídas. La opción a escoger debe cumplir los siguientes requisitos:

1. Facilitar la migración, Dynamics AX es un ERP que continuamente está evolucionando, con cada nueva versión llegan mejoras, pero también un trabajo de migración de la versión actual a la nueva. El método escogido debe facilitar esta tarea.
2. Máxima integración con AX 2012, cuanto más integrada esté la aplicación en AX 2012 más fácil es el mantenimiento, el desarrollo de aplicaciones que estén fuera de AX conllevan dificultades añadidas:
    a. Mayor dificultad al migrar hacia nuevas versiones.
    b. Mayor tarea de mantenimiento en la administración de estas aplicaciones que se traduce en:
        i. Complejidad en la gestión de errores (dónde se produce el error, en AX o en la aplicación externa)
        ii. Si se cambia el servidor de AX 2012 o la localización de la base de datos habrá que revisar que las aplicaciones externas sigan funcionando correctamente.
    c. Necesidad de añadir seguridad, desaprovechando la gestión de ésta que ya lleva integrada AX 2012
3. Desarrollo plural, la opción a escoger ha de dejar abiertas las opciones a otras alternativas, en este proyecto se presenta la aplicación móvil sobre Windows Phone, el desarrollo no debería impedir, en la medida de lo posible, poder usar otras plataformas móviles existentes en el mercado como tablets.
4. Tecnología ampliamente difundida y usada, cuanto más difundida esté la tecnología a usar mayor será la integración con otras plataformas, así como el soporte para posibles incidencias y futuras modificaciones o requerimientos. Se alarga la vida de la aplicación.

# 2. Posibles soluciones para acceder a los datos de la alerta

A continuación se mostrarán las distintas opciones para la búsqueda de los valores que identifican a un registro para poder construir la lista de alertas categorizada.

## *2.1. SQL como origen de datos*

La tarea no es sencilla, se presentan dos problemas, primero identificar la tabla asociada al evento y segundo el registro que lo ha provocado, para poder montar la lista por categoría (que representa el tipo de registro que generó el evento) y con lista de alertas dentro de ésta.

### *2.1.1. Como identificar la tabla que generó el evento*

Tal y como se ha explicado anteriormente, en la tabla EventInbox se contiene una referencia a la tabla (no el registro) que generó la alerta. Esta referencia es un número entero, y es que Dynamics AX identifica las tablas no por su nombre si no por un identificador único por tabla (Ilustración 6).

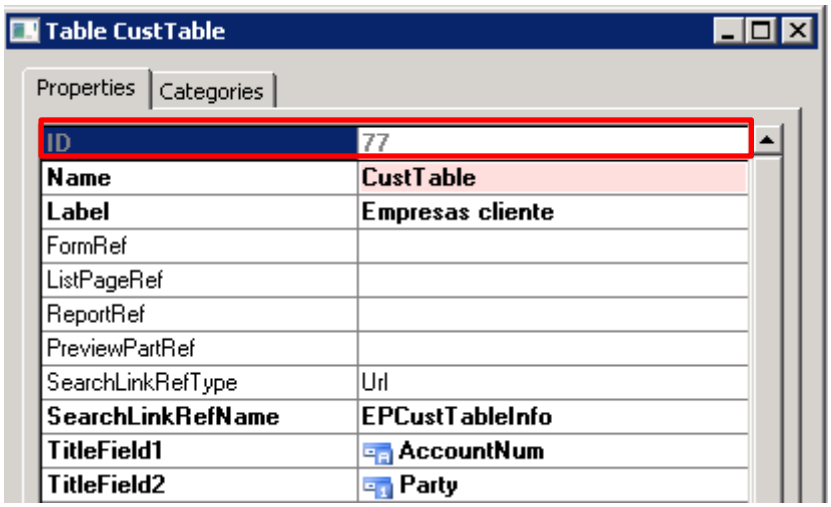



Ilustración 6. Identificador de la tabla

En el caso de la tabla de clientes, el identificador es el 77. Como se puede relacionar el código 77 con la tabla en sql “CustTable”?

Una opción es escribir el código del programa de manera que para cada identificador de tabla se devuelva el nombre de la tabla en SQL. Esta solución es descartable sólo por el hecho de utilizar hardcode (2). No es adaptable a futuras modificaciones y el trabajo a realizar es inmenso teniendo en cuenta el número de tablas que tiene Dynamics AX.

Dynamics AX tiene la tabla SQLDICTIONARY donde guarda para cada tabla y campo un registro con el nombre que tiene análogo en SQL. Esto es así porque no siempre el nombre de una tabla o campo es igual en el diccionario de datos de Dynamics AX.

La consulta que por lo tanto devuelve el nombre de la tabla asociado al identificador que ésta tiene en Dynamics AX es:

```
USE AXFORMACIONR2CU7
SELECT SQLNAME FROM SQLDICTIONARY
WHERE
TABLEID = 77 AND
FIELDID = 0 --EL REGISTRO CON VALOR FIELDID = 0 IDENTIFICA EL
NOMBRE DE LA TABLA
```

Hay que tener en cuenta que lo que encontramos es el nombre de la tabla (CustTable), no la etiqueta (Empresas clientes), dado que esta última no se encuentra a nivel de base de datos, si no en un fichero de tipo ASCII

Una vez identificada la tabla asociada al evento queda averiguar cómo identificar el registro en particular que disparó el evento. Esta parte es compleja, ya que, tal y como se ha explicado, Dynamics AX guarda en la tabla EventInboxData esta información en un array, de hecho lo que hace es serializar el registro asociado a la alerta en binario. Lo cual lo hace complejo identificar, no es posible extraer los datos deseados.

La opción pasaría por modificar Dynamics AX, añadiendo una relación más en la tabla EventInbox, que identifique de forma única el registro que provoca el evento. Con esta relación se podría consultar el registro responsable de la alerta y buscar el identificador asociado.

La solución a este nuevo problema es el RecId, todas las tablas en Dynamics AX tienen este campo de tipo entero que identifica de forma única un registro. Cada vez que se crea un registro nuevo en una tabla Dynamics AX inicializa de forma automática el valor del campo.

Es por tanto, clave primaria, el uso de este campo facilita las relaciones entre registros y puede usarse para identificar el registro que generó el evento, pero no para usarlo a modo informativo para el usuario.

#### *2.1.1.1. Modificar la tabla EventInbox para añadir una nueva relación.*

Para adaptar Dynamics AX a la solución planteada anteriormente hay que hacer básicamente dos modificaciones:

1. Modificar la tabla EventInbox añadiendo una referencia por RecId.
2. Modificar la clase que genera el registro EventInbox para informar el campo creado en el punto anterior.

De esta manera conseguimos que la tabla EventInbox tenga una relación con la tabla i el registro en particular que generó el evento.

Para el primer punto se añadirá a la tabla EventInbox el campo RefRecId (Ilustración 7) de tipo entero:

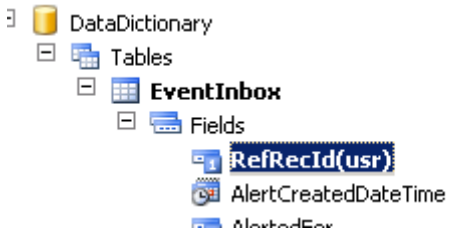



Ilustración 7. Añadir RefRecId

Para el segundo punto habrá que modificar la clase EventNotification (Ilustración 8), esta clase es la encargada de generar los registros en la tabla EventInbox cada vez que sucede uno de los eventos configurados para la generación de alertas.

```
        // happening upon drill-down, and AlertTableId contains the id of the table
        // for which are changes must be monitored in DB log
        inbox.AlertTableId = table.id();
        inbox.ParentTableId = table.id();

        //ADAPTACIÓN
        //INBOX, VARIABLE DE TIPO EVENTINBOX
        //RECORD, VARIABLE DE TIPO COMMON QUE CONTIENE EL REGISTRO QUE GENERÓ EL EVENTO
        inbox.RefRecId = record.RecId;

        recordCaptionGenerator = WorkflowRecordCaptionGenerator::construct(record);
        inbox.AlertedFor = recordCaptionGenerator.caption();
        list = SysDictTable::getUniqueIndexFields(table.id());

        if (list)
        {
            inbox.keyFieldList(list.pack());
            inbox.keyFieldData(SysDictTable::mapFieldIds2Values(list,record).pack());
        }
    }

    if (!inbox.UserId)
        return 0;

    if (!inbox.Subject && !inbox.Message)
        return 0;

    inbox.InboxId = inboxId;
    inbox.ShowPopup = true;

    if (record && inbox.GlobalRule == NoYes::No)
        inbox.CompanyId = record.company();

    inbox.insert();
```

Ilustración 8. Modificación de EventNotification

De esta forma, una vez identificada la tabla, con el valor del campo RefRecId se puede localizar el registro ya que es clave primaria.

Esta solución aunque sencilla, tiene el riesgo de tener que hacer futuras modificaciones, puede que en la siguiente versión de Dynamics AX la clase EventNotification sea modificada y por tanto el código añadido deje de funcionar correctamente.

### *2.1.2. Como obtener el nombre de una tabla para montar la lista por categorías*

Aún queda saber cómo hacer las agrupaciones, mostrar en las categorías el nombre de la tabla no es intuitivo para el usuario, ¿cómo se puede obtener el nombre de la tabla de clientes "CustTable"?, Dynamics AX utiliza un sistema complejo de etiquetas ya que la aplicación funciona en múltiples idioma, así pues, la CustTable es en español "Empresas clientes" y en inglés "Customers", pero esta información no está accesible en la base de datos (se guarda la descripción de los distintos idiomas en ficheros ascii).

En caso de escoger la opción de hacer la consulta directamente contra la base de datos, el nombre de las agrupaciones se podría resolver mediante una tabla de configuración donde guardar el nombre de la tabla de manera que esta información fuera accesible.

Esto implica una vez más añadir más modificaciones al sistema para poder llegar a la solución y además una tarea de mantenimiento de configuración para poder resolver los nombres de las tablas.

### *2.1.3. Como identificar los valores significativos para el usuario*

Una vez obtenido el nombre de la tabla, se puede pasar a localizar el registro responsable de la alerta por el campo RecId, para mostrar en la lista la información de qué registro ha lanzado el evento de la alerta.

Hay que buscar la clave primaria para así obtener el valor de aquél campo con el que el usuario pueda relacionar el registro. En SQL existen distintas vistas que nos pueden ayudar, para el caso de la tabla de clientes:

```
USE AxFormacionR2CU7
SELECT SYS.indexes.name AS INDEX_NAME, SYS.COLUMNS.name AS
FIELD_NAME FROM sys.indexes
INNER JOIN SYS.tables
ON
SYS.tables.object_id = SYS.indexes.object_id AND
SYS.tables.name = 'CUSTTABLE'
INNER JOIN SYS.index_columns
ON
SYS.index_columns.object_id = SYS.indexes.object_id AND
SYS.index_columns.index_id = SYS.indexes.index_id
INNER JOIN SYS.columns
ON
SYS.COLUMNS.object_id = SYS.index_columns.object_id AND
SYS.COLUMNS.column_id = SYS.index_columns.column_id
WHERE
SYS.INDEXES.is_primary_key = 1
```

El resultado de la consulta no sólo devuelve el campo esperado "AccountNum" si no también los campos "DataAreaId" y "Partition". ¿Cuál es la razón?

El campo DataAreaId identifica la empresa, Dynamics AX la incluye siempre porque puede tener el mismo código de cliente para distintas empresas.

El campo Partition, se incluye una funcionalidad nueva en Dynamics AX 2012 R2, según Microsoft se trata de “La división de procesos de una aplicación en unidades lógicas o funcionales.” (3) se trata de una funcionalidad para aislar datos y no dar acceso a usuarios no autorizados.

Ambos campos son incluidos en todos los índices por temas de rendimiento, ya que cualquier consulta que haga el sistema por la petición de un usuario, siempre van filtradas por empresa (dataAreaId) y partición (Partition).

Por lo tanto, si se opta por la solución de usar consultas directamente contra la base de datos, se tendrá que obviar estos dos campos, ¿de qué manera?, nuevamente distintas opciones:

1. Hardcode, solución que como la anterior no es una opción. No es adaptable a futuras modificaciones.
2. Configurar los nombres de los campos en una tabla en Dynamics AX, mejor opción que la primera, pero mayor tarea de mantenimiento, obliga al administrador del sistema a hacer una tarea de investigación para saber qué campos no quieren que se muestren en la lista por el hecho de estar incluidos en el índice que es clave primaria. En la versión Dynamics AX 2012 RTM a diferencia de Dynamics AX 2012 R2 el campo Partition no está incluido.

Todo esto supone una mayor complejidad, mayor mantenimiento, lo que no ayuda a futuras mejoras, modificaciones debido a cambios de versión en Dynamics AX. Aún entonces tenemos el problema de qué campos de la clave primaria vamos a decidir usar para mostrar en la lista de alertas al usuario.

La solución más genérica pasaría por usar el primer campo del índice. Esto no asegura que sea un valor que el usuario pueda identificar o asociar con el registro. Como ejemplo de la complejidad asociada al problema (descartando ya, los campos de DataAreaId y Partition):

1. Tabla de clientes (por si se configura una alerta al modificar el nif de un cliente), la clave primaria es AccountNum, que es el código de cliente, este es lo suficientemente claro para el usuario como para que sea capaz de relacionar ese valor con un cliente.
2. Tabla de transacciones de cliente (por si se configura una alerta al producirse un impago), la clave primaria es TransDate (fecha) y Voucher (asiento), ninguno de los dos campos puede hacer que un usuario, al verlo, relacione ese valor con el cliente que ha generado el impago.

Una vez obtenido el nombre del campo que se va a usar para mostrar en la lista de alertas, aún queda hacer la consulta para devolver el valor del mismo. Tal y como se ha mostrado en la obtención del origen de datos, la relación con la tabla que origina el evento se hace por RecId, y por el campo TableId, se ha explicado también cómo buscar el nombre de la tabla en SQL.

Por lo tanto, se debería desarrollar un sistema, que en tiempo de ejecución, fuera capaz de montar las distintas consultas en transact-sql, usando el nombre de la tabla, el valor de Recid y devolviendo el valor del campo identificativo tal y como se ha explicado al inicio de este punto.

Aunque complejo, este sistema asegura poder resolver la solución buscada sea cuál sea la tabla relacionada con la alerta sin necesidad de hacer continuas modificaciones o adaptaciones.

#### *2.1.4. Como obtener los datos de la alerta en detalle*

Una vez obtenidos todos los datos para poder montar la lista, hay que poder consultar en detalle los datos de una alerta en particular. La tabla EventInbox contiene los campos de asunto, mensaje, notificado para y fecha explicados anteriormente.

El problema está en que la información restante que contiene los cambios de valor del campo para el que se ha configurado la alerta se guardan en un array en formato binario, por lo que no sería posible consultar estos datos desde sql.

#### *2.1.5. Como actualizar la alerta para marcarla como leída*

Por último quedaría actualizar el registro para marcarlo como leído, lo único que hay que hacer es modificar el valor del campo IsRead de la tabla EventInbox para inicializarlo a “1”. Aunque pueda parecer suficiente existe un riesgo. Los objetos tabla en Dynamics AX pueden contener código que se ejecute en ciertos eventos, como por ejemplo, al insertar un registro, al modificar un campo de una tabla o al actualizarla. Por lo tanto, actualizar el registro directamente por SQL tiene el riesgo de no ejecutar código que de hacerlo desde Dynamics AX si se haría. En el caso de la tabla EventInbox, no existe código para el evento UPDATE, pero esto no asegura que en futuras versiones de Dynamics AX si se haga, dejando por lo tanto, un problema que habría que resolver haciendo una modificación.

### *2.2. Dynamics AX como origen de datos*

Otra forma de encontrar el origen de datos que generó el evento es desde el propio Dynamics AX. La opción que se mostrará a continuación, a diferencia de la anterior no necesita hacer modificaciones del código estándar para encontrar los datos deseados.

#### *2.2.1. Como identificar la tabla que generó el evento*

Tal y como se ha explicado en el punto 1.3, la tabla EventInbox tiene en el campo AlertTableId que contiene el identificador de la tabla que generó el evento y la tabla EventInboxData el registro serializado. Las opciones a partir de aquí son dos:

1. Deserializar el registro a partir de la tabla EventInboxData.
2. Ejecutar las mismas funciones que usa Dynamics AX, cuando desde una alerta se usa la función “Ir a origen”.

Ambas opciones son válidas, pero la primera está sujeta a mayores riesgos, puede ser que en futuras versiones se cambie la estructura de datos usada para guardar el registro pero las llamadas a las funciones sigan igual adaptándose la funcionalidad.

Un ejemplo de esta situación que se podría dar en un futuro son las facturas de venta. Desde la versión 3.0 a la versión 2012 la estructura de datos usada para guardar las facturas de venta en Dynamics AX 2012 ha cambiado de forma significativa, incluso las clases que generan las facturas también han cambiado notablemente, pero la llamada al método principal para registrar una factura sigue inalterable. Esto es así ya que de esta manera se consigue facilitar la migración de adaptaciones  a las nuevas versiones.

Por lo tanto, se trata de seguir el estándar y su funcionalidad en la medida de lo posible. Con una variable de tipo common se puede obtener, a partir de la tabla EventInbox (Ilustración 9), al igual que hace el estándar, el registro que ha disparado el evento

```
protected str commonIdFromEventInbox(EventInbox _eventInbox)
{
    EventFieldData  eventFieldData = _eventInbox.keyFieldData();
    SysDictTable    sysDictTable = new SysDictTable(_eventInbox.AlertTableId);
    Common          common;
    str             ret;

    common = SysDictTable::findFromKeyData(_eventInbox.ParentTableId, _eventInbox.keyFieldData());
```

Ilustración 9. Obtener common a partir de EventInbox

Un objeto common es una clase que se usa de forma polimórfica para hacer referencia a una tabla. No contiene ningún dato, a excepción del identificador de la tabla y el RecId, estos valores se pueden utilizar luego para obtener otros valores en específico del registro deseado.

### *2.2.2. Como obtener el nombre de una tabla*

Tal y como se ha explicado, hay que montar una lista de alertas agrupada por categorías, cada categoría está representada por una tabla, la cuestión es, ¿cómo obtener en Dynamics AX el nombre descriptivo de la tabla?.

El objeto clase SysDictTable permite acceder a las propiedades de la tabla. Por lo tanto, éste se puede inicializar a partir del registro EventInbox, que contiene el identificador de tabla que generó el evento que disparó la alerta.

La propiedad “Label” (Ilustración 10) contiene el nombre de la tabla:

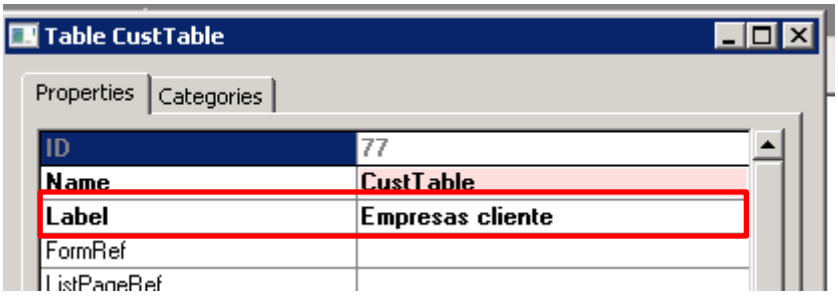



Ilustración 10. Propiedad Label

Se puede observar como Dynamics AX usa de forma asidua esta propiedad, en el ejemplo de la tabla de clientes y su formulario maestro (Ilustración 11):

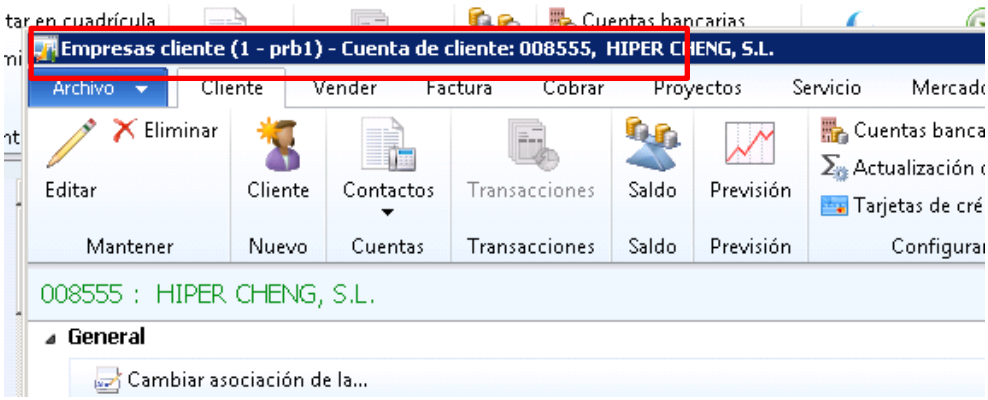



Ilustración 11. Propiedad Label en formulario de clientes

Esta es la propiedad que Dynamics AX usa siempre que se refiere a registros de cualquier tabla, por lo tanto, el usuario la puede identificar rápidamente.

Con el objeto SysDictTable (Ilustración 12) se puede acceder de forma sencilla a la propiedad y obtener el texto descriptivo de la tabla que se usará en las agrupaciones.

```
1  static void tutorialTableId2Label(Args _args)
2  {
3      info(new SysDictTable(tableNum(CustTable)).label());
4  }
```

Ilustración 12. Uso de SysDictTable para acceder a Label

### 2.2.3. *Como identificar los valores significativos para el usuario*

En este punto se mostrará cómo aprovechar la propiedad TitleField de las tablas para poder mostrar en cada categoría, una lista con un texto que el usuario pueda identificar y vincular con una identidad de forma sencilla y rápida. El objeto tabla que pertenece a la arquitectura en Dynamics AX permite varias funciones:

1. Crear una tabla análoga en Microsoft SQL.
2. Escribir en lenguaje x++ sentencias de transact-sql para el acceso a datos de forma rápida y sencilla.
3. Añadir métodos y propiedades que pueden ser usados durante el desarrollo para facilitar adaptaciones y modificaciones.

La propiedad TitleField (Ilustración 13) del objeto tabla, según las best practices (4), debe ser descriptivo para el usuario. En el caso de la CustTable (tabla de clientes), el TitleField1 es el CustAccount que identifica de forma única a un cliente:

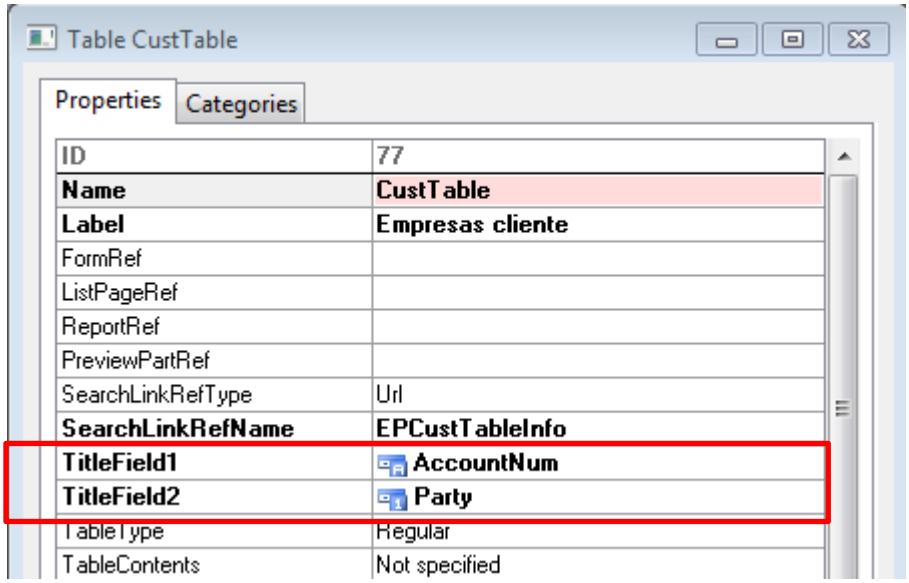



Ilustración 13. Propiedad TitleField

La cuestión a resolver ahora sería, como buscar el registro que es origen del evento, y devolver el valor del campo asociado a TitleField, ya que este valor sería mostrado en la lista. Dynamics AX tiene más de 3000 tablas, cada una de las tablas tendrá un titleField distinto.

La solución no puede pasar por escribir una consulta distinta para cada tabla que exista en el sistema. No se facilitaría la migración a versiones futuras, mantenimiento de código muy costoso, no aporta soluciones a personalizaciones de código que tengan otras instalaciones de Dynamics AX.

Dynamics AX tiene objetos que permiten llegar a la solución deseada, de forma que con poco código se obtenga el valor deseado sea cual sea la tabla.

Partiendo del objeto common que se explica al inicio del origen de datos y de la propiedad titlefield que contiene el identificador del campo que se quiere devolver se puede obtener su valor (Ilustración 14).

```
1  protected str commonIdFromEventInbox(EventInbox _eventInbox)
2  {
3      SysDictTable    sysDictTable = new SysDictTable(_eventInbox.AlertTableId);
4      Common          common;
5      str             ret;
6
7      common = SysDictTable::findFromKeyData(_eventInbox.ParentTableId, _eventInbox.keyFieldData());
8
9      if (common.RecId != 0)
10     {
11         if (sysDictTable.titleField1())
12         {
13             ret = any2str(common.(sysDictTable.titleField1()));
14         }
15         else
16         {
17             ret = int642str(common.RecId);
18         }
19     }
20
21     return ret;
22 }
```

Ilustración 14. Obtener valor de TitleField1

En la línea 3 se usa el objeto clase SysDictTable que se construye con el identificador de la tabla (que se encuentra en el registro EventInbox). Este objeto permite acceder a las propiedades de la tabla.

En la línea 7, tal y como se explicó en el punto de origen de datos, se obtiene el objecto polimórfico common que devuelve el registro que provocó el evento.

En la línea 13, usando el objeto SysDictTable y el objeto common se accede al valor del campo especificado en la propiedad TitleField1 mediante el método titleField1 de la clase sysDictTable.

Esto es posible porque, aunque el objeto common no contiene los campos de las tablas (no se puede especificar common.AccountNum para obtener la cuenta de cliente), en caso que common se construya a partir de la tabla de clientes, sí es posible acceder al valor de cualquier campo si se especifica el identificador de éste.

Todos los campos de las tablas de Dynamics AX tienen un identificador de tipo entero (Ilustración 15), en el caso de la tabla de clientes:

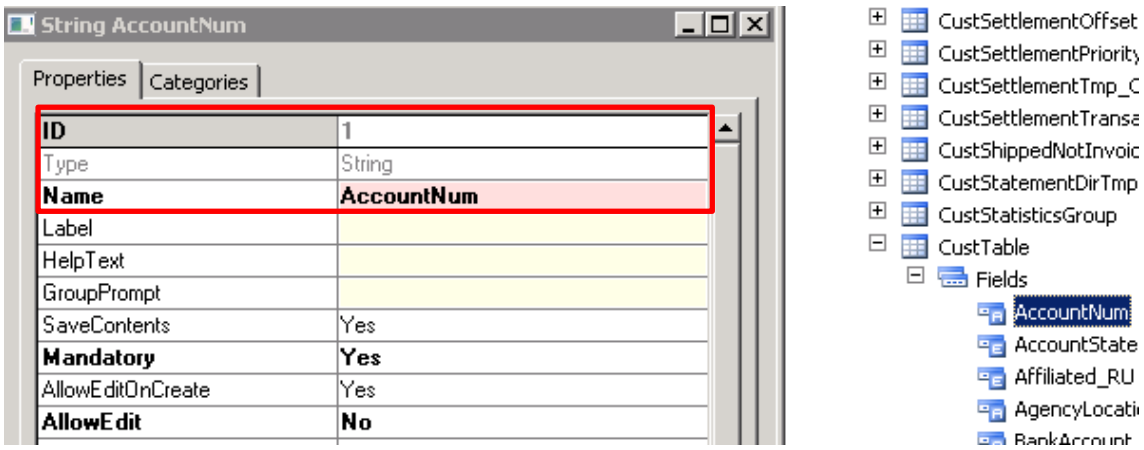



Ilustración 15. Identificador de campo para una tabla

El valor identificador del campo AccountNum (que además es el valor especificado en la propiedad TitleField1) es el 1. Es decir, si el objecto common es de tipo CustTable, entonces common.(1) nos devuelve el valor de la cuenta de cliente “AccountNum”.

Por lo tanto esta solución, es capaz, para cualquier registro de una alerta, de devolver el valor del campo TitleField1, que se usará a modo de información para el usuario en la lista de alertas de la aplicación de Windows Phone.

### *2.2.4. Como obtener los datos de la alerta en detalle*

Para la obtención de los datos para la consulta de la alerta en detalle, desde el propio Dynamics AX, lo más recomendable es ejecutar los mismos métodos que utiliza el ERP para inicializar el formulario EventAlertInbox.

Hacerlo de esta manera, y no intentando sacar la información del array que se encuentra en la tabla EventInboxData, nos asegura una mayor seguridad en la funcionalidad del código. En caso contrario, habría que asegurarse la correcta interpretación de los datos del array para cada uno de los distintos tipos de eventos que se puede configurar en una alerta. Mejora una mayor integridad con Dynamics AX, es decir, aseguramos mejor la migración de la aplicación a versiones futuras, puede que se cambie la estructura de datos o del array, pero probablemente, la manera en que se extrae la información usando las mismas clases que se usan actualmente siga funcionando prácticamente igual.

### 2.2.5. Como actualizar la alerta para marcarla como leída

Para marcar la alerta como leída hay que actualizar el registro EventInbox (que se crea cuando se origina una alerta), poniendo el valor del campo IsRead (de tipo booleano) a true (Ilustración 16). Hacerlo de esta manera, desde Dynamics AX, asegura que si hay código en el evento UPDATE, este sea ejecutado, esto es muy importante, ya que de lo contrario, podrían generarse inconsistencia en los datos.

```
public WebService_PushResult udpateAlertIsRead(RecId    _recId)
{
    WebService_PushResult   ret = new WebService_PushResult();
    EventInbox              eventInbox;

    select firstOnly forUpdate eventInbox
        index hint RecId
        where eventInbox.RecId  == _recId;

    if (eventInbox)
    {
        eventInbox.IsRead = true;
        eventInbox.update();

        ret.parmProcessOk(true);
    }
    else
    {
        ret.parmProcessOk(false);
        ret.parmProcessTxt(strfmt("No se encontró el registro con RecId %1. Hable con el administrador.", _recId));
    }

    return ret;
}
```

Ilustración 16. Actualizar alerta como leída

## 2.3. ¿Por qué descartar SQL para acceder a los datos?

En los puntos anteriores se ha mostrado cómo resolver los problemas para acceder a los datos generados por las alertas para mostrar los datos en la aplicación móvil. A continuación se exponen las razones por las que se descarta el acceso a los datos de las alertas directamente contra SQL:

1. No facilita la migración, escribir el código fuera de Dynamics AX para la consulta de datos complica el proceso de migración de la aplicación. En cada nueva versión cabe la posibilidad que cambie la estructura de datos, por lo tanto, habría que reescribir el código. Se puede pensar que el mismo problema se presenta si el código está escrito en el propio Dynamics AX, pero la tarea en este siempre es más sencilla, ya que el código se apoya en el propio framework del sistema. Y a la vez que se hace la

migración de Dynamics AX a la nueva versión, también se está migrando el código que se usará para la aplicación de Windows Phone.

2. Hay que realizar adaptaciones para poder acceder al registro que provocó el evento, nuevamente no facilita la migración. En el caso de usar Dynamics AX para consultar los datos de las alertas no hay que realizar modificaciones, se minimiza el riesgo en una migración.
3. Las agrupaciones se deberían hacer por el nombre de la tabla y no por su etiqueta, dado que esta última no es accesible en la base de datos, esta información se encuentra en un fichero ASCII y no hay manera de saber en la base de datos saber que identificador de etiqueta tiene una tabla.
4. Por SQL no hay acceso a los valores del campo que desembocaron el evento al ser modificados, dado que estos datos se guardan en un array en un campo de tipo binario. Por lo tanto, hay menos información accesible.
5. Mínima integración con Dynamics AX, el código está fuera del ámbito de la aplicación, con lo que hay que volverse a plantear funcionalidades que ya están resueltas por el sistema, como por ejemplo:
    a. Seguridad, Dynamic AX 2012 dispone de un sistema que permite filtrar los datos a los que puede acceder un usuario o incluso el propio acceso a la aplicación.
    b. Conexión con la base de datos, el propio Dymamics AX se encarga de abrir las conexiones y traducir las consultas que se hacen mediante código x++ a transact-sql. Por lo tanto, en caso de escoger la opción de SQL hay que llevar a cabo una tarea de configuración, acceso y seguridad a la base de datos.
    c. Se desaprovecha el framework de Dynamics AX que facilita el acceso a datos o las propiedades de tablas para obtener las descripciones.

# 3. Opciones para acceder a Dynamics AX desde una aplicación externa

Una vez descartada la opción de SQL para la consulta de datos de las alertas, queda el problema de cómo se puede acceder a Dynamics AX desde una aplicación externa.

Hasta ahora todo lo que se ha explicado sobre cómo acceder a los datos desde Dynamics AX se hacía dentro del propio sistema, la pregunta es, ¿Cómo hacerlo posible desde otra aplicación?, al fin y al cabo todo el desarrollo está orientado para consultar datos (que se generan de Dynamics AX) desde la aplicación móvil.  Existen dos opciones:

## *3.1. Business Connector*

Business Connector es un componente de Dynamics AX que se incluye en el framework de .net para poder ejecutar código x++ y llamadas a objectos de Dynamics AX (tablas, vistas, clases…) desde un programa escrito en C#.
Este componente fue introducido por primera vez en la versión 4.0 (5)http://msdn.microsoft.com/en-us/library/aa659581(v=ax.10).aspx), el objetivo era poder llamar a los objetos de Dynamics AX desde el portal de negocio, una aplicación web creada específicamente para trabajar con AX.
En la versión 2012 se mejoró Business Connector, ya que desde el propio Visual Studio 2010 se puede ver el árbol de objetos de Dynamics AX y trabajar con los objetos directamente, en las versiones anteriores esto no era así, lo que complicaba notablemente el desarrollo de aplicaciones.

## *3.2. Application Integration Framework*

Application Integration Framework (AIF) fue introducida en la versión Dynamics AX 4.0 (6), esta herramienta permite la comunicación con sistemas externos mediante el intercambio de ficheros XML para la lectura, actualización o inserción de datos en la aplicación de forma asíncrona. Un proceso por lotes interpreta y procesa los ficheros, ejecutando los procesos necesarios en la aplicación, como la creación de pedidos, actualización de facturas, etc…

En la versión Dynamics AX 2012 se incluyeron grandes mejoras, como el poder publicar procedimientos desarrollados dentro de la aplicación sobre un servicio de Windows basado en Windows Comunication Foundation. El propio AOS actúa como un servicio de Windows que puede ser consumido por otras aplicaciones utilizando el protocolo SOAP.

## *3.3. ¿Por qué escoger AIF?*

AIF es la mejor opción, a continuación se muestran las razones:

1. Principio de responsabilidad individual, ¿de quién es la responsabilidad de buscar la información y organizar las listas con categorías?, ¿de Dynamics AX o de un programa

escrito en c# usando business connector?. Está claro que esa responsabilidad recae en Dynamics AX, puesto que es ahí donde encontraremos el origen de los datos, las descripciones de las tablas para poder montar la estructura de datos que luego se usará en la aplicación móvil. La aplicación que luego pasa esa información al Windows Phone no tiene la responsabilidad de saber dónde está esa información o como se debe organizar, la responsabilidad de ésta debe ser la de hacer accesible esta información.

2. Mejora de la migración, en caso de escoger AIF, al migrar la aplicación se está migrando el código usado para la aplicación móvil, en caso contrario, habrá que migrar tanto Dynamics AX como la aplicación escrita en c# que usa bussiness connector y que puede haberse quedado obsoleta al cambiar la estructura de datos y objetos usada.
3. AIF mejora la interacción con otros programas externos, puesto que se basa en WCF usando protocolo SOAP (7) altamente extendido.

# 4. AIF, cómo configurar el servicio

Existen básicamente dos formas para la creación de un servicio basado en AIF, el primero que se va enseñar se basa en el uso de queries, el segundo se basa en el uso de datacontracts.

## *4.1. AIF basado en queries*

El objeto query en Dynamics AX permite definir una consulta con una o varias tablas relacionadas, AIF puede usar el objeto para publicarlo en el servicio para así consultar, insertar o actualizar datos. Lo que hace el sistema es serializar los resultados de la consulta, generando listas de la consulta realizada, en caso de haber tablas relacionadas, los resultados se anidan, de manera que una lista contiene a su vez listas de aquellos registros vinculados por el criterio que se haya indicado en la query. En caso de querer insertar registros, habría que crear la estructura de listas para poder insertar los valores deseados.

### *4.1.1. Estructura de la query.*

A continuación se muestra como montar la query i el consecuente servicio de AIF para consultar las alertas debiendo usar la tabla EventInbox i EventInboxData que como se ha indicado anteriormente, contienen los datos de las alertas:

Se crea la query (Ilustración 17) con los campos que se desean publicar al servicio que haga al consumo del mismo. Objecto query:

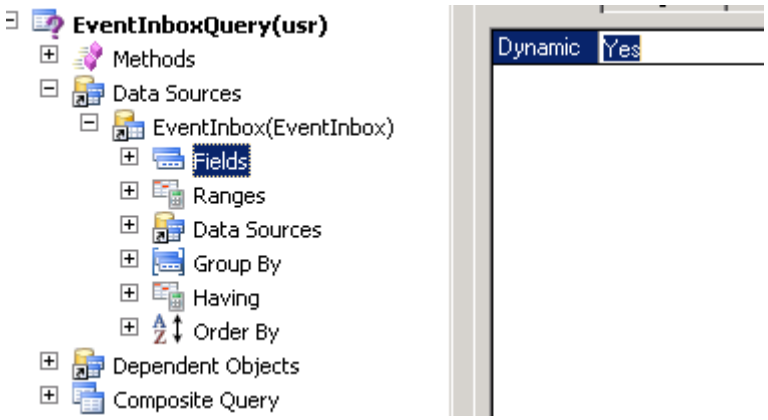



Ilustración 17. Objeto query

En la query se ha incluido la tabla EventInboxData con las relaciones con EventInbox, la propiedad de Fields se ha dejado a Dynamics Yes para hacer públicos en el servicio todos los campos de la tabla (se pueden limitar la lista de campos a mostrar).

Con el asistente de servicio de documento AIF (Ilustración 18) Dynamics AX genera todas las estructuras necesarias para poder montar el servicio AIF basado en la query.

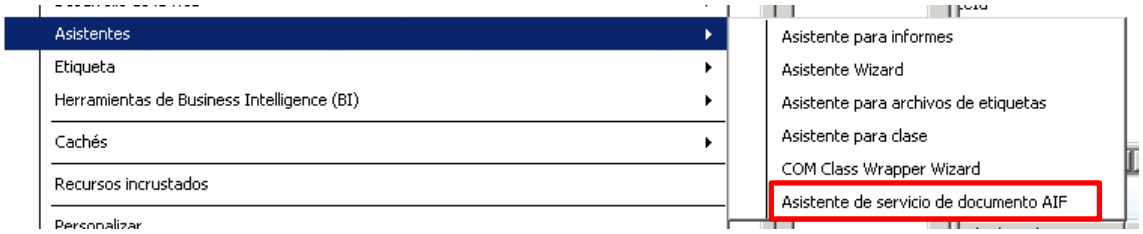



Ilustración 18. Asistente de servicio de documento AIF

Lo que facilita enormemente la tarea de programación, el proceso únicamente necesita como parámetros (Ilustración 19) la query, el nombre de la clase que generará y la etiqueta que se usará a modo descriptivo

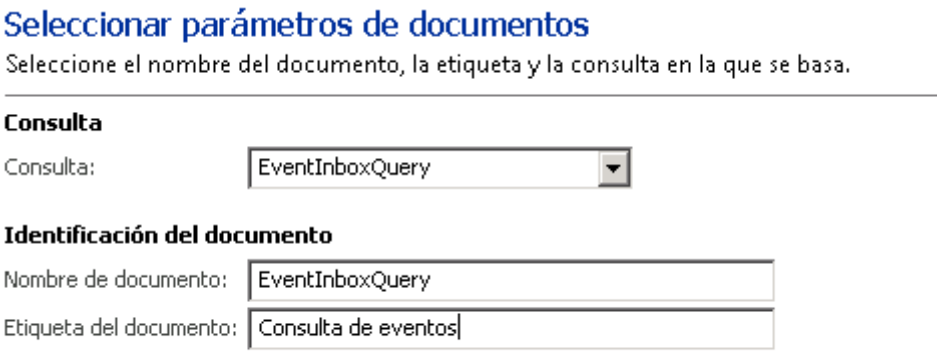



Ilustración 19. Parámetros para el asistente de servicio de documento AIF

Dado que AIF lo que usa una query que al final lo que hay detrás es una tabla, proporciona una serie de operaciones (Ilustración 20) para trabajar con ellas

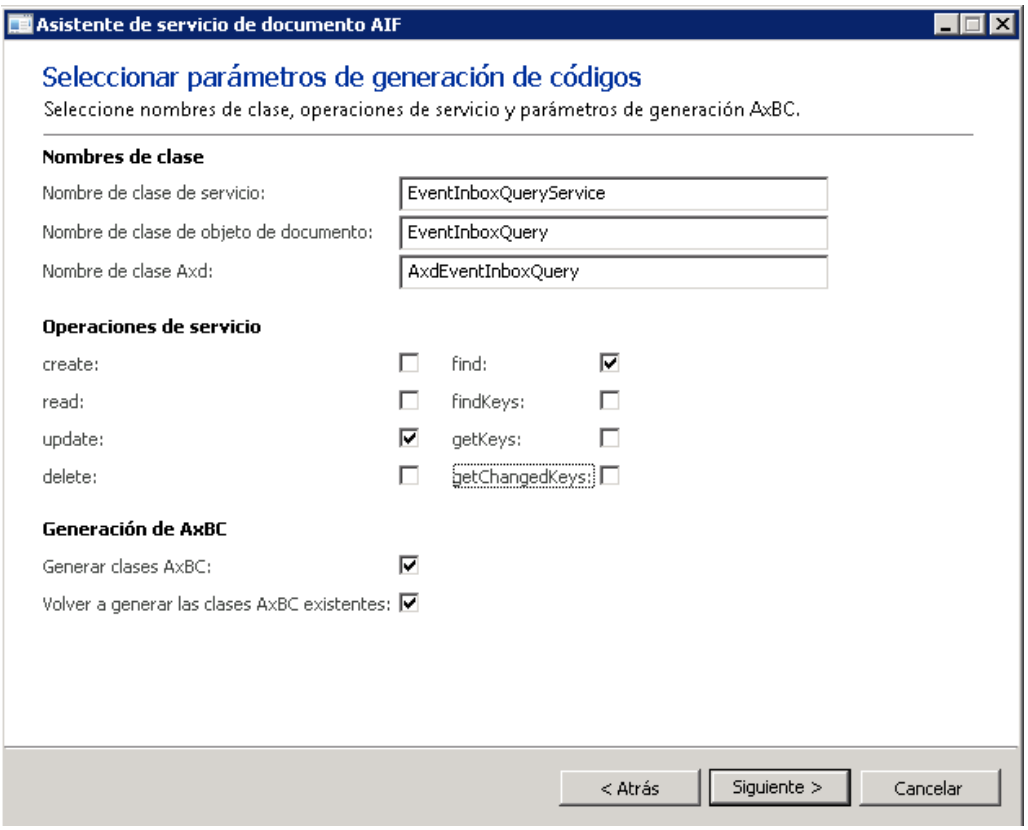



Ilustración 20. Operaciones del servicio de documento AIF

En ese caso son sólo necesarias “find”, para la búsqueda de datos en la query, y update para actualizar el registro y para poder marcar las alertas como leídas.

Una vez finaliza el asistente Dynamics AX genera el proyecto (Ilustración 21) con los objetos necesarios para poder atacar al servicio desde el exterior, como por ejemplo las clases necesarias para hacer el mapeo entre el dato de la tabla y el dato que se leerá en el XML

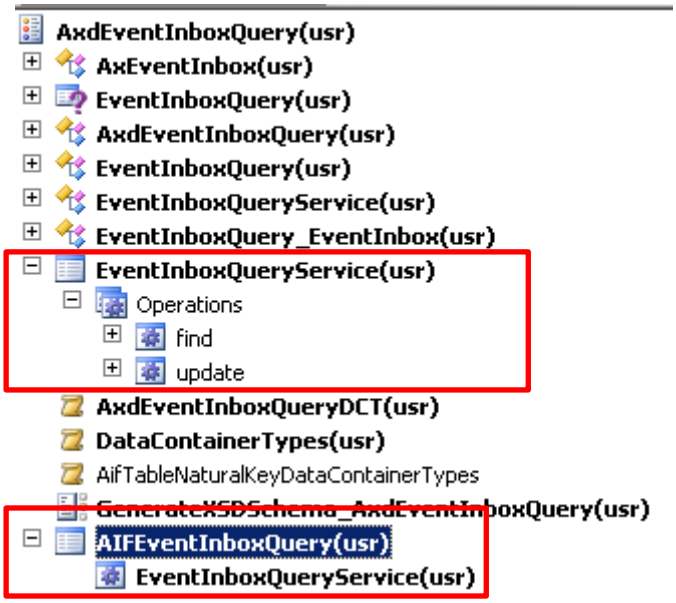



Ilustración 21. Proyecto generado por el asistente

El sistema automáticamente ha generado el servicio EventInboxQueryService que permite las dos operaciones, find y update, sobre la tabla EventInbox que se ha definido en la query.
Para poder asociar estas operaciones a un puerto de salida, de manera que Dynamics AX genere el servicio se necesita el grupo de servicios AIFEventInboxQuery donde se incluirán todos los servicios que sean necesarios.
Para poder publicar este servicio se debe crear un puerto de entrada (Ilustración 22) en Dynamics AX

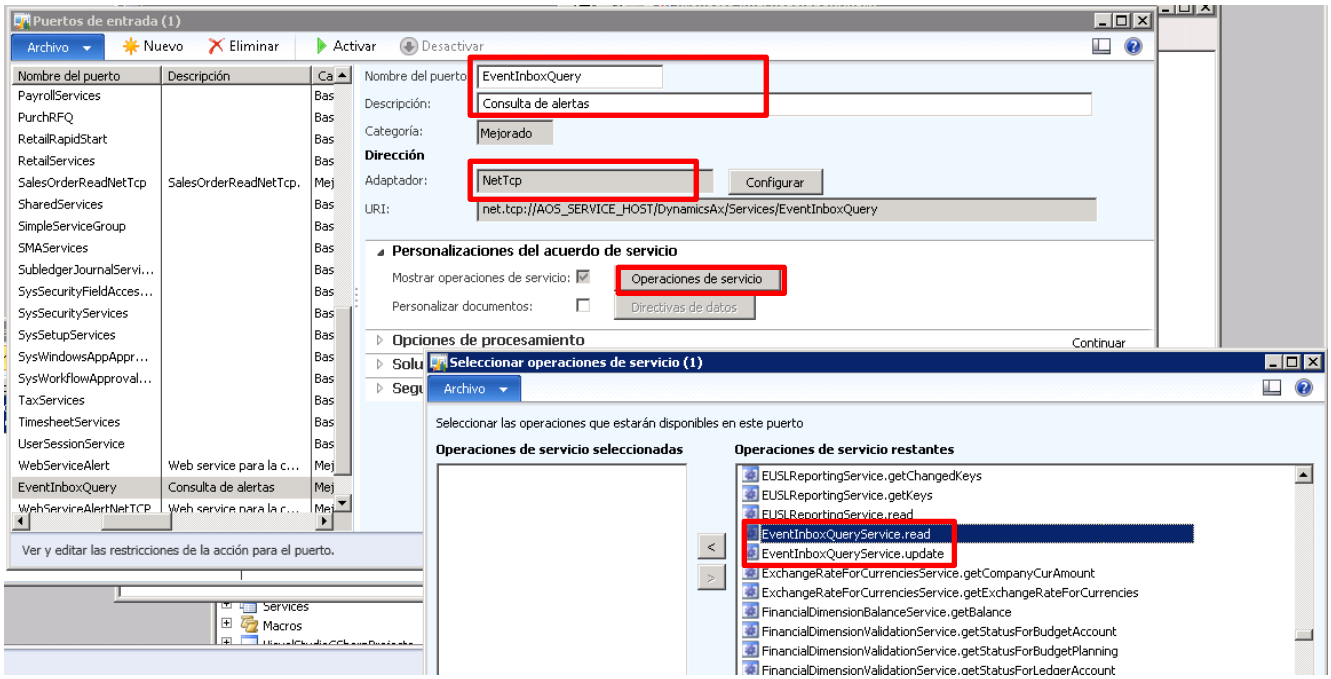



Ilustración 22. Configuración del puerto de entrada

La herramienta de AIF dispone distintos adaptadores para la comunicación, los cuales son:

1. HTTP, para poder enviar mensajes de formas síncrona usando HTTP o HTTPS a través de un servicio web.
2. NetTCP, usado para enviar mensajes entre extremos basados en WCF usando protocolo SOAP, la comunicación es síncrona y los datos son intercambiados casi al momento.
3. FileSystem, permite el intercambio asíncrono de ficheros en formato XML
4. MSMQ, permite usar colas para el transporte de mensajes, es un tipo de comunicación asíncrona incluida en WCF, a diferencia de NetTcp, si el extremo no está activo, el mensaje queda en la espera en la cola a que el servicio vuelva a estar activo y devuelva la información.

Para el caso que se está mostrando se decide usar NetTcp, dado que se desea que la comunicación entre Dynamics AX y la aplicación que habrá de publicar la información sea prácticamente inmediata.

En las operaciones de servicio se añadirán las operaciones de lectura y actualización que se han creado durante el asistente. Una vez correctamente configurado el puerto de entrada, cuando se inicializa Dynamics AX devuelve la URL que se deberá agregar como referencia de servicio en la aplicación extremo para poder consumir las funciones del servicio.

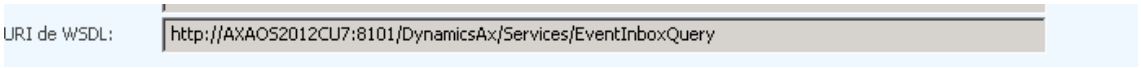



Ilustración 23. Dirección del servicio

Es interesante ver como la dirección apunta a AXAOS2012CU7:8101 (Ilustración 23) que en realidad es el servicio de AOS que ejecuta el servidor de Dynamics AX y que funcionará como servicio de Windows para los propósitos para que ha sido configurado, en este caso, devolver la lista de alertas.

### *4.1.2. Como consumir el servicio creado*

Una vez el puerto de entrada está activo, el servicio de Windows es accesible para una aplicación que esté dentro del mismo dominio. En este caso se mostrará la aplicación de consola programada en c# con la referencia de servicio para valorar las ventajas que ofrece la creación de un servicio por AIF mediante queries.

Una vez creado el proyecto de visual studio para aplicación de consola, se agrega la referencia de servicio (Ilustración 24)

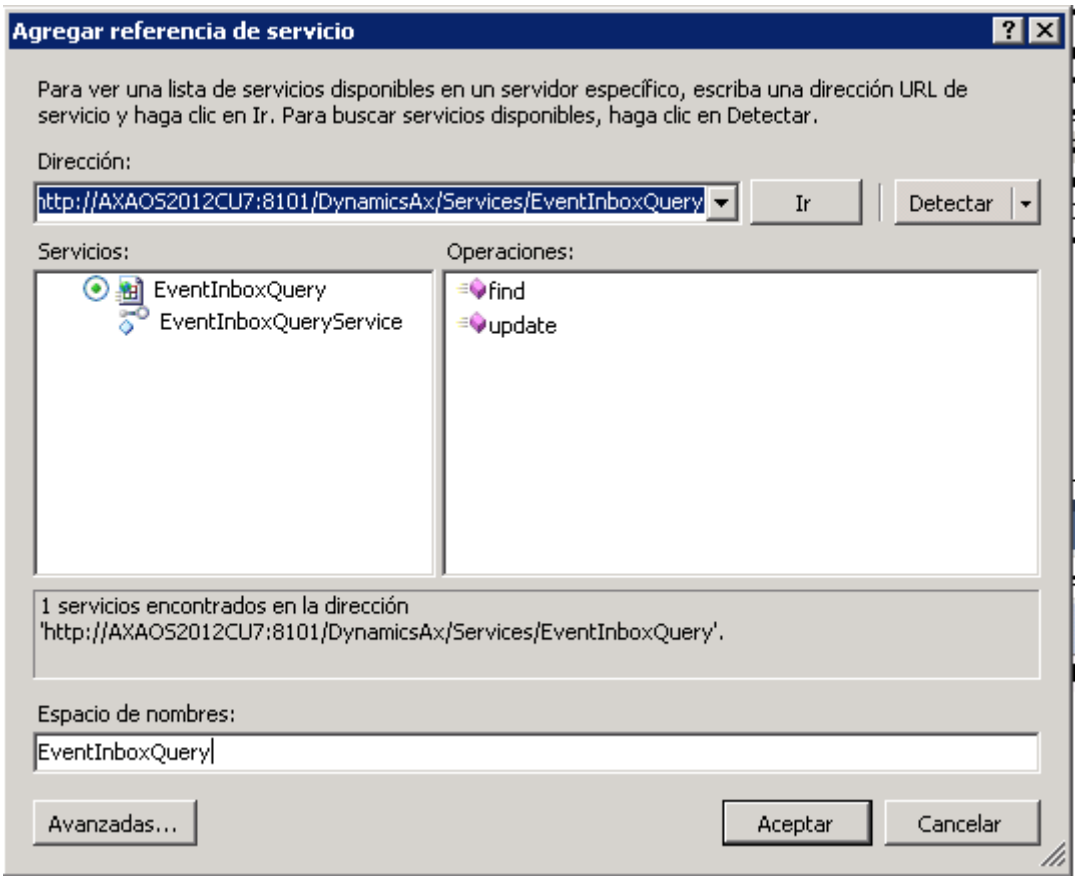



Ilustración 24. Agregar referencia del servicio creado en Dynamics AX

Donde se nos muestran los servicios y operaciones accesibles, el nombre del servicio que genera Dynamics AX coincide con el de la query "EventInboxQuery".

```
static void Main(string[] args)
{
       //LISTAR ALERTAS
       //CRITERIA ELEMENT, OBJETO PARA FILTRAR LA CONSULTA
       CriteriaElement criteriaElement = new CriteriaElement();
       //DATASOURCENAME NOMBRE DE LA TABLA EN LA QUERY QUE SE DESEA FILTRAR
       criteriaElement.DataSourceName = "EventInbox";
       //FIELDNAME, NOMBRE DEL CAMPO A FILTRAR
       criteriaElement.FieldName = "UserId";
       //VALUE1, VALOR QUE SE FILTRARÁ
       criteriaElement.Value1 = "mgracia";
       //OPERATOR, TIPO DE OPERACIÓN A USAR EN EL PREDICADO DE LA CONSULTA
       criteriaElement.Operator = Operator.Equal;
       //SE INICIALIZA EL OBJETO QUERYCRITERIA AGREGANDO LOS PARÁMETROS
ANTERIORES
       QueryCriteria query = new QueryCriteria();
       query.CriteriaElement =
       new CriteriaElement[1] { criteriaElement };
       EventInboxQueryServiceClient serviceClient = new
EventInboxQueryServiceClient();
       //SE EJECUTA EL MÉTODO DISPONIBLE EN EL SERVICIO FIND PARA LA BÚSQUEDA CON
EL OBJETO QUERY
```

```
        AxdEventInboxQuery eventInbox = serviceClient.find(null, query);

        if (eventInbox.EventInbox != null)
        {
               for (int i = 0; i < eventInbox.EventInbox.Length; i++)
              {
                     //EN CADA ITERACIÓN DEL RESULTADO SE IMPRIMEN TRES VALORES
                     Console.WriteLine(
                     String.Format("{0} - {1} - {2}",
                     eventInbox.EventInbox[i].UserId,
                     eventInbox.EventInbox[i].AlertTableId,
                     eventInbox.EventInbox[i].Subject));
              }
        }

        Console.ReadLine();
}
```

Criteria element es el objeto donde se indica la tabla (DataSourceName), campo (FieldName) y valor a filtrar (Value1), en ese caso se han filtrado las alertas del usuario “mgracia” . Una vez incializado el objeto se puede consultar el resultado que envía el servicio de AIF, de donde se puede extraer cualquier valor de los registros de alertas filtrados. El resultado es el siguiente (Ilustración 25)

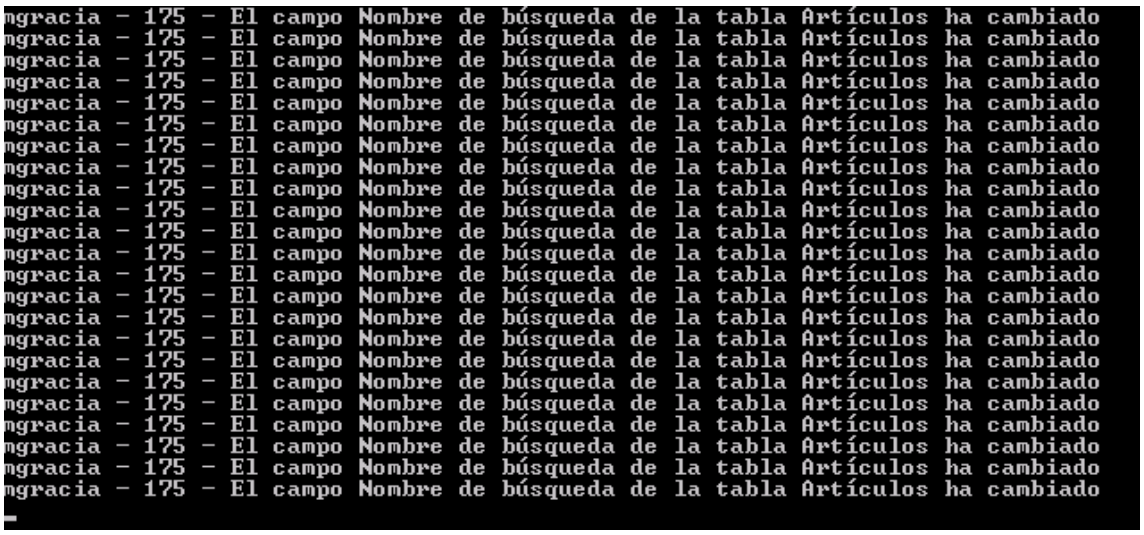



Ilustración 25. Resultado de la consulta contra el servicio en AIF basado en queries

### 4.1.3. Inconvenientes de usar AIF con queries

Pese a que logramos acceder a la información generada en Dynamics AX desde el exterior ayudados por el propio ERP, aún tenemos varios problemas:

1. Las agrupaciones vienen definidas por el campo RefTableId de la tabla eventInbox, esto obligaría a modificar la query asociada al puerto de entrada, haciendo primero un agrupado por el valor RefTableId, y luego añadir otra tabla del mismo tipo, montando la relación por RefTableId y UserId para así obtener todas las alertas de la categoría

establecida por RefTableId. O bien trabajar con dos objetos listas, una para incluir sólo las tablas vinculadas a los eventos, y otra para incluir los datos relacionados. Esto habría que hacerlo en la aplicación exterior que se encargaría de construirlas.

2. Aún nos queda el problema que no tenemos el valor de la descripción de la tabla indicada en RefTableId. AIF devuelve los valores asignados a la tabla, y tal y como se comentó, lo que contiene es una referencia al identificador de la tabla, que es numérico y no un campo descriptivo que es lo que se necesita.
3. ¿Cómo relacionar el registro que provocó el evento?, anteriormente se indicó una modificación para guardar la relación en la tabla EventInbox por el campo RecId. Pero como incluir esto en la query?, la alternativa pasaría por agregar en la query todas y cada una de las tablas mediante outer join montando la relación por RefTableId y RefRecId (que es clave primaria).
4. Aunque se lograra añadir todas las tablas en la query, ¿cuál sería el campo usado para mostrarlo en la lista tal y como se comentaba al inicio, de manera que fuera lo más descriptible posible?
5. No facilita la migración a nuevas versiones, aunque el servicio de AIF esté montado en Dynamics AX, la construcción de las listas, las búsquedas en las tablas de alertas se hace en la aplicación externa. Por lo que en caso de migrar a una nueva versión habría que volver a revisar todo el código.

Esta solución no aporta grandes mejoras con respecto a la mostrada por SQL, su resolución sigue siendo muy compleja, por lo que no facilitaría futuras modificaciones.

Sigue sin cumplir el principio de responsabilidad individual, dado que será la aplicación externa la que tiene que hacer la consulta en las tablas filtrando los datos y escogiendo los campos apropiados para devolver la información al usuario.

El objetivo que se está buscando es que sea el propio Dynamics AX quien genere las listas categorizadas para podérselas pasar a un proceso externo (del que se hablará más adelante) y las haga públicas y accesibles desde el Windows Phone.

### *4.2. AIF basado en data contracts.*

Ya se ha indicado al inicio que AIF permite la comunicación con aplicaciones externas usando protocolo SOAP y el intercambio de ficheros XML. En esta segunda opción que se muestra, y que será la escogida para la comunicación de alertas a Windows Phone, se enseñará cómo se pueden escribir objetos complejos como clases, con parámetros de entrada y salida y cómo AIF es capaz de serializarlos para usarlos en la comunicación con un servicio basado en WCF.

Esto permite simplificar el código, no necesita hacer modificaciones del código estándar al contrario que en los casos anteriores, haciendo que sea más fácilmente migrarlo en un futuro.

Antes de describir el proceso hay ciertos conceptos que se deberán aclarar:

1. Servicios, objeto usado por Dynamics AX para vincularlo con métodos de clases de manera que puedan ser usados por el programa que consuma el servicio de WCF. A estos métodos se les llamará operaciones de servicio.

2. SysEntryPointAttribute, toda operación de servicio debe tener especificado este atributo. Indica si el nivel de acceso a datos se debe validar contra los permisos del usuario que ejecuta el servicio o no.
3. AifCollectionTypeAttribute, se usa para definir el tipo de dato de entrada o salida de la operación de servicio. Si el tipo es nativo de .net como un string, integer o boolean no es obligatorio especificarlo.
4. DataContractAttribute, se especifica en la declaración de la clase para especificar que debe ser serializada. Es decir, si la clase se va a usar como un parámetro de entrada o salida que deba ser consumido en el servicio web.
5. DataMemberAttribute, para especificar que el método se debe serializar. Se puede tener una clase compleja que va a ser serializada, y aquellos métodos que se desean publicar en WCF deben tener el parámetro. Aquellos métodos que sean private no tendría sentido serializarlos.

### *4.2.1. Estructura de clases para la consulta de datos*

A continuación se muestra el diagrama de clases (Ilustración 26) y una explicación de cada una que se ha usado para montar la estructura que generará las listas agrupadas por tablas para publicarlas por AIF:

1. WebService_AlertId, clase que identifica la alerta, debe tener la propiedad DataContractAtribute :
    a. alertId, campo que identifica el registro para el usuario. Debe tener la propiedad DataMemberAttribute.
    b. AlertRecId, clave primaria del registro EventInbox. Debe tener la propiedad DataMemberAttribute.
2. WebService_AlertTypeCategory, clase que identifica una agrupación por tabla y una lista de las alertas relacionadas con la tabla. Debe tener la propiedad DataContractAtribute :
    a. alertRefTableId, id. de la tabla que generó la alerta. Debe tener la propiedad DataMemberAttribute.
    b. alertCategoryName, string que se usará para identificar la agrupación, es el nombre descriptivo de la tabla en Dynamics AX (propiedad label). Debe tener la propiedad DataMemberAttribute.
    c. webService_AlertIdList, lista de alertas asociadas a la tabla, es de tipo WebService_AlertId. Debe tener la propiedad DataMemberAttribute.
3. WebService_AlertDetail, clase que contiene todos los campos que detallan una alerta. Debe tener la propiedad DataContractAtribute
4. WebService_PushResult, clase que devuelve el resultado, debe tener la propiedad DataContractAtribute :
    a. alertTypeCategoryList, objeto de tipo lista de WebService_AlertTypeCategory. Debe tener la propiedad DataMemberAttribute.
    b. processOk, booleano que indica si el proceso ha tenido un resultado satisfactorio. Debe tener la propiedad DataMemberAttribute.

    c. processTxt, string que contiene una descripción clarificadora en caso que el resultado no haya sido satisfactoria, como por ejemplo que el usuario no se haya identificado correctamente. Debe tener la propiedad DataMemberAttribute.
5. WebService_GetUserListAlerts, clase que valida el usuario que pide los datos y los devuelve en un objeto de tipo WebService_PushResult, al contrario que las otras clases, no hará falta serializarla entera:
    a. getUserAlert, función que buscará las alertas asociadas al usuario, validará los parámetros de entrada de usuario y password y devolverá un objeto de tipo WebService_PushResult. Debe tener la propiedad SysEntryPointAttribute, dado que será una operación de servicio en el AIF.
    b. validateUserPassword, método público que valida el usuario y password que entran como parámetros. Debe tener la propiedad SysEntryPointAttribute, dado que será una operación de servicio en el AIF.
    c. getAlertIdList, método que devuelve una lista de WebService_AlertId, las alertas que devuelve estarán asociadas al usuario y tabla que tiene como parámetro de entrada. El método no es accesible desde el servicio, es para uso interno de la clase.
    d. commonIdFromEventInbox, tiene como parámetro de entrada un registro de EventInbox, devuelve el string identificador asociado a la tabla que generó la alerta, es decir para, la tabla de clientes devuelve el identificador de cliente, para la tabla de artículos el identificador del artículo. El método no es accesible desde el servicio, es para uso interno de la clase.

¿Cuál es el objetivo de toda esta estructura?:

1. Hacer la búsqueda de alertas desde el propio Dynamics AX, ya se ha demostrado anteriormente qué es más sencillo e implica cero modificaciones del código estándar, lo que mejora futuras adaptaciones o migraciones.
2. Principio de responsabilidad individual, es el propio Dynamics AX quien se encarga de montar las estructuras de datos necesarias para que la aplicación de Windows Phone únicamente las tenga que consumir y publicar. Si en un futuro la estructura de datos donde se originan las alertas se modificara, únicamente habría que adaptar la manera en que se informan los datos que se publica en la aplicación móvil, con lo que en este último no habría necesidad de hacer modificaciones.
3. Serializar un conjunto de clases para simplificar la presentación de la información, WebService_AlertId contiene una alerta, WebService_AlertTypeCategory contiene una categoría y una lista de alertas asociadas a esa categoría, WebService_PushResult el resultado de la consulta de alertas, una lista de categorías y cada una con sus alertas correspondientes.

Con estas estructuras, la aplicación de Windows Phone es capaz de montar una lista de alertas por categoría, enseñando en cada una, otra lista con un identificador que el usuario es capaz de vincular a la categoría y una breve descripción.

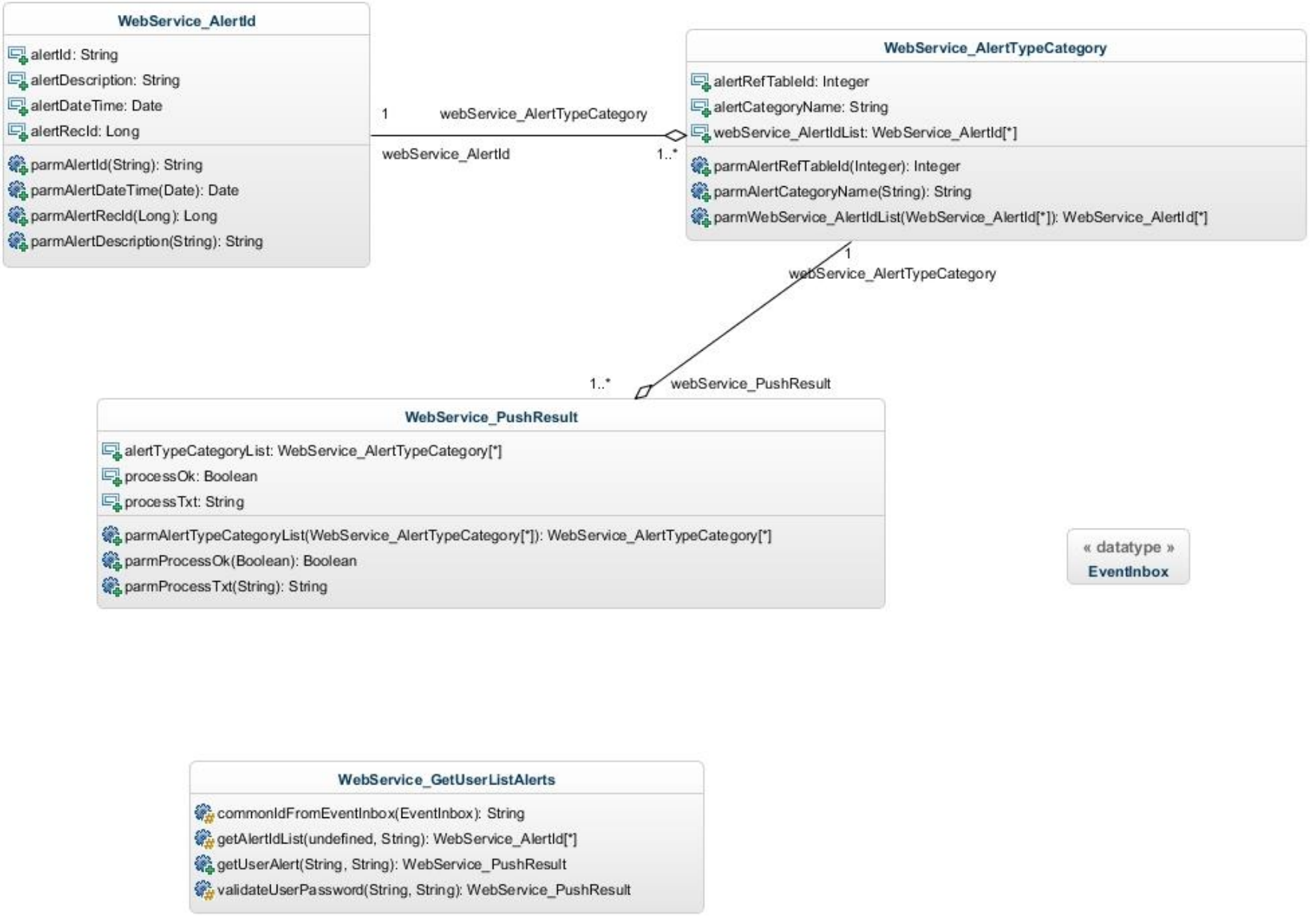



Ilustración 26. Diagrama de clases para la lista categorizada de alertas

### 4.2.2. Algoritmo para obtener la lista de alertas

El método getUserAlert de la clase WebService_GetUserListAlerts es el que se llamará desde la aplicación móvil, el algoritmo (Ilustración 27) es el siguiente:

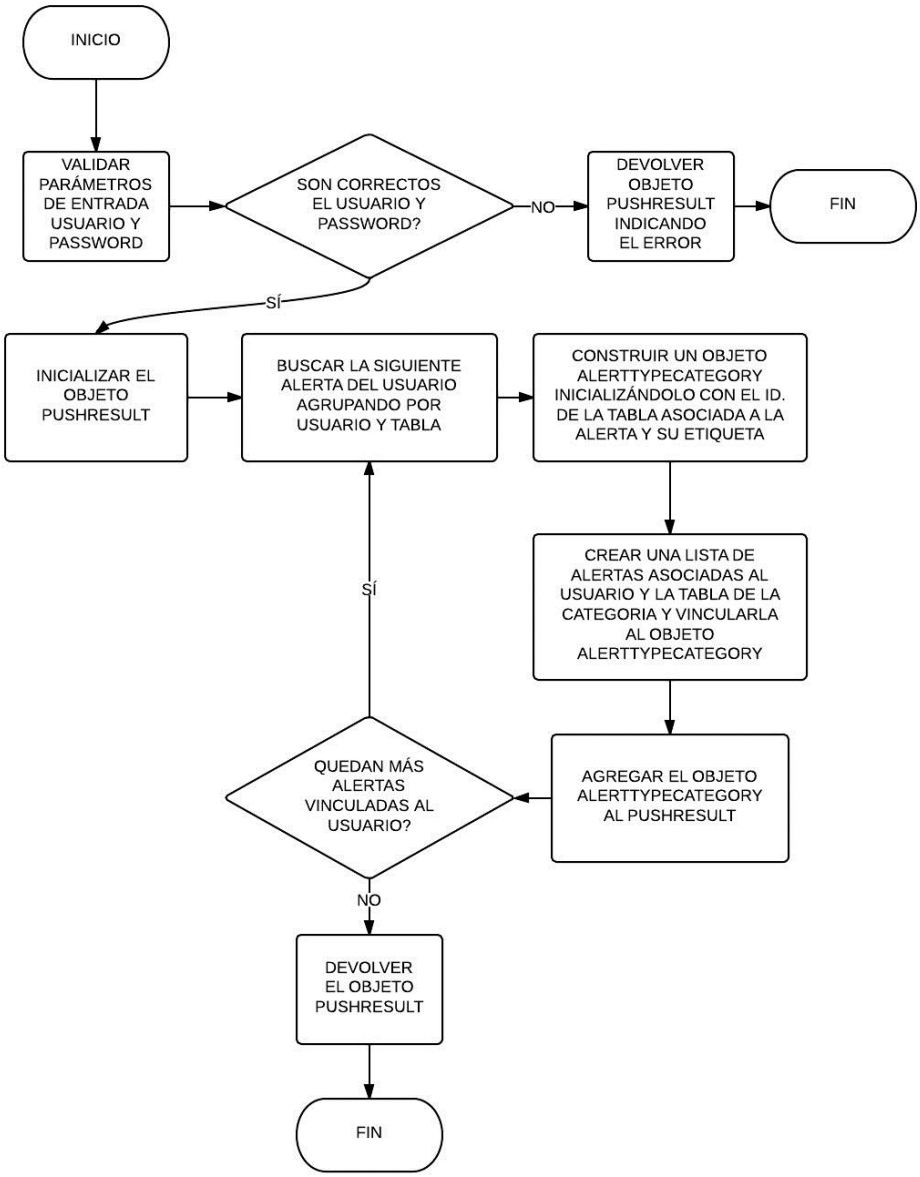



Ilustración 27. Algoritmo para la consulta de la lista de alertas del usuario

### 4.2.3. Como consumir el servicio creado.

Al igual que en el ejemplo mostrado en el punto 4.1.2 se agregará en un proyecto de visual studio 2010 una referencia de servicio, en este caso el servicio de AIF no está montado mediante queries, si no con la estructura de clases presentadas en el punto 4.2.1, estas clases serán serializadas, por lo que podrán ser usados en el siguiente programa realizado en c#:

```
static void Main(string[] args)
{
//CREACIÓN DEL CALLCONTEXT PERMITE EDITAR LAS CREDENCIALES PARA IDENTIFICARSE EN DYNAMICSAX
EventInboxDataContract.CallContext callContext = new EventInboxDataContract.CallContext();
//CREACIÓN DE LA CLASE WebService_GetUserListAlerts
EventInboxDataContract.WebService_GetUserListAlertsClient getUserListAlertsClient = new
EventInboxDataContract.WebService_GetUserListAlertsClient();
//EN EL OBJETO PushResult SE GUARDA EL RESULTADO DE LLAMAR AL MÉTODO getUserAlert DE LA CLASE
WebService_GetUserListAlertsClient
EventInboxDataContract.WebService_PushResult pushResult =
getUserListAlertsClient.getUserAlert(callContext, "01", "2807");
//SI EL RESULTADO DEVUELTO EN PushResult ES TRUE (ES DECIR, NO HAY ERRORES)
if (pushResult.ProcessOk == true)
{
        //PARA CADA WebService_AlertTypeCategory QUE SE ENCUENTRA EN PushResult
        foreach (EventInboxDataContract.WebService_AlertTypeCategory alertTypeCategory in
pushResult.AlertTypeCategoryList)
        {
                //IMPRIMIMOS LA CATEGORIA POR PANTALLA
                Console.WriteLine(alertTypeCategory.alertCategoryName);
                //POR CADA WebService_AlertId INCLUIDO EN LA CATEGORÍA
                foreach (EventInboxDataContract.WebService_AlertId alertId in
        alertTypeCategory.AlertIdList)
                {
                        //IMPRIMIMOS INFORMACIÓN DE LA ALERTA
                        Console.WriteLine("ID: " + alertId.AlertId + " D: " +
                alertId.AlertDescription);
                }
        }
}
//SI PushResult DEVUELVE FALSE
else
{
        //IMPRIMIMOS EL ERROR DEVUELTO POR EL SERVICIO
        Console.WriteLine(pushResult.ProcessTxt);
}

Console.ReadLine();
}
```

El resultado de ejecutar este sencillo programa es el siguiente

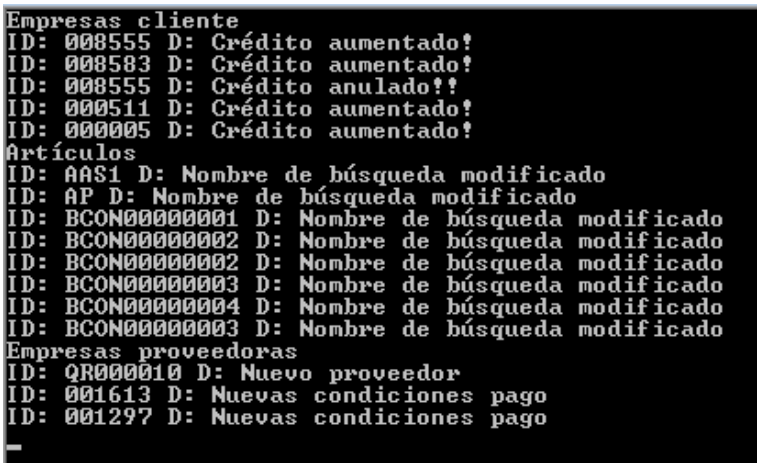



Ilustración 28. Resultado de la consulta contra el servicio en AIF basado en data contracts

A diferencia del ejemplo del punto 4.1.2 éste es más sencillo e intuitivo al estar basado en una estructura de clases que han sido diseñadas específicamente para obtener un objetivo en concreto.

# 5. Comunicación Dynamics AX Windows Phone.

Ya se ha mostrado en los puntos anteriores como la mejor manera de hacer accesible los datos de alertas para una aplicación externa es mediante el uso de AIF. La cuestión ahora es cuál va a ser el canal intermediario que haga de puente entre la aplicación de Windows Phone y Dynamics AX (para el intercambio de datos) teniendo en cuenta que ambos pueden usar WCF.

## *5.1. Opciones.*

Se valoran tres opciones:

1. Virtual private network (VPN), ofrece muchas ventajas, es una conexión punto a punto, haciendo que el cliente forme parte del dominio donde se está conectando, las comunicaciones están cifradas. De hecho el cliente obtiene todas las ventajas de pertenecer al dominio de su red, con lo cual, una vez esté dentro de la red se puede atacar al servicio de Windows creado por el AIF.
2. Servicio web, permite la comunicación entre aplicaciones usando protocolo SOAP que se basa sobre el intercambio de datos en formato XML. La ventaja es que tanto AIF como Windows Phone soportan esta tecnología. Para que las comunicaciones sean seguras se puede usar un certificado que será emitido por el servidor donde está alojado el servicio web, de manera que las comunicaciones estarán cifradas.
3. Hacer accesible la máquina donde está el servicio de AIF público desde el exterior, de manera que se pueda consumir el servicio directamente.

La comunicación por VPN se descarta por la siguiente razón, la aplicación se ha desarrollado con la versión 7.1 de Windows Phone que no soporta conexión por VPN, esta sí está soportada en la nueva versión 8.1 (que ha sido liberada durante el desarrollo de este proyecto).

La comunicación directa con el servicio de AIF se descarta porque:

1. La comunicación no es segura, la información no va cifrada.
2. Se está haciendo accesible desde fuera, la máquina donde está alojado Dynamics AX, lo que supone un riesgo inasumible en cuanto a la seguridad de la aplicación. Se podría dejar inaccesible la máquina mediante ataques por denegación de servicio, lo que implicaría poder dejar la aplicación fuera de servicio.

Son por estas razones que la opción a escoger es el servicio web para la comunicación entre Dynamics AX y la aplicación de Windows Phone.

Antes de explicar cómo instalar y configurar un servicio web, es necesario entender cómo autentifica un usuario Dynamics AX, ya que hay dos maneras de montar el servicio web, y se escogerá una de las dos opciones mostrando las razones.

Todo usuario se autentifica en Dynamics AX con un usuario de dominio, por lo tanto, todo usuario que ejecute el servicio web debe ser un usuario de dominio dado de alta en Dynamics AX y las alertas van siempre asociadas al usuario. Esto ayudará a entender la opción escogida.

## 5.2. Instalación de un servicio web en un servidor de IIS con Dynamics AX 2012.

La primera opción que se va a mostrar es una solución 100% estándar que viene con Dynamics AX, y es que este proporciona las herramientas necesarias para implementar un servicio web en un servidor web.

## 5.3. Configurar servicio web en Windows Server 2008

Como pre-requisito será necesario disponer de un servidor de internet information services (IIS) 6.1 sobre Windows server 2008.

Los pasos a seguir son:

1. Sobre el administrador de la máquina, en el servidor web (iis), escoger añadir rol

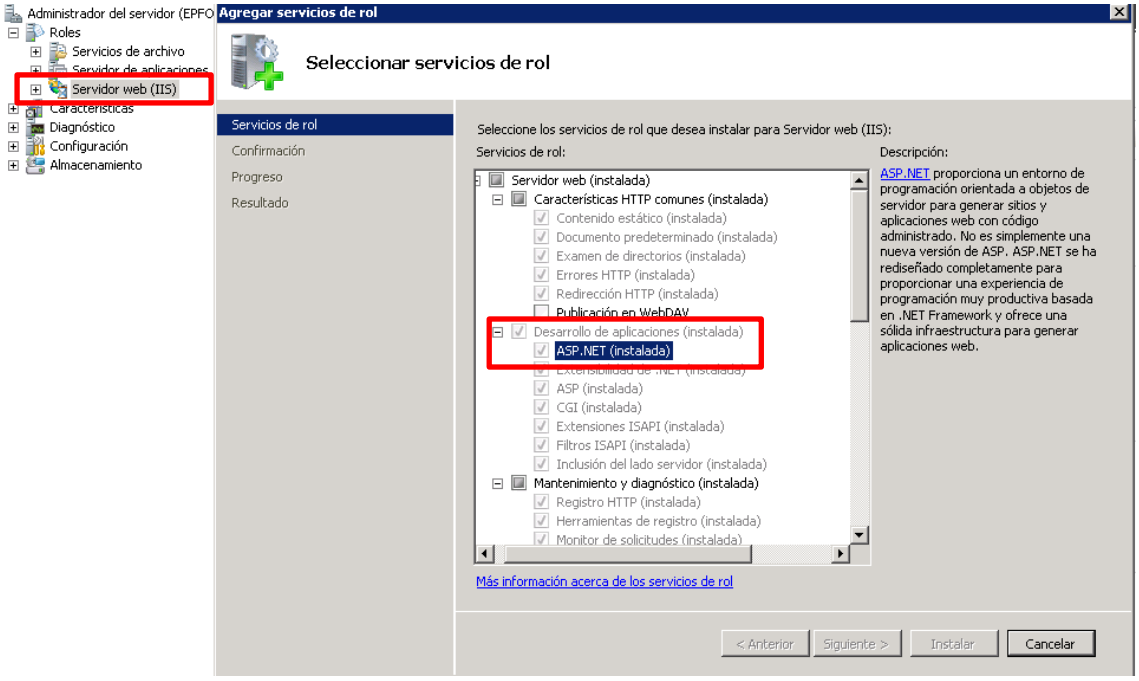



Ilustración 29. Configuración de IIS

Y seleccionar “Desarrollo de aplicaciones” y “ASP.NET” (Ilustración 29).

2. Crear un sitio web (Ilustración 30), más adelante se usará este sitio web para instalar los servicios web que se programen en Dynamics AX.

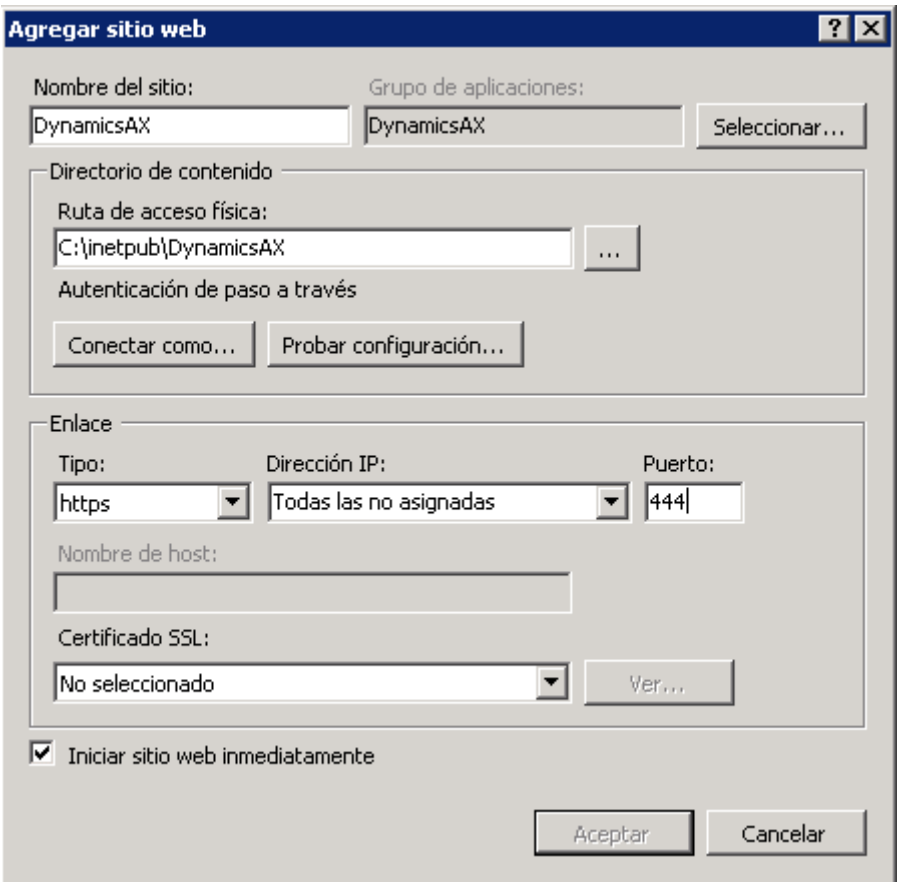



Ilustración 30. Creación del sitio web

En este caso el enlace es de tipo https, la seguridad se detallará más adelante, el puerto es el 444.

3. Ahora hay que instalar los componentes de Dynamics AX (Ilustración 31) necesarios para que se puedan instalar los servicios web programados desde el propio Dynamics AX. Desde el instalador hay que seleccionar agregar componentes

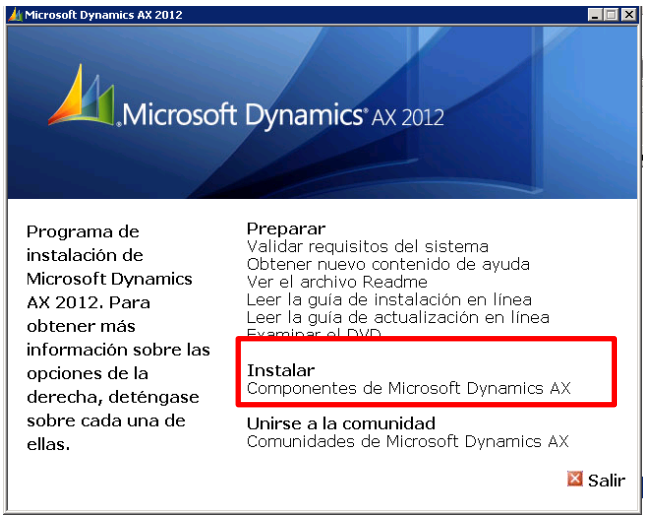



Ilustración 31. Asistente de componente de Dynamics AX

Y seleccionar el componente “Servicios web en IIS” (Ilustración 32)

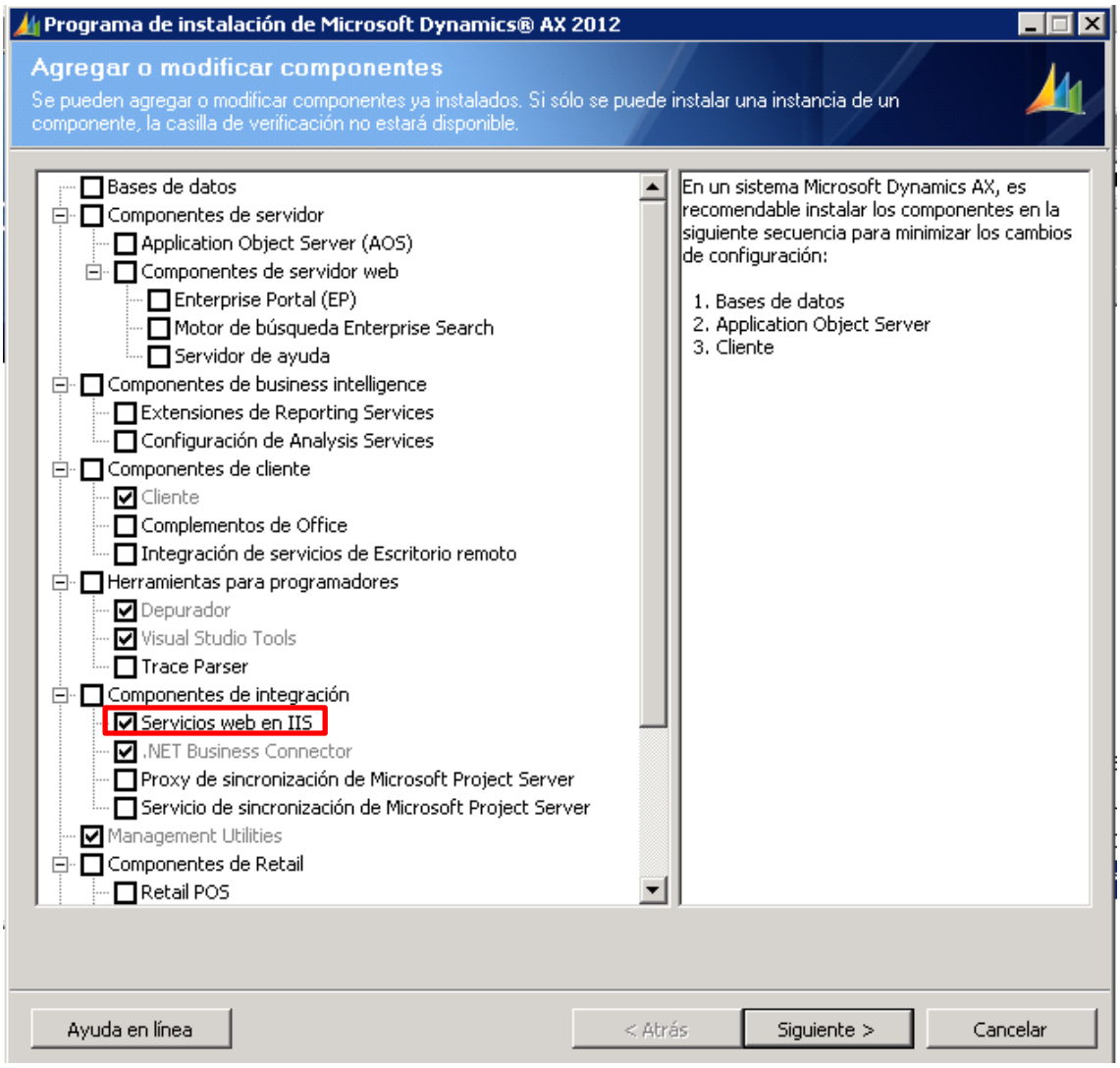



Ilustración 32. Seleccionar componente de servicios web en IIS

Una vez instalado, el asistente nos pide el password del usuario de business connector (Ilustración 33), este usuario habrá de configurarse para tener permisos para acceder al servidor web, ya que será el encargado de instalar las dll’s necesarias para el funcionamiento del servicio web que se programe en Dynamics AX

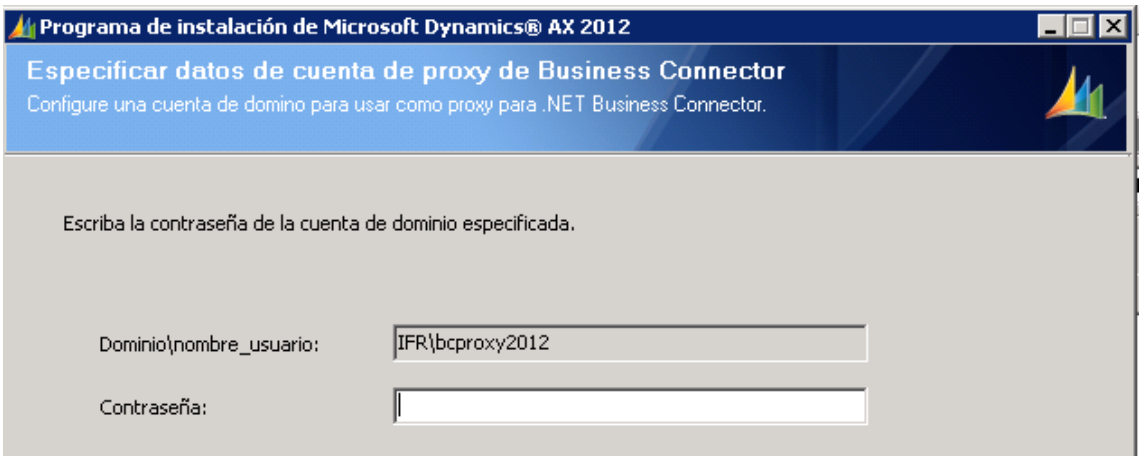



Ilustración 33. Configuración del usuario de business connector

Una vez introducido el password, se muestra el último paso del asistente (Ilustración 34) donde se especifica el sitio web que se acaba de crear, el grupo de aplicaciones que se vinculará al sitio web i el directorio virtual donde se guardarán las dll's generadas por Dynamics AX

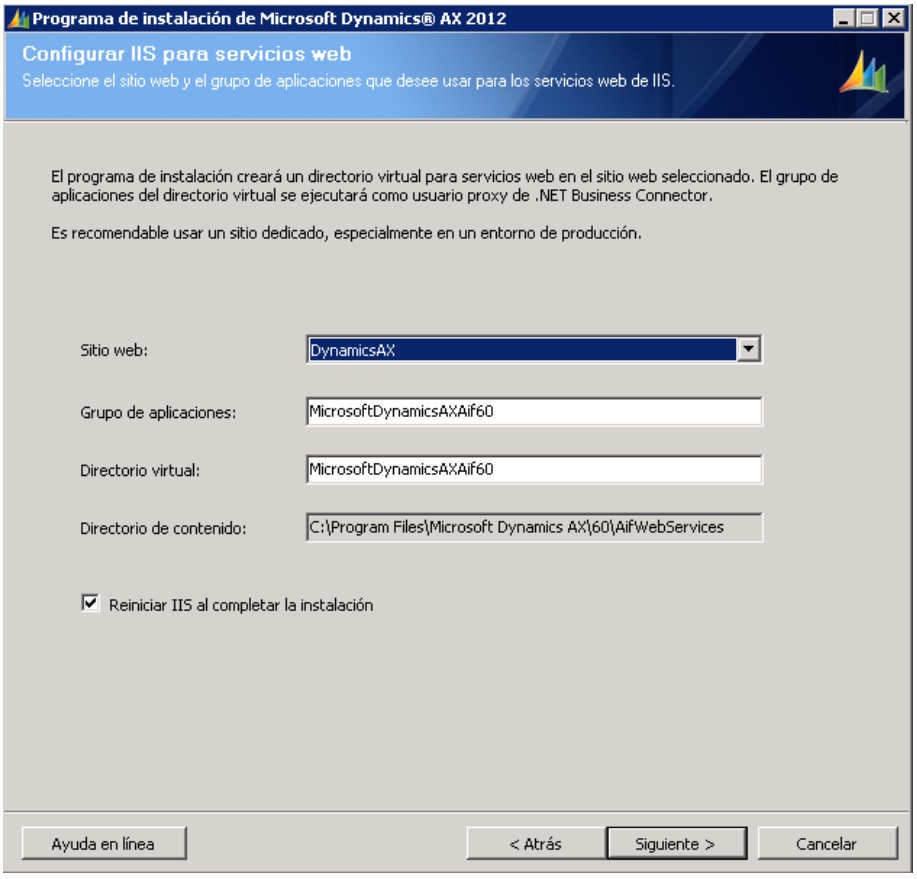



Ilustración 34. Finalización del asistente

Los nombres por defecto del grupo de aplicaciones y el directorio virtual donde se ubicarán las dll's de los servicios web no hace falta cambiarlos.

Una vez finaliza el asistente, Dynamics AX ya está preparado para instalar los servicios web desde el propio ERP. Se puede observar que el instalador ha realizado varios cambios (Ilustración 35):

En el servidor web se ha creado un nuevo grupo de aplicaciones, con el mismo nombre que se ha definido durante el asistente, y el directorio virtual donde Dynamics AX dejará las dll's necesarias para ejecutar el servicio web.

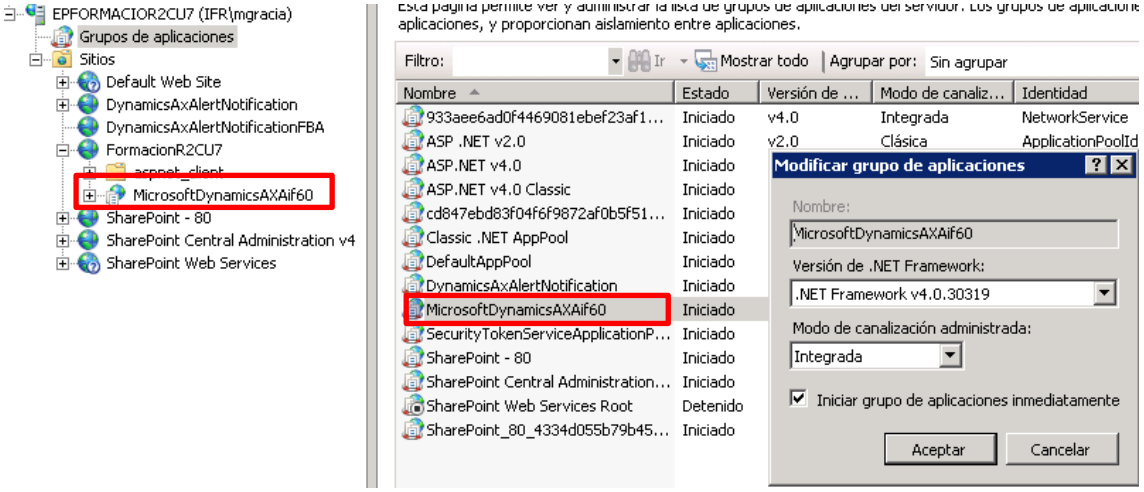



Ilustración 35. Grupo de aplicaciones

## 5.4. Configurar el servicio web en Dynamics AX 2012

A continuación se detalla como configurar Dynamics AX 2012 para poder publicar los servicios web en el servidor configurado en el punto anterior.

En Dynamics AX, en el punto de menú que se encuentra en /Administración del sistema/Configurar/Services and Application Integration Framework/Sitio web

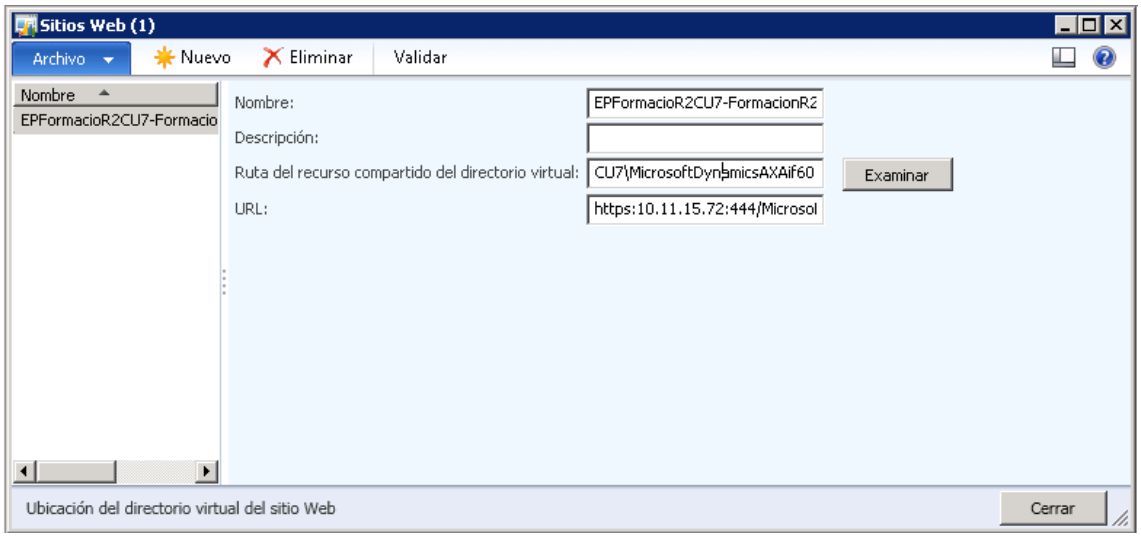



Ilustración 36. Configurar sitio web en Dynamics AX

se encuentra la configuración (Ilustración 36) necesaria para crear el servicio web, el directorio virtual donde dejará los archivos necesarios para la ejecución del servicio y una URL que usará como base para montar cada una de las URL's que se generarán por cada servicio web. Esta configuración ha sido generada por el asistente de instalación de componentes ejecutado en el punto anterior.

Por último, hay que crear el puerto de entrada. Lo primero será definir las operaciones que se harán públicas al servicio, antes se mostró, en la estructura de clases que se va a utilizar para montar el servicio web, que eran dos: "getUserAlert" y "validateUserPassword", razón por la cual tienen la propiedad SysEntryPointAttribute. Esto permitirá crear el grupo de servicios (Ilustración 37)


Ilustración 37. Grupo de servicios

En el grupo de servicios se define la clase que se va a usar y los métodos que van a publicarse en el servicio web.

Al crear el puerto de entrada (Ilustración 38) hay que darle un nombre y seleccionar el sitio web configurado anteriormente. Con lo que automáticamente se crea la URL para el servicio web.

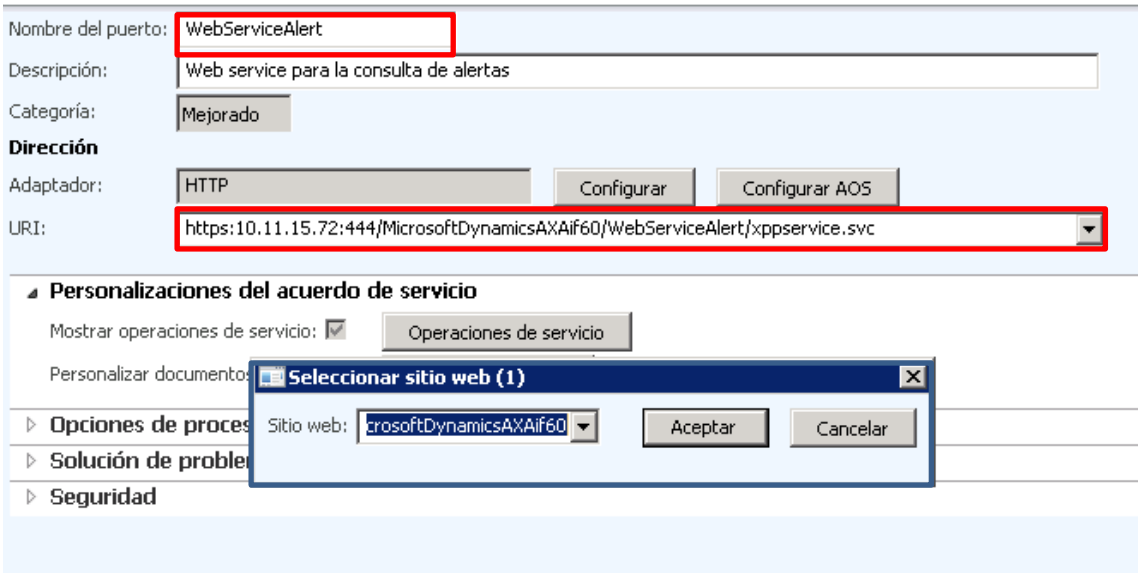



Ilustración 38. Puerto de entrada

E incluir las operaciones de servicio (Ilustración 39) configuradas anteriormente

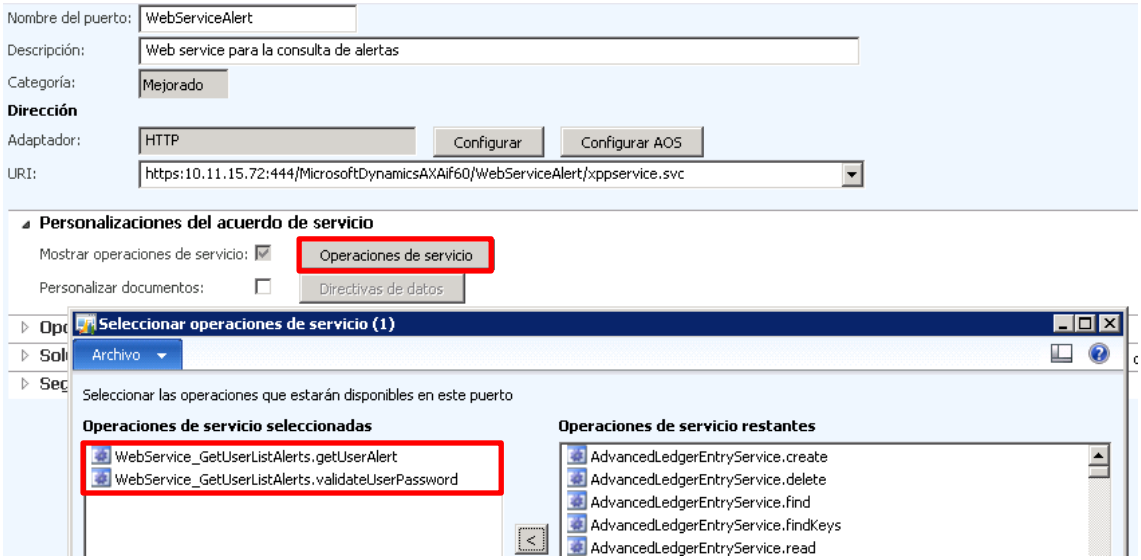



Ilustración 39. Operaciones de servicio vinculadas al puerto de entrada

De esta manera, el puerto de entrada genera una URL donde es accesible el servicio programado en AX para la obtención de una lista de alertas vinculadas a un usuario, este servicio puede ser consumido por un programa externo (mediante protocolo SOAP), en este caso, por la aplicación de Windows Phone.

### *5.5. Configuración del web.config en Dynamics AX 2012*

Por último, hay que configurar el web.config, es decir, el archivo de configuración del servicio web, esta parte se realiza con una interfaz desde el propio puerto de entrada, seleccionando la opción Configurar.

En la parte de enlaces (binding), crearemos uno nuevo “DynamicsAX” de tipo basichttpbinding (Ilustración 40)

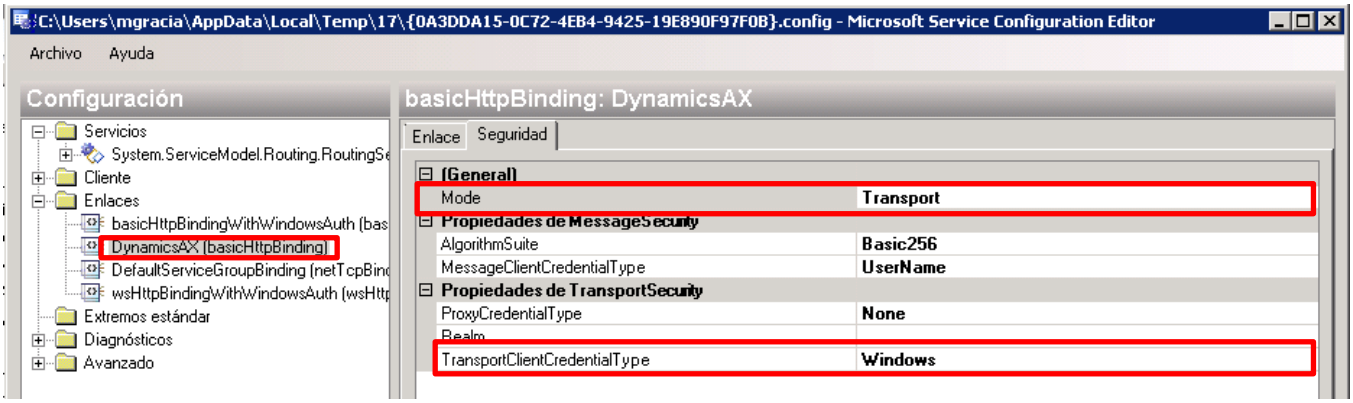



Ilustración 40. Configuración del binding

Por defecto la clase basicHttpBinding no lleva seguridad implementada, para ello se ha de especificar en el modo de seguridad en “Transport”, las credenciales de cliente se establece en “Windows” dado que los usuarios de Dynamics AX se autentifican con usuarios de dominio.

En el EndPoint (donde se configura cómo se va a realizar la comunicación) se asigna el binding creado (Ilustración 41)

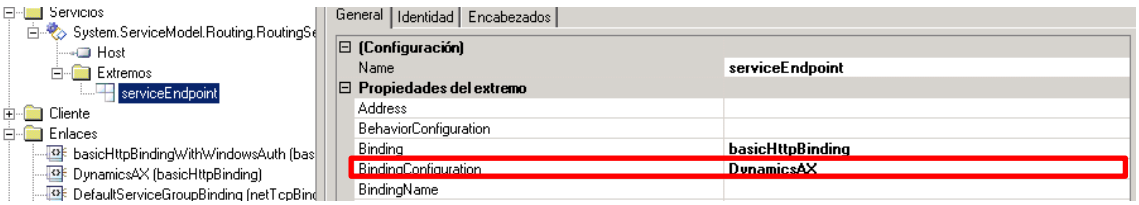



Ilustración 41. Configuración del EndPoint

En el servicebehaviour (donde se define el comportamiento del servicio) hay que especificar que el servicio acepta conexiones por https (Ilustración 42) y no por http.

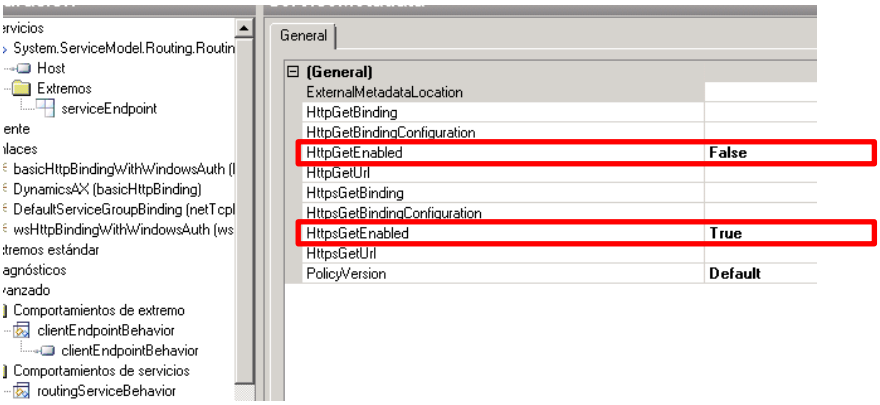



Ilustración 42. Configuración por https

Una vez activado el puerto, el servicio web está accesible y se puede consumir en una aplicación de consola o por el propio Windows Phone.

### *5.6. Consumir el servicio web.*

A continuación se muestra una aplicación de consola programada en C# que va a consumir el servicio.

Se mostrará cómo están disponibles las funciones configuradas en el puerto de entrada configurado en Dynamics AX, y cómo las clases mostradas en el diagrama de clases son accesibles para poder mostrar una lista categorizada de alertas de una forma mucho más sencilla que la primera opción por queries que se mostró.

El primer paso es agregar la referencia al servicio web por la url que devolvió el puerto de entrada (Ilustración 43). Se mostrarán entonces las funciones que se configuraron

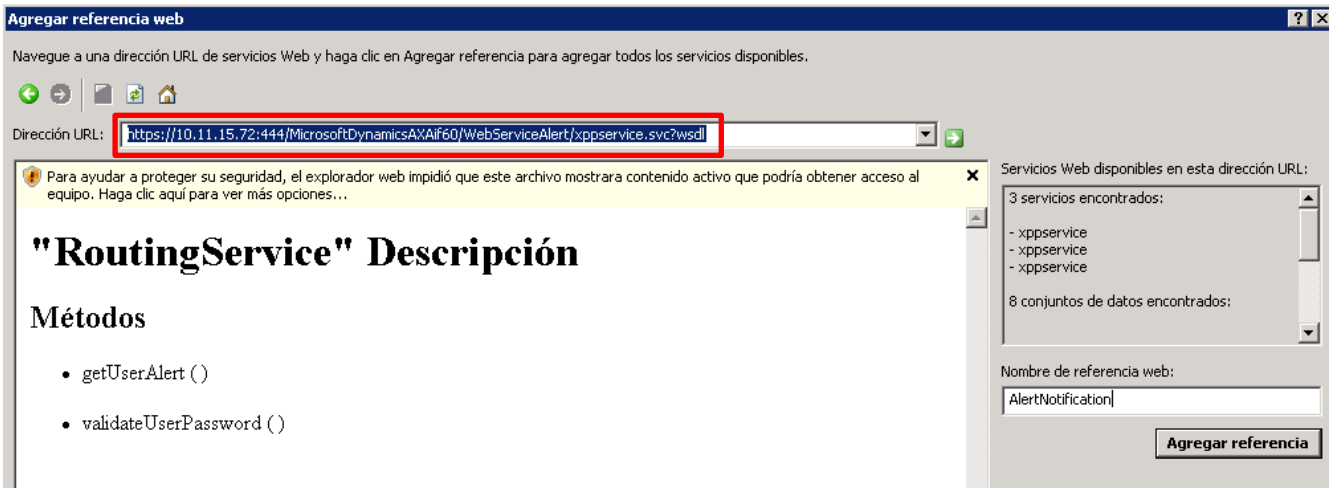



Ilustración 43. Funciones publicadas por el servicio web

Y las clases que se serializaron para poder consultar las alertas con sus métodos y variables públicas (Ilustración 44)

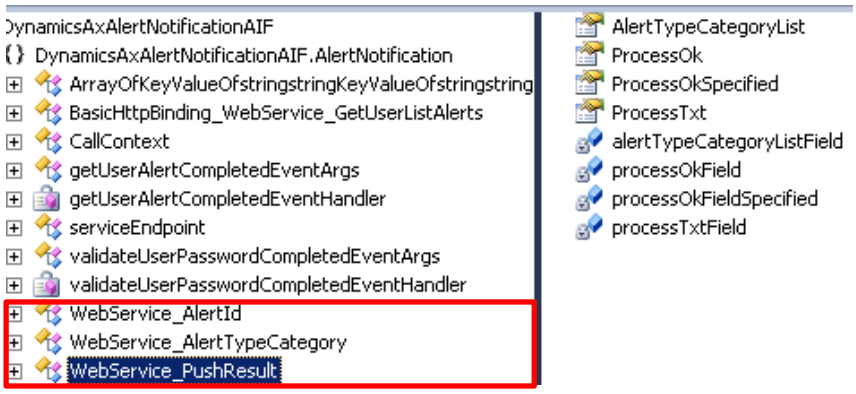



Ilustración 44. Objetos publicados por el servicio web

Lo que permite consultar los datos de una forma sencilla y rápida

```
//SE INICIALIZA EL CONTEXTO
DynamicsAxAlertNotificationAIF.AlertNotification.CallContext callContext = new
DynamicsAxAlertNotificationAIF.AlertNotification.CallContext();
//SE CREA EL CLIENTE QUE VA A CONSUMIR EL SERVICIO
DynamicsAxAlertNotificationAIF.AlertNotification.BasicHttpBinding_WebService_GetU
serListAlerts client = new
DynamicsAxAlertNotificationAIF.AlertNotification.BasicHttpBinding_WebService_GetU
serListAlerts();
//SE PASAN LAS CREDENCIALES CON UN USUARIO DE DOMINIO
System.Net.NetworkCredential credentials = new
System.Net.NetworkCredential("USUARIO", "PASSWORD", "DOMINIO");
```

```
client.Credentials = credentials;
//SE EJECUTA LA FUNCIÓN DE CONSULTA DE ALERTAS
DynamicsAxAlertNotificationAIF.AlertNotification.WebService_PushResult result =
client.getUserAlert("01", "2807");

foreach
(DynamicsAxAlertNotificationAIF.AlertNotification.WebService_AlertTypeCategory
category in result.AlertTypeCategoryList)
{
       //PARA CADA CATEGORÍA SE IMPRIME SU DESCRIPCIÓN
       Console.WriteLine(category.alertCategoryName);
       foreach
       (DynamicsAxAlertNotificationAIF.AlertNotification.WebService_AlertId
       alertId in category.AlertIdList)
       {
              //Y POR CADA CATEGORÍA SE IMPRIMEN LAS ALERTAS ASOCIADAS
             Console.WriteLine("ID: " + alertId.AlertId + " D: " +
             alertId.AlertDescription);
       }

}
Console.WriteLine("FIN");
Console.ReadLine();
```

El resultado

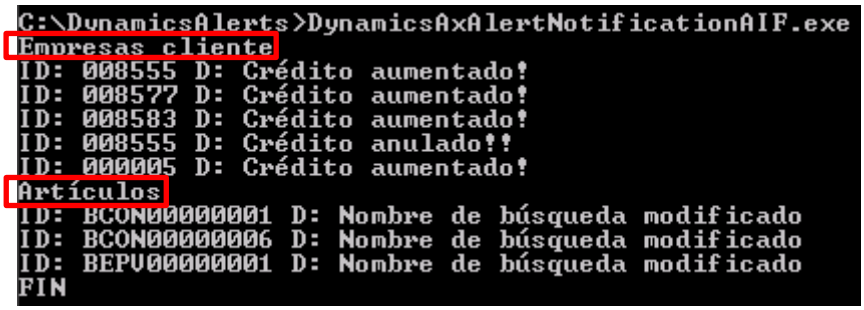



Ilustración 45. Resultado al consultar las alertas usando el servicio web

Se puede observar que hay dos agrupaciones: "Empresas cliente" y "Artículos" y que para cada una de las agrupaciones se listan las alertas asociadas (Ilustración 45).

Básicamente hay 3 pasos:

1. Abrir la conexión con el servicio web y pasarle usuario y password de dominio para identificarse en Dynamics AX.
2. Ejecutar la función getUserAlert que nos devuelve un objeto de tipo WebService_PushResult
3. Para cada categoría de alerta encontrada en el paso 2 se imprime la descripción.
4. Para cada alerta contenida en la categoría del paso 3 se imprime identificador y descripción.

Este método es mucho más sencillo que hacerlo mediante objeto queries, dado que la información se devuelve ya estructurada para mostrarse en la aplicación del otro extremo (en este caso la aplicación de Windows Phone).

Pese a que esta parece la solución a aplicar aún existe un problema que es inherente a Dynamics AX, y es el hecho que hay que autentificarse mediante un usuario de dominio.

¿Por qué ha de suponer esto un problema?, un usuario de dominio puede tener permisos de administrador para acceder a otras máquinas, puede que si se use correo con servidor de Exchange, la cuenta de usuario sirva para acceder al correo electrónico.

Hay que tener en cuenta que en la aplicación móvil habrá que configurar la cuenta de dominio para poder acceder al servicio web para poder obtener los datos, dado que cada usuario querrá consultar sus alertas, si esta información se viera comprometida, terceras partes podrían tener acceso a datos restringidos.

En el siguiente capítulo se mostrará la opción definitiva que se va a usar en el desarrollo del proyecto.

# 6. Instalación de un servicio de Windows Comunication Foundation sobre un servidor IIS con referencia a un servicio de AIF

El objetivo de esta solución es que el usuario que se identifique en la aplicación móvil no tenga que usar su usuario de dominio, algo que es obligatorio en caso de escoger la opción explicada anteriormente.

Existen distintas maneras de autentificarse en un servidor de IIS:

1. Anónima, cualquier usuario puede acceder al servicio web, pero aun así para consumir el servicio de AIF hay que identificarse en Dynamics AX con un usuario de dominio.
2. Básica, se puede identificar en el servicio web usando un usuario que esté dado de alta en la máquina que hospeda el servicio web. Este usuario no es de dominio, pero aún así sigue siendo necesario el usuario de dominio para acceder a Dynamics AX. Esto obligaría a configurar dos usuarios y contraseñas en la aplicación móvil.
3. Windows, la que usa por defecto AIF cuando se instala sobre un servidor web. Se usa un usuario de dominio.
4. FBA, form-based authentification, el propio IIS es capaz de crear una base de datos para guardar usuarios que validará para acceder al servicio web. Pero aún así sigue siendo necesario el usuario de dominio para acceder a Dynamics AX. Esto obligaría a configurar dos usuarios y contraseñas en la aplicación móvil.

Siempre tenemos el mismo problema, hay que identificarse con un usuario de dominio para acceder a Dynamics AX.

Usar las opciones anteriores, a excepción de la primera, obliga a identificarse dos veces, la primera contra el servicio web y la segunda contra Dynamics AX, el objetivo es autentificarse con un único usuario y password.

Para poder llegar a la solución se va a realizar lo siguiente:

1. Crear un servicio WCF sobre un sitio web en un servidor de IIS.
2. Agregar seguridad mediante el uso de un certificado y protocolo SSL para cifrar las comunicaciones.
3. Agregar el puerto de entrada creado mediante AIF como referencia de servicio en el WCF mediante NetTcp para la consulta de alertas.
4. Identificarse contra el servicio web mediante la impersonación usando un usuario de dominio con seguridad restringida.
5. Crear una nueva estructura de usuario y contraseña en Dynamics AX para solicitar los datos desde la aplicación móvil.
6. Solicitar las alertas vinculadas al usuario consumiendo el servicio de AIF que se encuentra en el WCF creado en el primer punto.

Lo que se consigue de esta manera, es que el servicio de WCF sea siempre ejecutado por el usuario configurado en el punto 4, esta parte la realiza el servidor de IIS automáticamente.

De esta manera se logra acceder a Dynamics AX, para que cada usuario pueda consultar sus alertas de forma segura, se usa la identificación mediante la estructura de datos creada en el punto 9.2.1.

Así el usuario y password que se deberá configurar en la aplicación móvil, no será un usuario de dominio, ese usuario sólo servirá para funcionar contra el servicio web para la consulta de alertas. Se aumenta de esta manera la seguridad.

### *6.1. Seguridad en Dynamics AX*

Tal y como se ha explicado en el punto anterior la autentificación en el servicio web va a ser mediante la impersonación de un usuario con seguridad restringida. Es decir, todos los usuarios que se conecten al servicio web van a acceder a Dynamics AX con el mismo usuario.

Una vez el usuario esté conectado ha de poder consultar sus alertas, y este acceso se ha de poder restringir, para que no se pueda acceder a los datos de otros usuarios, por lo tanto se va a crear una tabla nueva para poder autentificarse con este servicio, esta tabla "PushUserTable" contendrá los siguientes campos:

1. UserId, usuario estándar de Dynamics AX, campo obligatorio y clave primaria.
2. PushUserId, campo obligatorio, identificador de usuario de aplicación móvil, se crea un índice que no permita duplicados.
3. PushPassword, campo obligatoro, password asociado al identificador del usuario de la aplicación móvil.
4. PushURI, URI del móvil asociado al usuario.

También se creará la tabla "PushUserTableLog" con los siguientes campos:

1. PushUserTable, de tipo Int64 para relacionarlo con la tabla PushUserTable.
2. Log, campo de tipo texto que contiene un log que se detalla en el punto 7.5.2
3. CreatedDateTime, campo que informa de la fecha y hora en que se insertó el registro.

El objetivo de estas tablas, con su formulario asociado, es que un usuario administrador pueda dar de alta usuarios para la aplicación móvil y asociarlos a usuarios de Dynamics AX, aparte de poder hacer un seguimiento por si se ha producido un error vinculado al usuario y la APP. De esta manera, la aplicación móvil se identificará mediante un usuario que no es de dominio y devolverá los datos asociados al usuario estándar de Dynamics AX que tenga vinculado.

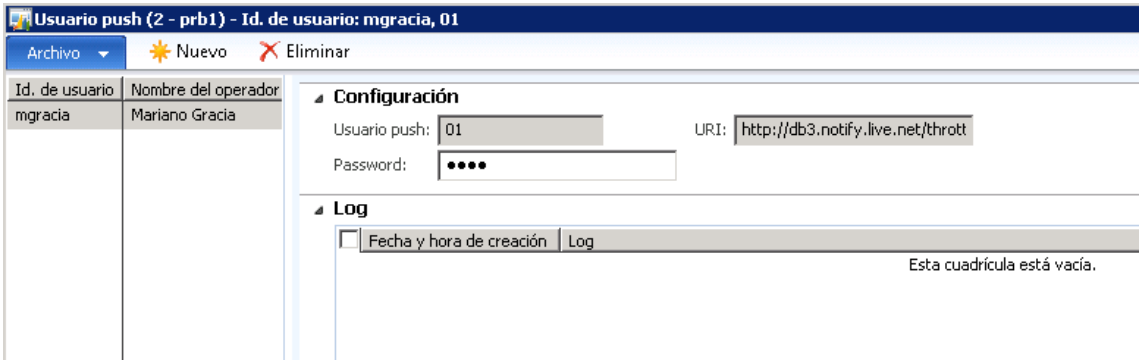



Ilustración 46. Usuario Push

## 6.2. Crear el sitio web.

El primer paso a realizar es crear el sitio web donde se pondrá el servicio de WCF que consumirá la aplicación de Windows Phone.

Antes de crear el sitio web se debe crear un nuevo grupo de aplicaciones (application pool).

Este concepto es particular del internet information services (IIS). IIS es capaz de manejar peticiones de código HTML realizadas por los usuarios, siempre que se hable de código HTML estático. Cuando se trata de código dinámico escrito, por ejemplo, en ASP.NET, entra en juego el application pool, cada sitio web hospedado en el IIS puede tener el suyo propio, esto permite tener distintas configuraciones para cada sitio web, e incluso permite aislar los procesos, de manera que código malicioso o incorrecto no afecte a los demás sitios web.

En el grupo de aplicaciones de IIS se creará uno nuevo (Ilustración 47):

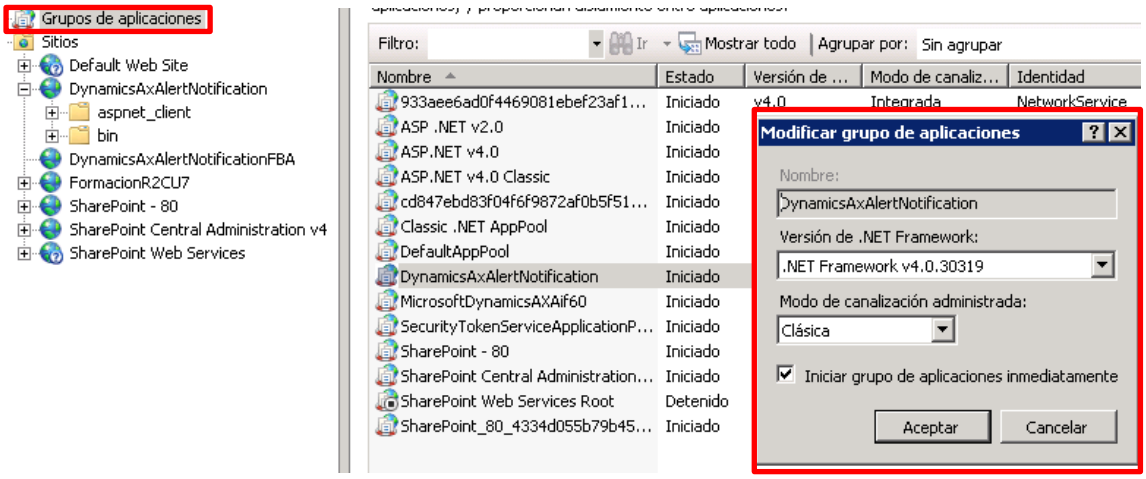



Ilustración 47. Grupo de aplicaciones

Se configurará de manera que use .Net framework 4.0 (versión necesaria para poder hospedar el servicio de WCF).

## 6.3. Seguridad, el usuario.

En grupo de aplicaciones se configurará un usuario para acceder al servicio web (Ilustración 48), cualquier usuario que se conecte al servicio web será impersonado por éste:

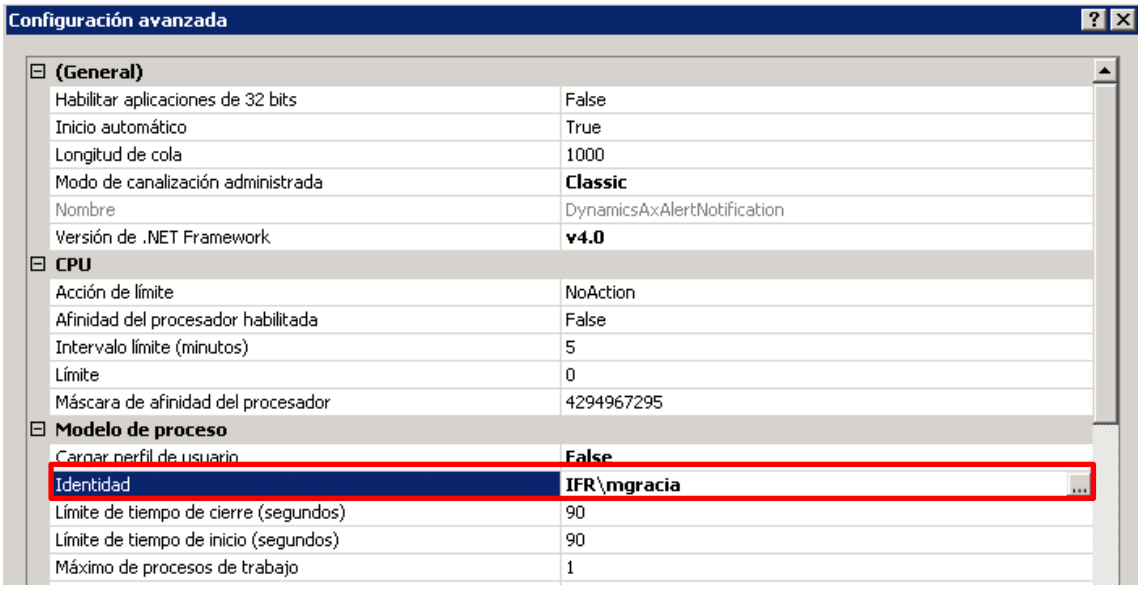



Ilustración 48. Usuario impersonación

Este usuario será el que use IIS para identificarse en el servicio web, por lo tanto, deberá ser un usuario lo más restringido en la medida de lo posible.

Básicamente el usuario sólo tendrá acceso a Dynamics AX, y dentro del ERP se restringirá sus permisos de manera que sólo pueda acceder a la consulta de alertas.

Para ello se creará en Dynamics AX un nuevo privilegio (Ilustración 49), que se llamará DynamicsAXAlert

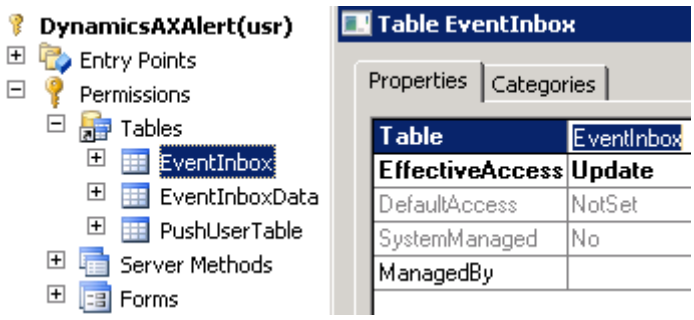



Ilustración 49. Privilegio de seguridad

Y se le vincularán las tablas EventInbox, EventInboxData y PushUserTable, de manera que sólo tendrá acceso a las tablas de alertas, el nivel de acceso para la primera es Update ya que se requiere poder actualizar el registro de alerta y marcarlo como leído. El acceso a la tabla PushUserTable es para poder autentificar el usuario.

Se creará luego el objeto rol DynamicsAXAlert (Ilustración 50) y se le vinculará el privilegio que se acaba de crear.


Ilustración 50. Rol de seguridad

Este rol se vinculará al usuario configurado en el application pool restringiendo así el acceso a datos dentro del propio Dynamics AX aumentando la seguridad.

### *6.4. Seguridad, certificado, cifrar las comunicaciones.*

Una vez configurado el application pool, dado que se desea que la comunicación entre la aplicación de Windows Phone y el servicio web sea cifrada, se realizará la comunicación por https mediante protocolo secure sockets layers (SSL).

Por lo tanto, antes de crear el sitio web se creará un certificado de seguridad desde el servidor de IIS (Ilustración 51):

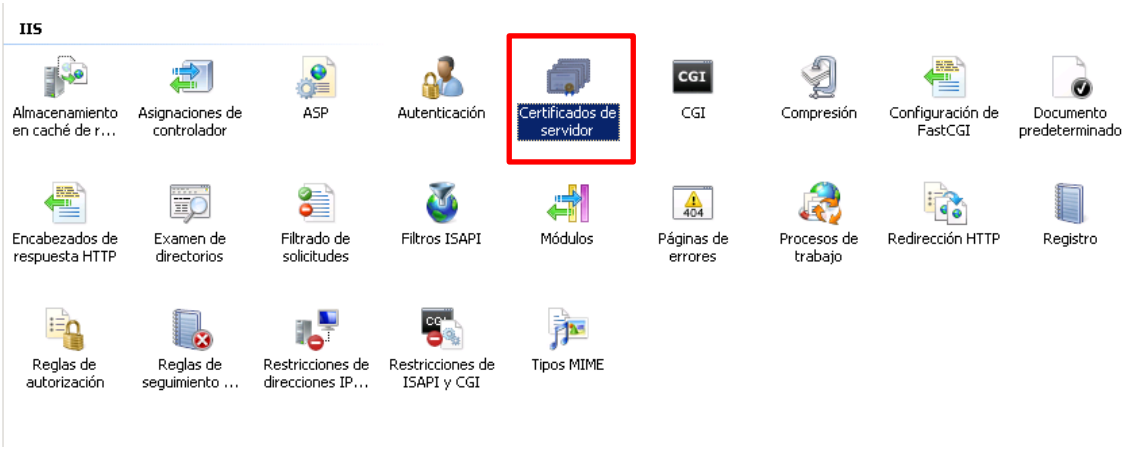



Ilustración 51. Certificado de servidor

La aplicación se está desarrollando para una aplicación móvil en Windows Phone, éste sistema operativo tiene una lista de autoridades de certificación (CA) y lleva pre-instalado sus certificados de raíz, esto implica que, si se solicita e instala un certificado emitido por (8)http://msdn.microsoft.com/en-us/library/windowsphone/develop/gg521150(v=vs.105).aspx):

- Definitions
- AOL (United States)

- Comodo (United States)
- DigiCert (United States)
- Entrust (Canada)
- GlobalSign (United Kingdom)
- GoDaddy (United States)
- Keynectis (France)
- QuoVadis (Bermuda)
- RSA Security (United States)
- SECOM Trust Systems Co. Ltd (Japan)
- Taiwan-CA Inc. (Taiwan)
- TrustCenter (Germany)
- Trustwave (United States)
- VeriSign (United States)
- VeriSign Business (United States)
- Related Topics

De hacerse así, la aplicación asumirá que se está comunicando realmente con quién cree hacerlo, en este caso el servicio web que le subministrará información sobre las alertas en Dynamics AX.

Dado que esta opción tiene un coste económico, se ha optado por crear un certificado auto-firmado. Según Microsoft se optará por esta opción (9)[http://technet.microsoft.com/es-es/library/cc753127(v=ws.10).aspx](http://technet.microsoft.com/es-es/library/cc753127(v=ws.10).aspx):

- Para solucionar los problemas relacionados con los certificados de otros fabricantes.
- Para administrar IIS de manera remota.
- Para crear un canal privado seguro entre su servidor y un grupo limitado de usuarios conocidos, como es el caso de un entorno de prueba de software.
- Para probar las características que dependen de la configuración de SSL.

Una vez escogida la opción de certificado auto-firmado, en el asistente, donde se pide el nombre del certificado (Ilustración 52) es **muy importante** usar el mismo nombre de dominio que tiene la máquina donde está el servidor IIS (10).

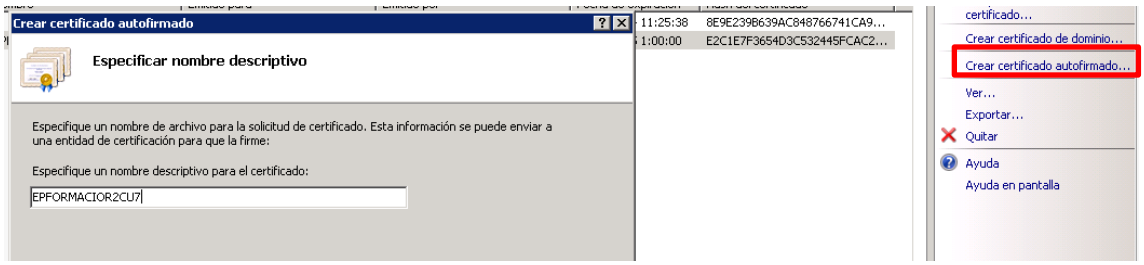



Ilustración 52. Configurar nombre de certificado

Esto es un requisito de seguridad Windows Phone para evitar problemas de suplantación de identidad (phishing), dado que el certificado se habrá de instalar en el móvil (el cómo se explicará más adelante). Cuando el terminal se conecte con el servicio web, éste verificará que

el nombre del certificado coincide con el nombre de la máquina del servicio web, en caso contrario no se podrá establecer la conexión.

Una vez creado el certificado se exportará la clave pública (Ilustración 53) para que se pueda instalar en el móvil:

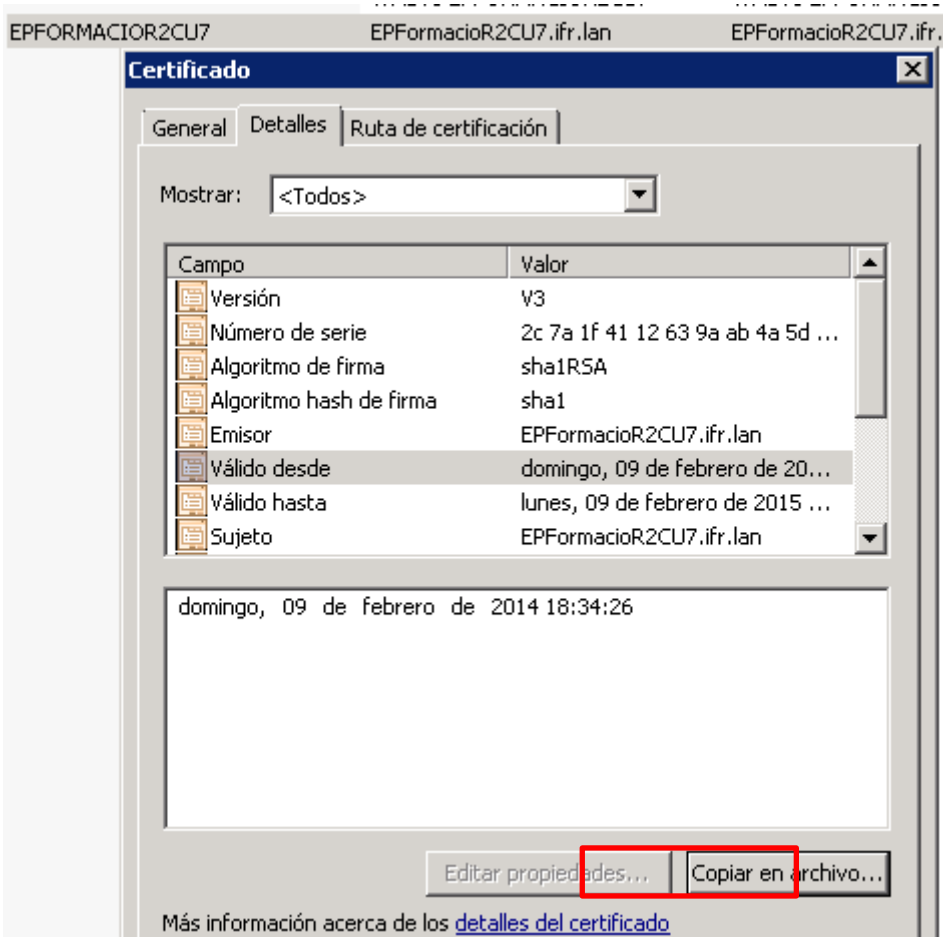



Ilustración 53. Exportación a fichero de certificado

Se escoge el formato (Ilustración 54)


Ilustración 54. Formato de exportación del certificado

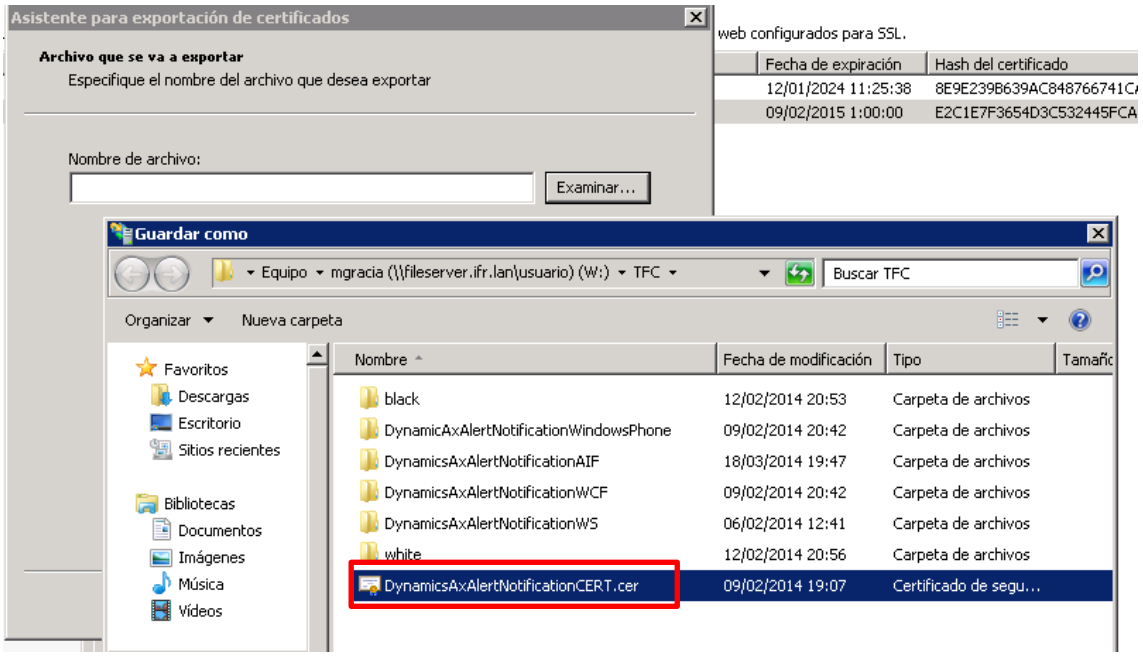



Ilustración 55. Nombre del archivo a exportar

El fichero generado (Ilustración 55) ha de comprimirse en formato zip para poderlo instalar más adelante en el móvil. Éste se dejará en la carpeta del sitio web (Ilustración 56):

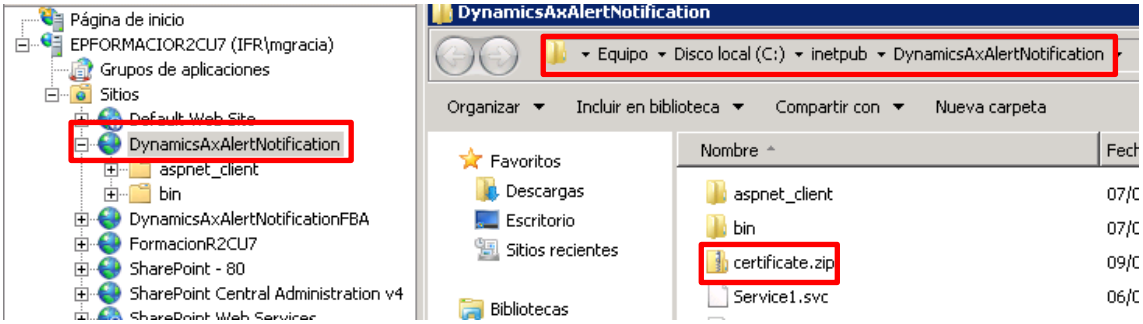



Ilustración 56. Guardar certificado en el sitio web

## 6.5. Configurar el sitio web.

Los servicios basados en WCF usan ISAPI, por defecto en un servidor de IIS está deshabilitado, para activarlo, en la configuración:

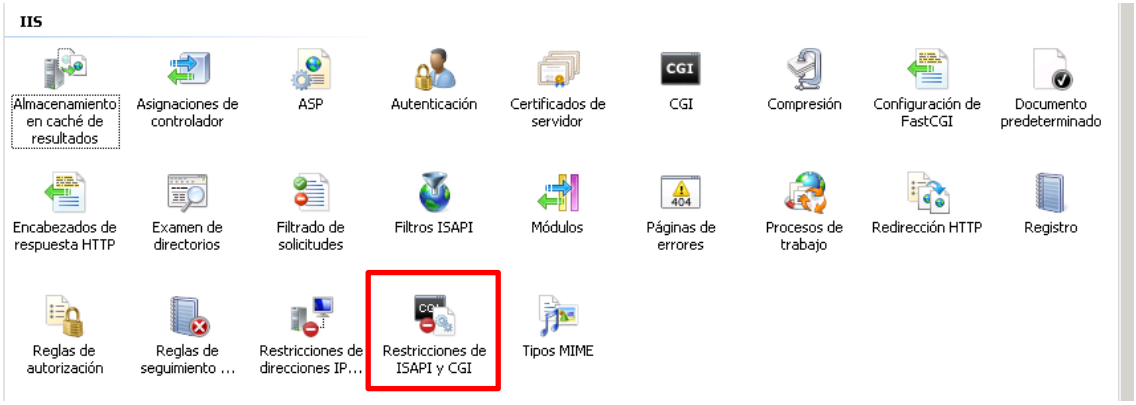



Ilustración 57. Restricciones ISAPI y CGI

Hay que seleccionar las restricciones de ISAPI y CGI (Ilustración 57)

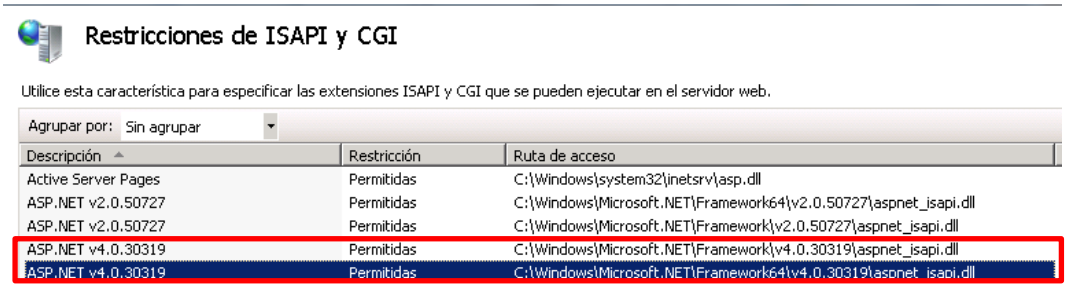



Ilustración 58. Configurar restricciones ISAPI y CGI

Y activarlas (Ilustración 58), en caso contrario al intentar acceder al servicio web, este responde con el error (Ilustración 59)


Ilustración 59. Error al no configurar correctamente ISAPI y CGI

Una vez realizadas las configuraciones pertinentes se puede proceder a crear el sitio web (Ilustración 60) que hospedará el servicio web.

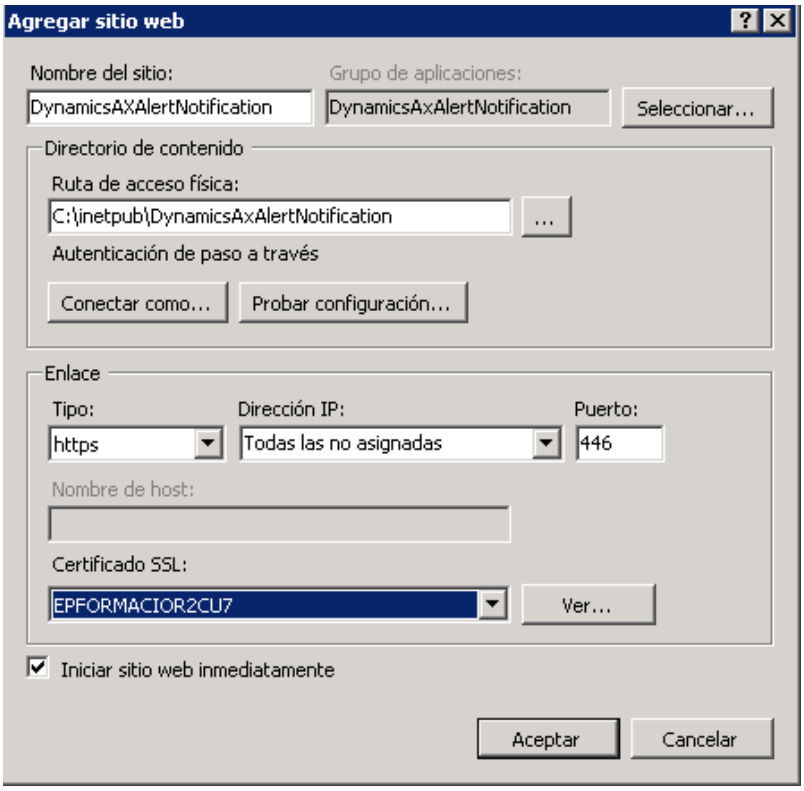



Ilustración 60. Creación del sitio web

Donde se seleccionarán el grupo de aplicaciones configurado anteriormente y el certificado SSL.

El tipo de enlace será https (se ha decidido cambiar el puerto 444 por 446, en este caso la decisión viene impuesta por el departamento técnico de IFR, dado que se ha usado su infraestructura para desarrollar el proyecto), no se configurará un segundo enlace o binding por http ya que la única manera en la que se desea acceder al servicio web es por https.

A partir de este momento "la aplicación de Windows Phone utiliza el certificado en conjunto con el servicio web para cifrar todas las comunicaciones, incluyendo el intercambio de las credenciales de autenticación" (11).

### *6.6. Crear el servicio de Windows Comunication Foundation.*

Una vez se ha creado el sitio web, se ha de crear un servicio web, en este caso desde Visual Studio 2010, creando un proyecto de WCF que se desplegará en el sitio web.

Este servicio web hará de puente entre Dynamics AX y la aplicación de Windows Phone, para ello se agregará en el WCF el servicio creado por AIF en Dynamics AX, de esta manera, se harán públicas las funciones para consultar las alertas.

Primero por tanto hay que crear el servicio de AIF para poderlo agregar en el servicio web. Al igual que en el punto 11.2 se creará el servicio con las mismas operaciones de servicio, a diferencia del anterior el protocolo a usar no será http si no netTcp (Ilustración 61), dado que

tanto el servidor de IIS y el AOS de Dynamics AX están dentro del mismo dominio y ambos funcionan bajo WCF.

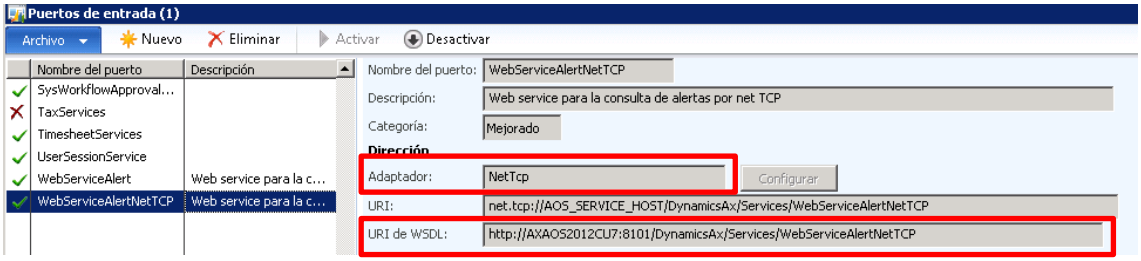



Ilustración 61. Configuración del puerto de entrada por NetTCP

Una vez creado el puerto de entrada, Dynamics AX devuelve la dirección (Ilustración 61) que se agregará en el proyecto de WCF.

En Visual Studio 2010 se crea un nuevo proyecto de WCF (Ilustración 62)

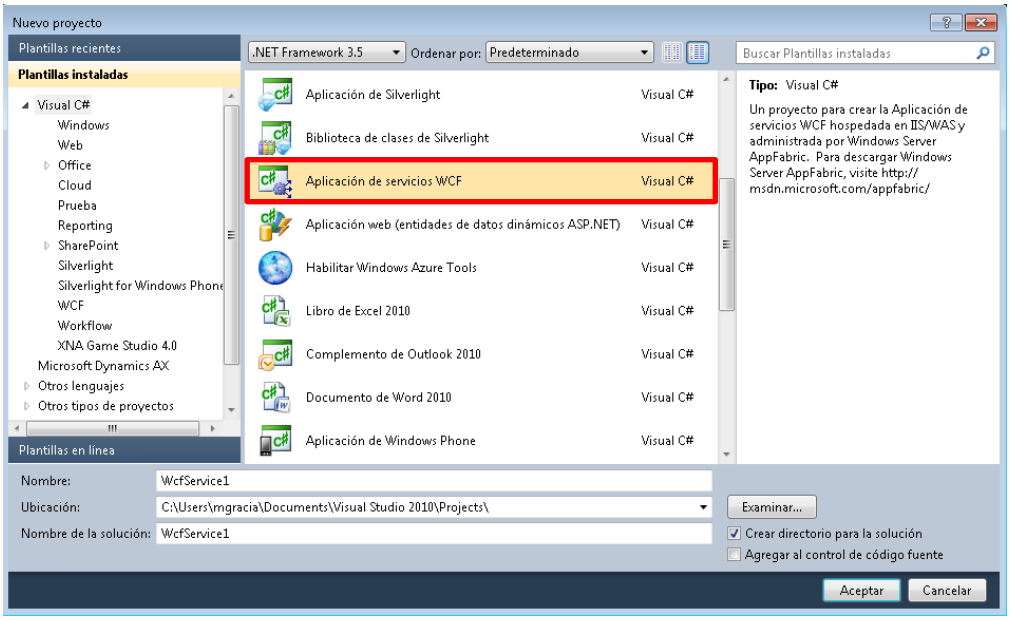



Ilustración 62. Creación de un proyecto de WCF en Visual Studio 2010

En el proyecto se agrega la referencia al servicio creado desde Dynamics AX usando la dirección proporcionada por el puerto de entrada

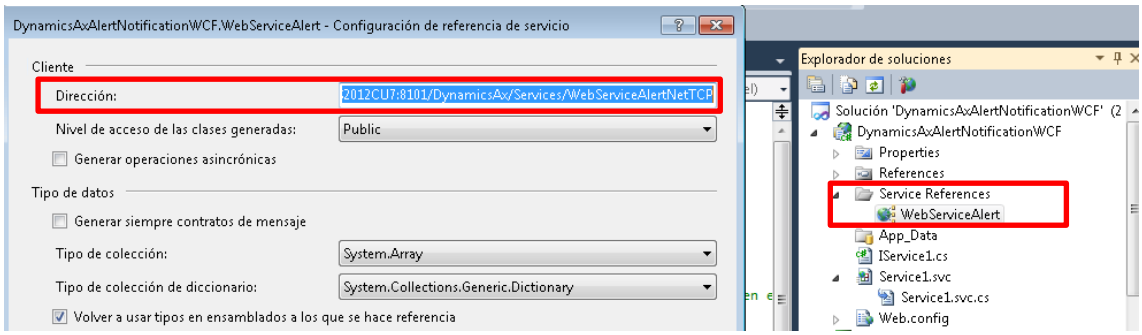



Ilustración 63. Agregar referencia al servicio web de Dynamics AX en el proyecto de WCF

Una vez el servicio es agregado ya se pueden acceder a las operaciones de servicio. Por lo tanto se trata de mapear estas operaciones de servicio, de manera que ahora sean accesibles mediante el servicio creado.

Se crean las mismas funciones con los mismos parámetros de entrada y salida en el proyecto de visual studio (Ilustración 64)

```
public class Service1 : IService1
{
    public WebServiceAlert.WebService_PushResult getAlertTypeList(string _pushUserId, string _pushUserPassword)...

    public WebServiceAlert.WebService_PushResult validateUserPassword(string _pushUserId, string _pushUserPassword)...

    public WebServiceAlert.WebService_PushResult updateUserPushURI(string _pushUserId, string _pushUserPassword, string _URIChannel)...
}
```

Ilustración 64. Funciones que publicará el servicio de WCF

Estas funciones son las que se llamarán desde la aplicación móvil y que a su vez llamarán a las funciones del servicio creado en Dynamics AX (Ilustración 65). Por lo tanto, en cada una de estas funciones, se hará una llamada al servicio creado con AIF .

```
public WebServiceAlert.WebService_PushResult getAlertTypeList(string _pushUserId, string _pushUserPassword)
{
    WebServiceAlert.WebService_PushResult ret = new WebServiceAlert.WebService_PushResult();

    try
    {
        WebServiceAlert.CallContext callContext = new WebServiceAlert.CallContext();
        WebServiceAlert.WebService_GetUserListAlertsClient client = new WebServiceAlert.WebService_GetUserListAlertsClient();

        ret = client.getUserAlert(callContext, _pushUserId, _pushUserPassword);
    }
    catch (Exception e)
    {
        ret.ProcessOk = false;
        ret.ProcessTxt = e.ToString();
    }

    return ret;
}
```

Ilustración 65. Ejemplo de llamada al servicio web publicado por Dynamics AX desde el servicio de WCF

En el ejemplo de la función getAlertTypeList:

1. Se inicializa el callContext, recordar que el servicio de IISse autentifica con el usuario configurado en el grupo de aplicaciones, que es un usuario de dominio dado de alta en Dynamics AX.
2. Se inicializa el cliente.

3. Se ejecuta la función getUserAlert pasándole el callcontext (que servirá para autentificarse contra el servicio de DynamicsAX) y un usuario y password para autentificar el usuario de Windows Phone.
4. Al igual que la operación de servicio de AIF, la función devuelve un objeto de tipo PushResult.

Para evitar cualquier tipo de problema se usa la gestión de errores, en caso de producirse uno, el mensaje del error se pasa al objeto PushResult, inicializando al variable ProcessOk a false y ProcessTxt con el mensaje del evento que generó la excepción.

Para el resto de funciones se habrá de realizar lo mismo de forma análoga. Todos los métodos devuelven un objeto de tipo WebService_PushResult para indicar el resultado de la operación.

### *6.7. Configurar el web.config.*

Finalmente habrá que configurar el web.config que es el fichero de configuración para el servicio web. Las partes a destacar son:

1. Binding (Ilustración 66), el enlace, define la manera en que el cliente se comunicará con el servicio web.

```
<bindings>
  <basicHttpBinding>
    <binding name="secureHttpBinding">
      <security mode="Transport">
        <transport clientCredentialType="None" />
      </security>
    </binding>
  </basicHttpBinding>
```

Ilustración 66. Binding

   Se usará la clase basicHttpBinding, por defecto no proporciona seguridad, para ello hay que definir el security mode a Transport, esto asegura el uso del protocolo SSL y del certificado para la encriptación de datos.
   No se usa TransportWithMessageCredential dado que no se pasan credenciales al conectarse al servicio web.
   La propiedad clientCredentialType está definida en “None” (autentificación anónima), tal y como se comentó en el punto 12.3 la autentificación se hará en el propio Dynamics AX.

2. Service (Ilustración 67), define el servicio programado y la configuración

```
<services>
  <service name="DynamicsAxAlertNotificationWCF.Service1" behaviorConfiguration="DynamicsAxAlertNotificationWCF.Service1Behavior">
    <!-- Service Endpoints -->
    <endpoint address="" binding="basicHttpBinding" bindingConfiguration="secureHttpBinding" contract="DynamicsAxAlertNotificationWCF.IService1">
    </endpoint>
    <endpoint address="mex" binding="mexHttpsBinding" contract="IMetadataExchange"/>
  </service>
</services>
```

Ilustración 67. Service

En este caso el servicio se llama Service1, y para el uso de un servicio con SSL hay que especificar la configuración como secureHttpBinding

### 6.8. Publicar y verificar el servicio web.

Una vez configurado y programado el servicio web tan sólo hay que hospedarlo en el sitio web creado (Ilustración 68), para ello, desde el propio Visual Studio, en el menú generar\publicar permite subir el proyecto al sitio creado anteriormente

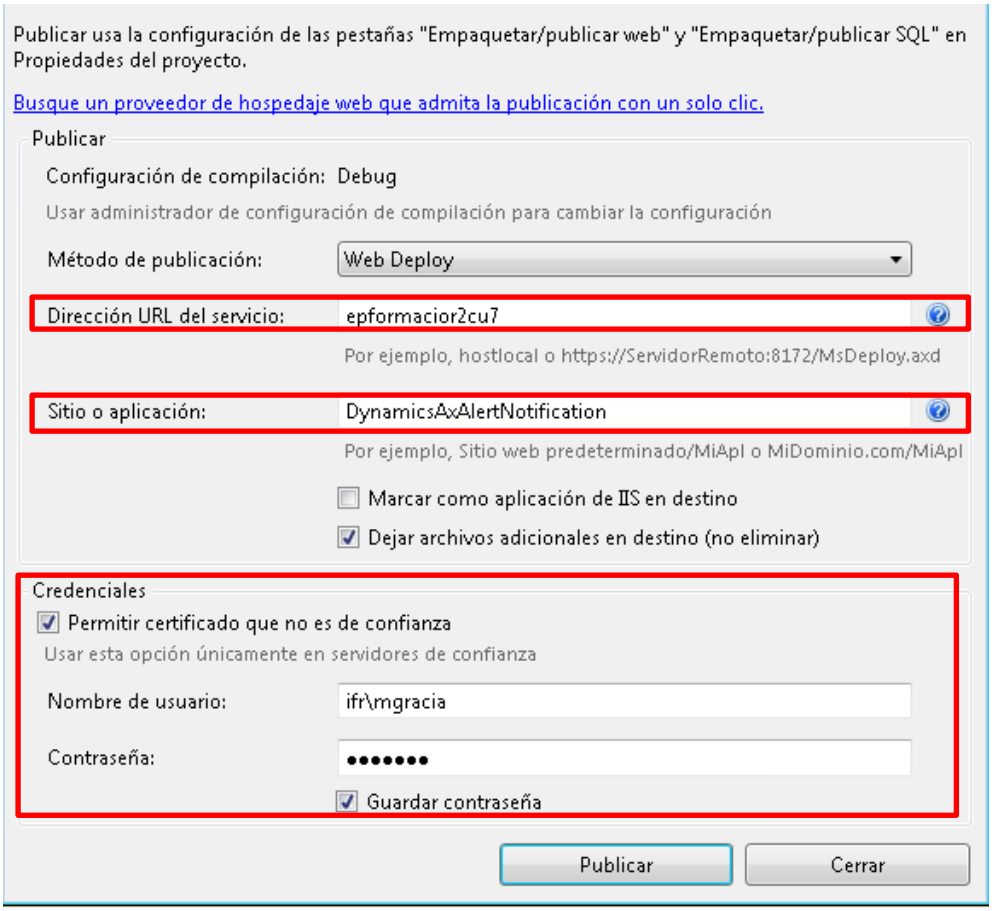



Ilustración 68. Publicar sitio web

Indicando el nombre del servidor, el nombre del sitio web y un usuario para autentificarse contra la máquina con suficientes derechos para realizar dicha acción.

A partir de este momento se puede desarrollar una aplicación para consultar alertas generadas en Dynamics AX desde otro software o aplicación consumiendo el servicio web.

Se puede verificar el acceso al servicio accediendo desde un navegador web, en este caso el servicio se llama Service1.svc

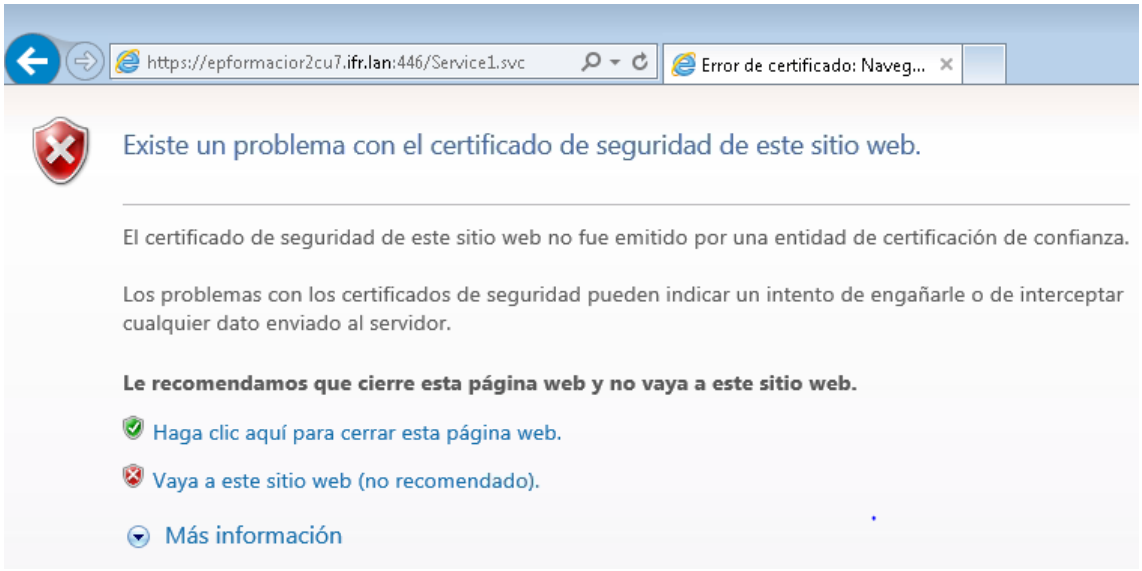



Ilustración 69. Aviso de certificado

Al acceder si aparece un aviso de certificado de seguridad (Ilustración 69), es debido a que no se ha instalado el certificado creado para el sitio web en la máquina donde se está ejecutando el navegador, igualmente se puede acceder, con lo que aparece el acceso al servicio de WCF (Ilustración 70)

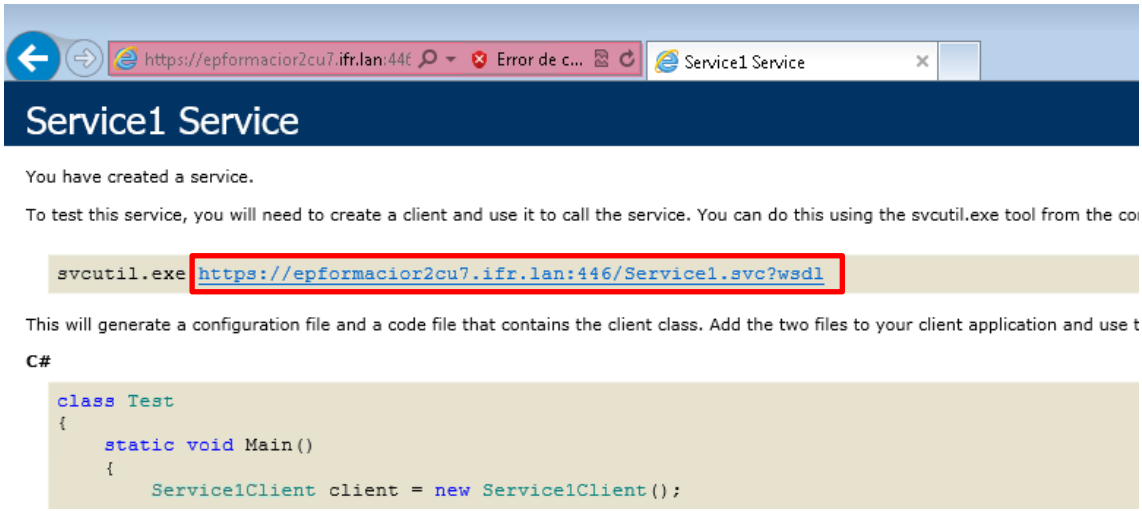



Ilustración 70. Acceso al servicio web desde el navegador

# 7. Protocolo push.

Hasta ahora se ha mostrado como crear un servicio web para que, desde una aplicación externa, se consulten alertas generadas en Dynamics AX, en este caso, una aplicación móvil.

Este sistema es activo, es decir, el usuario ha de conectarse al servicio para poder consultar si se han generado nuevas alertas. El objetivo inicial es que el sistema sea pasivo, es decir, ha de ser la aplicación móvil la que avise al usuario para indicarle que hay alertas nuevas sin necesidad que el usuario ejecute la aplicación para verificar si hay datos nuevos.

Se podría pensar en hacer que la aplicación móvil se conectara de forma automática cada x minutos para consultar si hay alertas nuevas, esto tiene varios inconvenientes:

1. El sistema no es pasivo, si no activo.
2. Se consume tráfico de internet de forma innecesaria si no hay nuevas alertas.
3. La aplicación debe estar en ejecución, con lo que se consumen recursos del móvil.

La alternativa a este sistema es el protocolo push, éste se basa en un servicio (A) en el cual se suscribe un cliente (B), cuando una tercera parte (C) desea hacer llegar un mensaje a B lo hace mediante A. En este caso, las partes son:

- A, el servicio de Microsoft Push Notification Service (MPNS).
- B, aplicación móvil de Windows Phone.
- C, Microsoft Dynamics AX.

En el protocolo push, tan pronto el mensaje es enviado, este se remite al cliente, un ejemplo son las aplicaciones de mensajería instantánea. Por lo tanto:

1. El sistema es pasivo y no activo.
2. No se consume tráfico de internet de forma innecesaria, dado que cuando se recibe la alerta se sabe seguro que hay nuevos datos que consultar.
3. La aplicación no debe estar necesariamente en ejecución para recibir la alerta.

Con este sistema el usuario no debe preocuparse de consultar las alertas ya que el móvil le avisará cada vez que se genere una alerta nueva.

La razón del uso de MPNS es que este servicio está especialmente preparado para trabajar con Windows Phone, en el caso de Apple para el sistema operativo de IOS se usa el Apple Push Notification Service, y en el caso de Android el servicio lo proporciona Google Cloud Messaging.

## *7.1. Tipos de notificaciones.*

MPNS permite enviar tres tipos de notificaciones distintas, cada una de ellas orientada a una necesidad distinta:

1. Tile, modifica el aspecto del tile (icono) de la aplicación, pudiendo indicar dos imágenes para el icono que irán cambiando automáticamente y un número, que

aparecerá en la parte superior derecha del tile. Esta notificación sólo se puede realizar si el icono está anclado en el escritorio principal del teléfono.
2. Toast, hace aparecer en el móvil un banner en la parte superior donde se pueden hacer llegar dos cadenas de texto cortas. Si se clica el banner la aplicación se ejecuta, en caso contrario, el banner desaparece en cinco segundos.
3. Raw, a diferencia de los anteriores no modifica el aspecto gráfico del móvil, el mensaje puede ser interpretado por la aplicación para llevar a cabo una acción en particular.

En el caso del proyecto se presentarán las notificaciones de la siguiente manera:

1. Tile, se cambiará el aspecto del icono haciendo aparecer un icono en forma de alerta con un número que indicará el número de alertas pendientes de leer.
2. Toast, si no se ha podido enviar un tile, (porque el programa no está anclado en el escritorio principal), se enviará un toast donde el banner notificará al usuario que se han generado nuevas alertas y el número pendiente de leer.
3. Raw, se podría enviar una notificación para obligar a la aplicación a abrir la consulta de la alerta que se ha generado, se ha descartado esta opción porque podría llegar a ser molesto para el usuario. El objetivo final es avisar al usuario y esto se consigue con los dos primeros tipos de notificación.

### *7.2. Estados de la respuesta de la notificación*

Es importante entender correctamente los estados de respuesta del push, ya que con ellos se puede llevar a cabo una gestión de errores que ayude a entender qué problema ha podido ocurrir, dado que aunque el servicio MPNS responda, puede que el mensaje no haya sido enviado (12)

El servicio responde indicando los estados de la propia notificación, estado del terminal móvil y el estado de la subscripción.

En el caso de la notificación los estados son:

1. Received, el mensaje fue recibido.
2. QueueFull, se pueden enviar un máximo de 30 mensajes sin que estos hayan sido recibidos, una vez llegado al tope se descartan las nuevas notificaciones hasta que las anteriores no hayan sido notificadas como recibidas.
3. Suppresed, se recibe si el cliente no espera la notificación para tratarla como tile o toast.
4. Dropped, se recibe si el URI del cliente ha expirado, es inválido o se ha alcanzado el límite de mensajes diarios (500).

En el caso del estado del terminal móvil, los estados son:

1. Connected, el móvil está conectado y accesible.
2. TempDisconnected, el móvil puede estar desconectado por múltiples razones, como por ejemplo que no tenga cobertura para datos, que esté apagado, que haya cerrado

las conexiones a datos, que el plan contratado con la compañía de teléfono haya llegado al límite de tarifa de datos contratados, etc…
3. Disconnected, si la subscripción al servicio ha caducado, o el URI proporcionado es inválido, el estado aparece como desconectado (no tiene nada que ver con el punto 2).

En el caso del estado de la subscripción, los estados son:

1. Active, está activa y sin problemas.
2. Expired, ha caducado o el URI proporcionado es incorrecto.
3. N/A, no disponible, puede ocurrir porque se haya enviado una notificación en formato XML incorrecto o que el servicio esté temporalmente no disponible.

### *7.2. Como enviar notificaciones des de Dynamics AX.*

Una vez se ha decidido usar el protocolo push para enviar notificaciones a Dynamics AX, la pregunta es, ¿cómo se realiza este proceso?, de la siguiente manera:

1. La aplicación móvil solicita registrarse en el servicio de MPNS, éste le devuelve un URI (en forma de url), el cual identifica de forma única el dispositivo móvil.
2. La aplicación móvil ha de enviar el URI a Dynamics AX para poderla vincular con un usuario.
3. Una vez el usuario y el móvil están vinculados, se pueden enviar las notificaciones mediante el protocolo push.

A continuación se detallan los puntos anteriores.

### *7.3. Registrarse en el servicio MPNS.*

Esta parte se detallará en el apartado de la memoria donde se habla de la aplicación móvil.

### *7.4. Vincular el URI con un usuario de Dynamics AX.*

El usuario de la aplicación móvil se autentifica contra Dynamics AX con una nueva estructura de datos comentada en el punto Seguridad en Dynamics AX, de esta manera, en la tabla PushUserTable tenemos vinculado un usuario de Dynamics AX con un usuario de la aplicación móvil.

En esta tabla además se ha creado el campo PushURI, se trata por tanto, de cuando la aplicación móvil solicite el URI al MPNS, actualizar el campo de la tabla. De esta manera se consigue tener vinculado el dispositivo móvil físico a un usuario de Dynamics AX, ¿Cómo?, nuevamente mediante servicio web creado usando el AIF.

```
1   [
2       SysEntryPointAttribute(true),
3       AifCollectionTypeAttribute('return', Types::Class, classStr(WebService_PushResult))
4   ]
5   public WebService_PushResult updatePushUserURI(PushUserId   _pushUserId,
6       PushPassword _pushPassword,
7       PushURI _pushURI)
8   {
9       WebService_PushResult   ret = new WebService_PushResult();
10      PushUserTable           pushUserTable;
11      container               validateUser = PushUserTable::validatePushUser(_pushUserId, _pushPassword);
12      boolean                 validate = conPeek(validateUser, 1);
13      str                     validateStr = conPeek(validateUser, 2);
14      UserId                  userId = conPeek(validateUser, 3);
15
16      ret.parmProcessOk(validate);
17      ret.parmProcessTxt(validateStr);
18
19      if (validate)
20      {
21          ttsBegin;
22
23          pushUserTable = PushUserTable::find(_pushUserId, _pushPassword, true);
24
25          if (pushUserTable && pushUserTable.PushURI != _pushURI)
26          {
27              pushUserTable.PushURI = _pushURI;
28              pushUserTable.update();
29          }
30
31          ttsCommit;
32      }
33
34      return ret;
35  }
```

Ilustración 71. Vinculación del URI del MPNS al usuario push

Se crea la clase WebService_Push y se le añade el método público updatePushUserURI, la función tiene especificada el atributo SysEntryPointAttribute dado que será una operación de servicio en el puerto de entrada creado mediante AIF para la notificación de alertas.

El objeto a devolver es de tipo WebService_PushResult, dado que con este objeto se puede notificar si la operación ha ido correctamente o no y el porqué.

Como parámetros de entrada, el usuario y password de la aplicación móvil y el URI. De esta manera la función valida si el usuario es correcto (línea 11 del código), y en caso afirmativo, se busca el registro en la base de datos que contiene el usuario (línea 23 del código) y se actualiza el campo URI en caso que este haya variado (línea 27 del código).

De esta manera un usuario de la aplicación móvil, y por tanto, un usuario de Dynamics AX, queda vinculado con el móvil, a partir de este momento puede recibir notificaciones.

Para poder vincular esta función al servicio web, hay que crear en Dynamics AX un servicio (Ilustración 72) con la clase que contiene el método y especificarlo para que se pueda vincular al puerto de entrada.

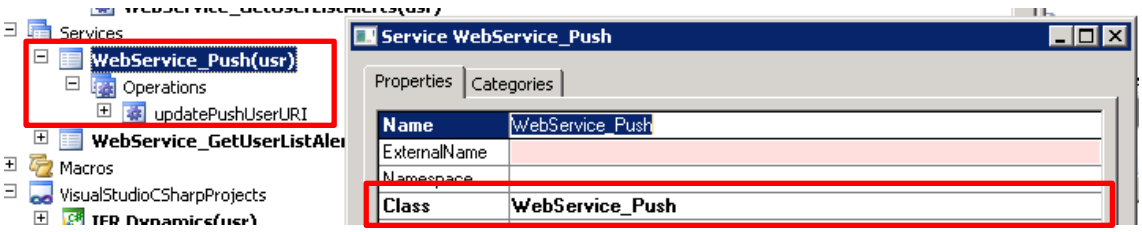



Ilustración 72. Servicio Push en Dynamics AX

Una vez creado el servicio, se puede vincular la operación (Ilustración 73) al puerto de entrada

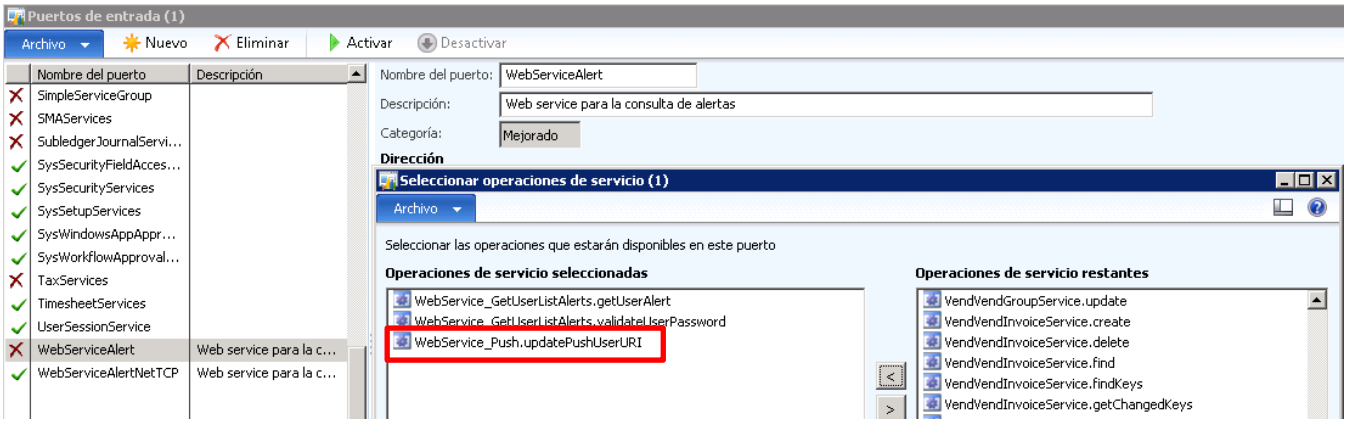



Ilustración 73. Vincular operación para actualizar URI en el puerto de entrada

Ahora ya se puede usar el webservice para actualizar el URI (proporcionado por el MPNS) del usuario cuando sea necesario.

### *7.5. Enviar la notificacion push a la APP.*

Una vez el usuario de Dynamics AX y el terminal móvil quedan vinculados se pueden enviar notificaciones cada vez que se genere una alerta.

Para poder enviar la notificación se necesita usar objetos del framework de .NET, Dynamics AX tiene su propio lenguaje, X++, que tiene acceso limitado al framework, por lo que existen dos opciones:

1. Escribir una dll en c#, y luego incluirla como referencia en Dynamics AX (Ilustración 74).

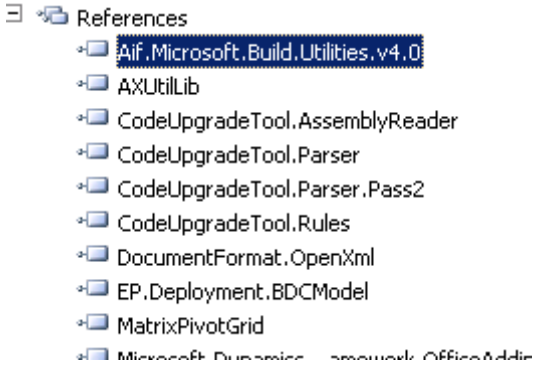



Ilustración 74. Referencia de una dll en Dynamics AX

2. Aprovechar la nueva funcionalidad de Dynamics AX 2012 para crear un proyecto de Visual Studio 2010 e incluirlo en el árbol de objetos de la aplicación (Ilustración 75).

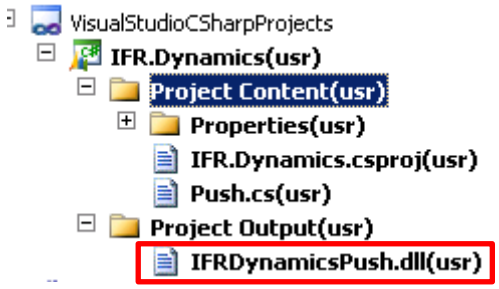



Ilustración 75. Proyecto de Visual Studio en Dynamics AX

La primera opción es totalmente funcional, una vez agregada la referencia a la dll, se pueden llamar a las clases y funciones definidas que deberían servir para enviar las notificaciones push. Existen sin embargo varios inconvenientes:

1. El código está fuera de Dynamics AX, si alguna vez éste se hubiera de modificar, el código fuente ha de estar localizable y accesible para poderlo modificar, volver a compilar y agregarlo nuevamente como referencia en Dynamics AX.
2. Dado que el código que envía las alertas se ejecuta en el servidor, hay que asegurarse que la dll esté registrada en cada uno de los servidores (máquinas físicas o virtuales) de Dynamics AX, ya que esta aplicación admite múltiple servidores para mejoras de rendimiento y escalabilidad.
3. Si se añaden nuevos servidores o se cambian físicamente de máquina, hay que recordar registrar nuevamente la dll.

La segunda opción ofrece claras mejoras respecto a la anterior:

1. El código está dentro de Dynamics AX, si alguna vez se hubiera de modificar, el código fuente está accesible y localizable, se puede abrir el proyecto nuevamente con Visual Studio para hacer las modificaciones y estas se actualizan automáticamente en Dynamics AX.
2. Una vez se compila el proyecto se genera la dll y es el propio Dynamics AX el encargado de propagarlo entre las distintas instancias de los servidores, asegurándose que todos tengan la misma versión.
3. Si se añaden nuevos servidores, estos verifican si hay dll's que descargarse y se actualizan automáticamente.

Por lo tanto, para el envío de notificaciones mediante protocolo push se realizará creando un proyecto de Visual Studio 2010 y vinculándolo a Dynamics AX.

### *7.5.1. Escribir la dll en C# y vincularlo al AOT de Dynamics AX*

Para poderlo realizar primero se debe asegurar que la configuración del cliente de Dynamics AX (Ilustración 76) esté apuntando a la aplicación (Ilustración 77) en la cual se desea crear el proyecto de Visual Studio, desde Windows, Inicio:

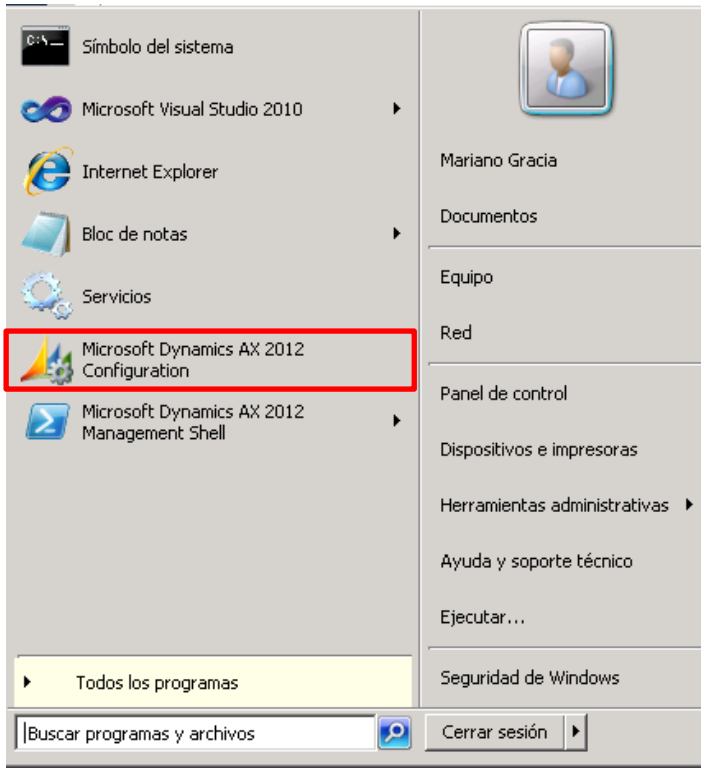



Ilustración 76. Acceso a la configuración del cliente de Dynamics AX

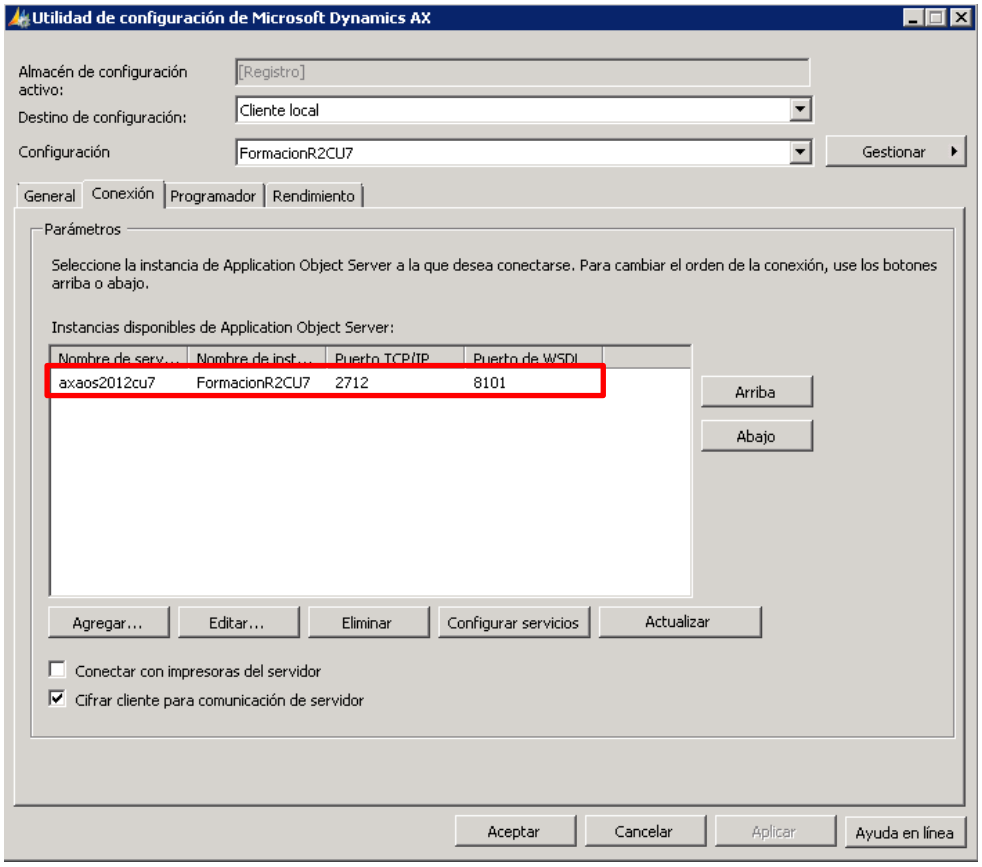



Ilustración 77. Configuración del cliente de Dynamics AX

Se debe verificar que la configuración apunta contra el servidor “axaos2012cu7” y la aplicación de Dynamics AX deseada, en este caso “FormacionR2CU7”

Una vez verificada la configuración, al iniciar Visual Studio 2010 se puede ver como en la ventana de “Application Explorer” aparece el árbol de objetos de la aplicación de Dynamics AX (Ilustración 78)

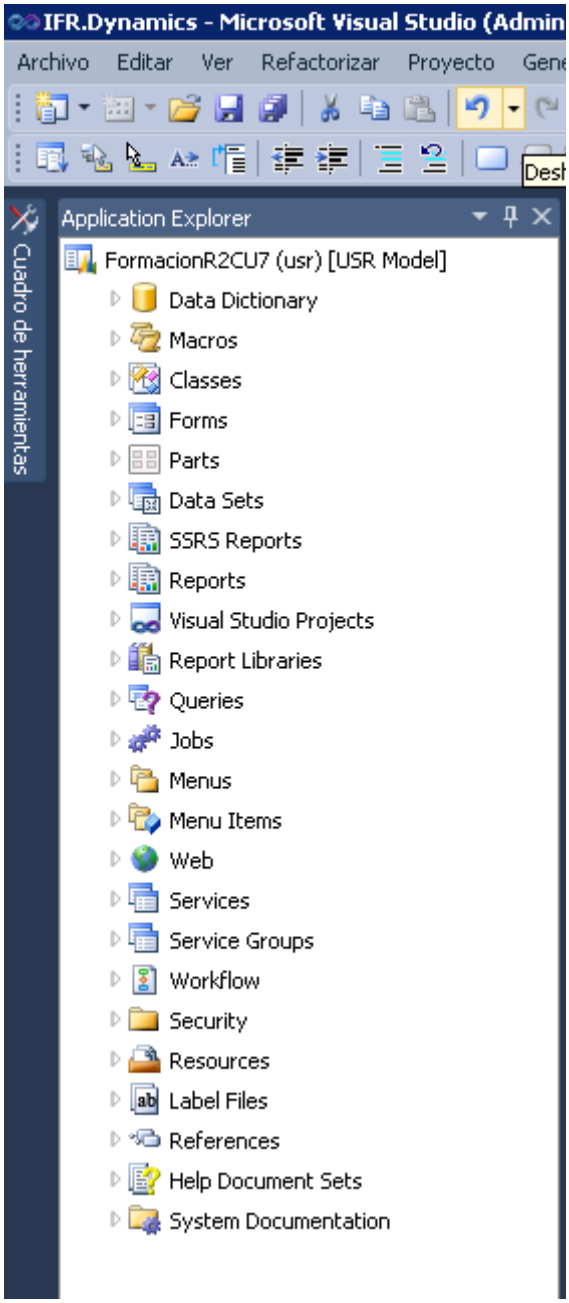



Ilustración 78. Árbol de objetos de Dynamics AX desde Visual Studio

Por lo tanto ya se puede crear un proyecto nuevo, en este caso una biblioteca de clases en c# (Ilustración 79)

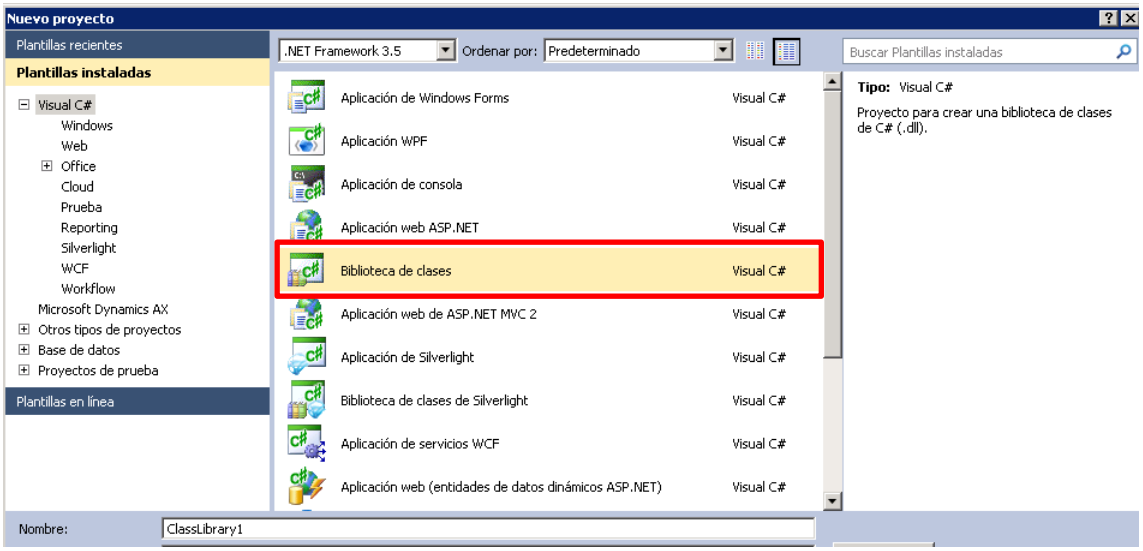



Ilustración 79. Creación de un proyecto de biblioteca de clases en Visual Studio

En el proyecto, en este caso IFR.Dynamics, se añade la clase Push, y se especifica a nivel de proyecto que la dll se desplegará tanto en el cliente como en el servidor (Ilustración 80).

El código que generan las alertas se ejecuta en el servidor, por lo tanto, la opción de desplegarlo en el cliente no es estrictamente necesaria. La opción Deploy to EP es para Enterprise Portal, la web de Dynamics AX, en este caso el código tampoco se va a ejecutar nunca en el portal.

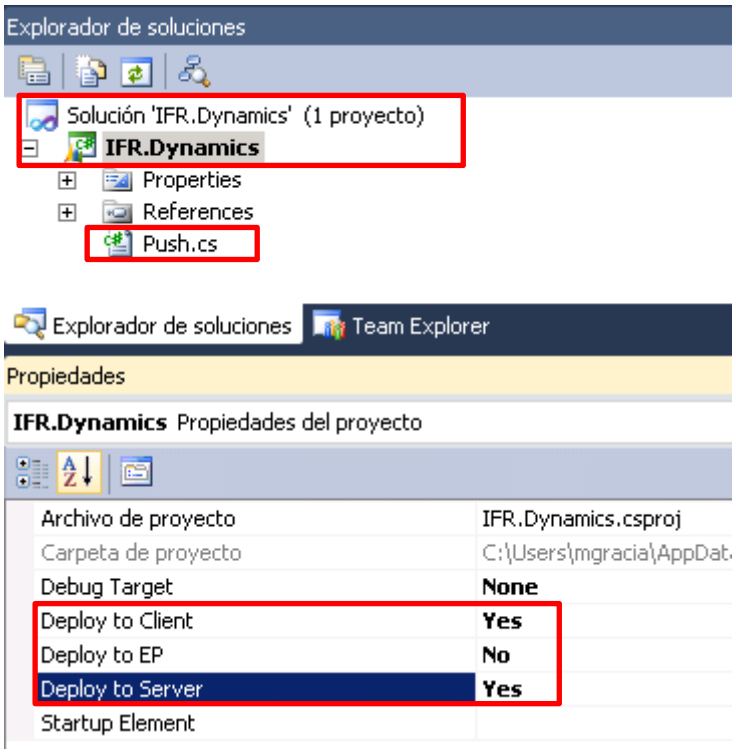



Ilustración 80. Implementación del proyecto de Visual Studio en Dynamics AX

La clase Push contiene dos métodos públicos (Ilustración 81):

```
public class Push
{
    public List<string> sendToast(string _subscriptionUri, string _toastXMLMessage)...

    public List<string> sendTile(string _subscriptionUri, string _tileXMLMessage)...

}
```

Ilustración 81. Métodos de la clase Push

uno para cada tipo de notificación, tile y toast (Ilustración 82). Como parámetros de entrada tienen el URI, que identifica a quién se envía la notificación, y un XML con mensaje que desea enviar.

Como parámetro de devolución una lista donde el primer elemento es “true” o “false” para indicar si el mensaje ha podido ser enviado, y los otros tres elementos contienen datos del estado de la notificación, la subscripción al servicio y el estado del terminal móvil.

```
public List<string> sendToast(string _subscriptionUri, string _toastXMLMessage)
{
    List<string> ret = new List<string>();

    try
    {
        HttpWebRequest sendNotificationRequest = (HttpWebRequest)WebRequest.Create(_subscriptionUri);
        sendNotificationRequest.Method = "POST";

        // Set the notification payload to send.
        byte[] notificationMessage = Encoding.Default.GetBytes(_toastXMLMessage);

        // Set the web request content length.
        sendNotificationRequest.ContentLength = notificationMessage.Length;
        sendNotificationRequest.ContentType = "text/xml";
        sendNotificationRequest.Headers.Add("X-WindowsPhone-Target", "toast");
        sendNotificationRequest.Headers.Add("X-NotificationClass", "2");


        using (Stream requestStream = sendNotificationRequest.GetRequestStream())
        {
            requestStream.Write(notificationMessage, 0, notificationMessage.Length);
        }

        // Send the notification and get the response.
        HttpWebResponse response = (HttpWebResponse)sendNotificationRequest.GetResponse();
        string notificationStatus = response.Headers["X-NotificationStatus"];
        string notificationChannelStatus = response.Headers["X-SubscriptionStatus"];
        string deviceConnectionStatus = response.Headers["X-DeviceConnectionStatus"];

        ret.Add("true");
        ret.Add(notificationStatus);
        ret.Add(notificationChannelStatus);
        ret.Add(deviceConnectionStatus);

        return ret;
    }
    catch (Exception ex)
    {
        ret.Add("false");
        ret.Add(ex.ToString());

        return ret;
    }
}
```

Ilustración 82. Método para enviar un toast

Con el URI se solicita mediante el objeto HttpWebRequest la petición al servicio MPNS, se le pasa como parámetro el XML que contiene una estructura en específico, en este caso para un mensaje de tipo toast, en el encabezado (objeto header del HttpWebRequest) del mensaje se le indica que el mensaje es un toast.

Una vez el servicio responde se pueden leer los tres estados (del encabezado de la solicitud) de los que se habla en el punto 4 y se construye el objeto a devolver.

Como precaución, si se produce una excepción, igualmente se construye el objeto a devolver, la diferencia, el primer parámetro contiene un “false” y el segundo el mensaje que produjo el error.

En el caso de la función sendTile el código es muy similar, únicamente varia el encabezado de la solicitud al servicio.

Una vez compilado el proyecto, la clase Push ya es accesible desde Dynamics AX.

### *7.5.2. Modificar Dynamics AX para usar la dll.*

Una vez se dispone de la clase Push para enviar notificaciones, hay que modificar el código estándar de Dynamics AX para que, cuando se genere una alerta, se envíe una notificación. El objetivo de esta notificación es avisar al usuario que se ha generado una nueva alerta y cuantas tiene pendientes de leer. Para ello se aprovechará la notificación tile que muestra en el icono el número de alertas pendientes de leer (Ilustración 83):

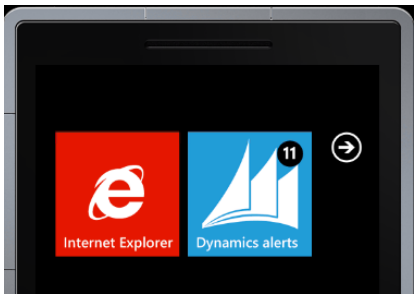



Ilustración 83. Notificación Tile mostrando el número de alertas pendinentes de lectura

Si el Tile de la aplicación no está anclado a la pantalla inicial se enviará una notificación Toast, la cual hace aparecer un aviso en la parte superior de la aplicación enviando un mensaje indicando el número de alertas sin leer (Ilustración 84):

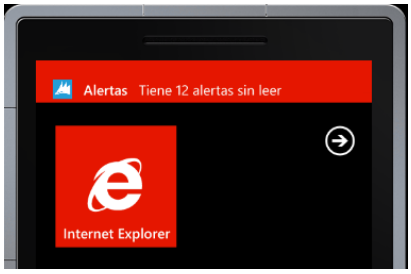



Ilustración 84. Notificación Toast mostrando el número de alertas pendinentes de lectura

El algoritmo es el siguiente (Ilustración 85):

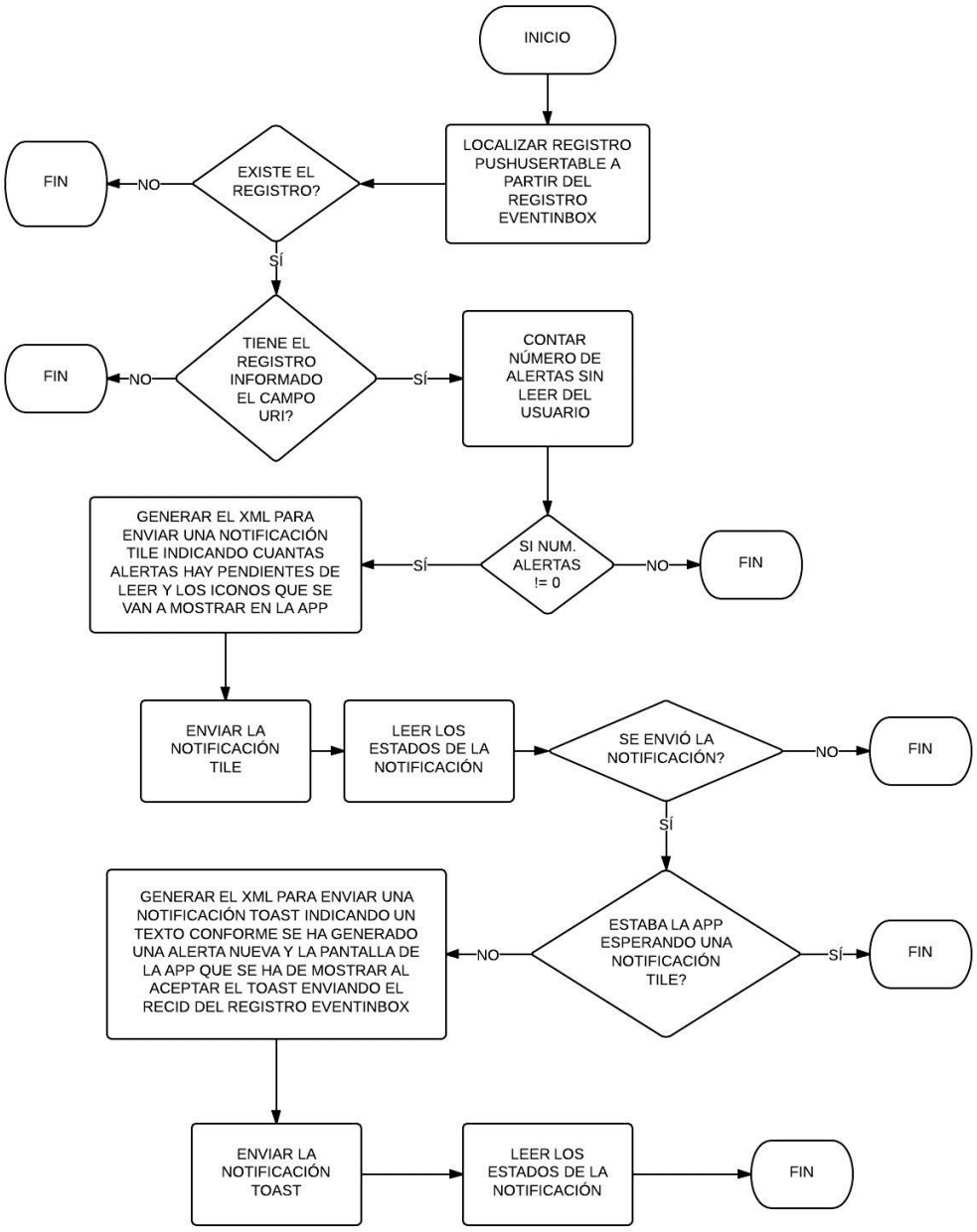



**Ilustración 85. Algoritmo para enviar notificaciones push**

Para ello se creará una clase nueva en Dynamics AX, SendPushNotification, que enviará las notificaciones cada vez que se genere una alerta nueva, esta clase debe seguir el algoritmo descrito (Ilustración 85):

1. El método constructor tiene como parámetro de entrada un registro EventInbox.
2. Se llamará a la clase ejecutando el método sendTile:

```
1   public void sendTile()
2   {
3       IFR.Dynamics.Push       push = new IFR.Dynamics.Push();
4       CLRObject               enumerator;
5       InteropPermission       permission;
6       str                     uRI;
7       str                     tileMessage;
8       boolean                 mPNSActive;
9       str                     notificationStatus;
10      str                     notificationChannelStatus;
11      str                     notificationDeviceConnectionStatus;
12
13      uRI = PushUserTable::findByUserId(inbox.UserId).PushURI;
14
15      if (uRI)
16      {
17          tileMessage = this.buildTileXMLMessage();
18
19          if (tileMessage)
20          {
21              permission = new InteropPermission(InteropKind::CLRInterop);
22              if (permission == null)
23              {
24                      return;
25              }
26              permission.assert();
27
28              enumerator = push.sendTile(uRI, tileMessage);
29
30              CodeAccessPermission::revertAssert();
31
32              [mPNSActive, notificationStatus, notificationChannelStatus, notificationDeviceConnectionStatus] = this.mPNSResponse(enumerator);
33
34              this.processMPSNRespone(uRI, mPNSActive, notificationStatus, notificationChannelStatus, notificationDeviceConnectionStatus);
35          }
36      }
37  }
```

Se localiza el URI vinculado al usuario (línea 13), se construye el XML para el mensaje tile que se enviará al uri (línea 17) y se usa la clase push descrita en el punto 7.5.1 para enviarlo al servicio MPNS. En la línea 32 se recoge la respuesta del servicio, estados del servicio, notificación, servicio y dispositivo descritos en el punto 7.2, con la respuesta se ejecuta el método processMPSNResponse:

```
1   protected void processMPSNRespone(PushURI _uRI,
2       boolean _mPNSActive,
3       str _notificationStatus,
4       str _notificationChannelStatus,
5       str _notificationDeviceConnectionStatus,
6       boolean _isTileMessage = true)
7   {
8       RecId   pushUserTableRecId = PushUserTable::findByUserId(inbox.UserId).RecId;
9       PushUserTableLog    pushUserTableLog;
10
11      //LA APP NO ESTABA ESPERANDO UN TILE, SE LE ENVIARÁ UN TOAST
12      if (_isTileMessage && _mPNSActive && _notificationStatus == "Suppressed" && _notificationChannelStatus == "Active" && _notificationDeviceConnectionStatus == "Connected")
13      {
14          this.sendToast(_uRI);
15      }
16      else if (!_isTileMessage && _mPNSActive && _notificationStatus == "Suppressed" && _notificationChannelStatus == "Active" && _notificationDeviceConnectionStatus == "Connected")
17      {
18          pushUserTableLog.Log = "La aplicación no estaba a la espera de notificaciones tile o toast!!";
19          pushUserTableLog.insert();
20      }
21      if (_mPNSActive && _notificationDeviceConnectionStatus == "TempDisconnected")
22      {
23          pushUserTableLog.Log = strFmt("No se pudo enviar el mensaje, el móvil del usuario %1 está temporalmente sin acceso", inbox.UserId);
24          pushUserTableLog.insert();
25      }
26      if (_mPNSActive && _notificationChannelStatus == "Expired")
27      {
28          pushUserTableLog.Log = strFmt("El servicio push para el usuario %1 ha expirado", inbox.UserId);
29          pushUserTableLog.insert();
30      }
31      if (_mPNSActive && _notificationChannelStatus == "N/A")
32      {
33          pushUserTableLog.Log = strFmt("El servicio push para el usuario %1 puede estar temporalmente fuera de servicio", inbox.UserId);
34          pushUserTableLog.insert();
35      }
36  }
```

En la línea 12 se verifica si la APP estaba esperando un Tile, en caso contrario se envía el Toast, luego se procesa la respuesta del servicio MPNS detallados en el punto 7.2 para insertar los registros necesarios en la tabla de log detallada en el punto 6.1.

A continuación se explica cómo modificar el estándar de Dynamics AX para que ejecute la clase SendPushNotification. Lo que hay que hacer es localizar en el código dónde se insertan nuevos registros en la tabla EventInbox, dado que será en ese punto donde se deberá generar la notificación a la aplicación móvil. ¿Cómo localizar el punto exacto?

Ya se ha hablado anteriormente del objeto tabla en Dynamics AX, este objeto tiene el evento insert que se lanza cada vez que se inserta un registro en la tabla. Dado que por el estándar, el método no está sobre-escrito el primer paso será anular el método (Ilustración 86):

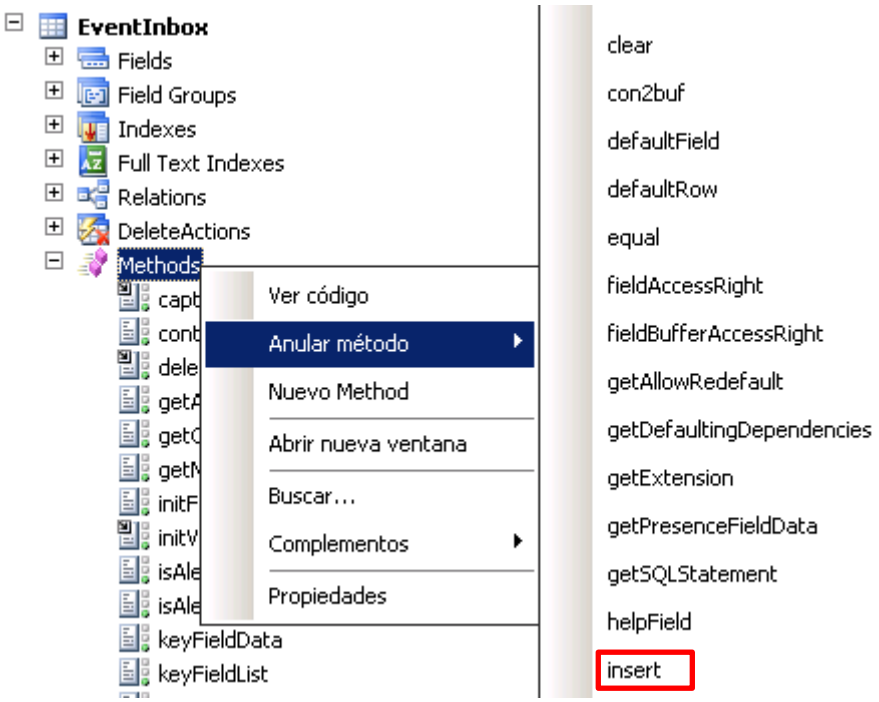



Ilustración 86. Anular método insert de la tabla EventInbox

Una vez anulado se puede poner un punto de interrupción en el evento (Ilustración 87):

```
public void insert()
{
    super();
}
```

Ilustración 87. Punto de interrupción en el método insert

De manera que al depurar el código podemos observar cómo el método es llamado desde la clase EventActionAlert, en el método execute (Ilustración 88), cada vez que se genera una alerta nueva:

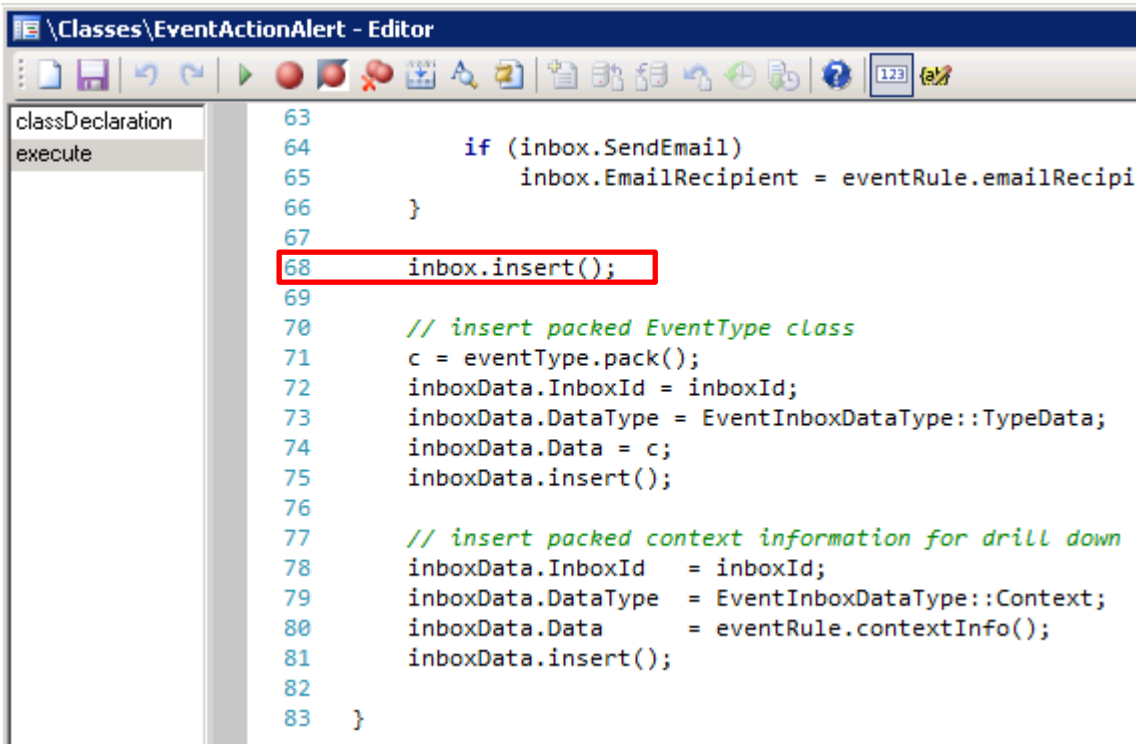



Ilustración 88. Inserción del registro EventInbox durante la generación de la alerta

Se puede pensar que el mejor punto donde lanzar la clase que genere las alertas es precisamente en el evento insert de la tabla EventInbox, dado que, sea desde donde sea que se inserte un registro, pasaría por tal evento.

Este razonamiento es erróneo, porque tal y como se puede observar en la clase EventActionAlert, cuando se inserta el registro aún no se han insertado los datos relacionados con la alerta en la tabla inboxData, que contiene información indispensable sobre el registro y los datos que han generado la alerta en Dynamics AX, con lo que faltaría información indispensable para enviar la notificación a la aplicación móvil.

### *7.5.3. Uso de eventos para modificar el código estándar.*

Una vez localizado el punto donde se debe hacer la llamada a la clase hay que realizar la modificación, existen dos opciones:

1. Modificar el método execute de la clase EventActionAlert, de manera que al finalizar el método se realice la llamada a la clase.
2. Añadir un evento en el método execute de la clase EventActionAlert, de manera que al finalizar la ejecución del método se realice la llamada a la clase.

La primera opción es válida, pero implica modificar código estándar de la aplicación, lo que puede añadir complejidad al migrar la aplicación a futuras versiones debiendo hacer adaptaciones del código.

La segunda opción es una mejora incluida en la versión de Dynamics AX 2012, esta permite añadir eventos en los métodos de las clases (Ilustración 89), el evento permite ejecutar un método estático al inicio o al final de la ejecución del mismo así como capturar los parámetros de entrada.

Esto facilita la migración a futuras versiones, dado que lo único que hay que asegurar es que la llamada al método estático siga funcionando y el código estándar no tiene que ser modificado.

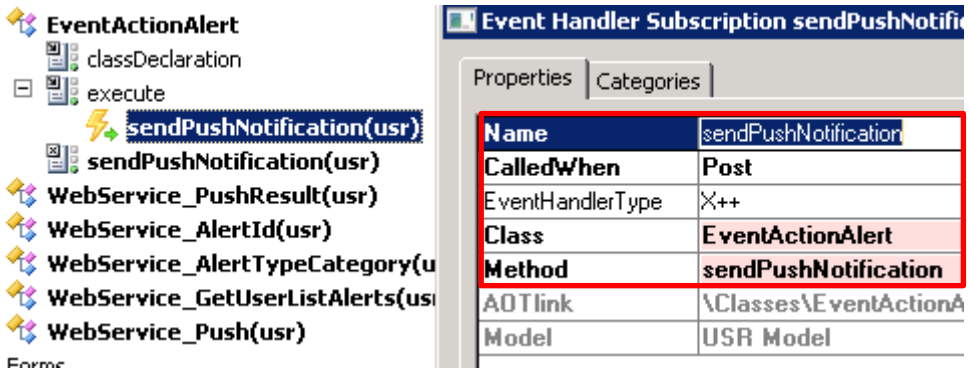



Ilustración 89. Uso de EventHandler

En el método execute se añade el evento “sendPushNotification” (Ilustración 90), se especifica:

1. Called When: “Post”, al finalizar el método se ejecuta el evento, dado que cuando este acaba, la alerta de Dynamics ya se ha generado.
2. Class: clase donde se ubica el método que ha de ejecutar el evento, en este caso el método se añade en la misma, “EventActionAlert”
3. Method, método a ejecutar de la clase especificada en el método 2, en este caso sendPushNotification.

```
1  public static void sendPushNotification(XppPrePostArgs _args)
2  {
3      EventInbox              inbox = _args.getArg("inbox");
4      SendPushNotification::construct(inbox).sendTile();
5  }
```

Ilustración 90. Método EventHandler sendPushNotification

En el método sendPushNotification se captura el parámetro de entrada “inbox” del método execute, que contiene el registro “EventInbox” con la información de la alerta, y a continuación se realiza la llamada al constructor de la clase “SendPushNotification” y se ejecuta el método sendTile que inicia el envío de la notificación al móvil del usuario.

# 8. Windows phone APP.

Ahora que la aplicación Dynamics AX ya es capaz de enviar tanto la información de las alertas mediante un servicio web como las notificaciones mediante el servicio MPNS se explica a continuación la aplicación desarrollada para Windows Phone 7.1.

La aplicación deberá de constar de las siguientes pantallas:

1. Pantalla donde configurar el usuario y password para acceder a la información de las alertas.
2. Pantalla principal con el listado de alertas categorizada por tabla.
3. Pantalla con el detalle de la información de la alerta seleccionada.

Para crear la aplicación desde Visual Studio 2010 hay que seleccionar un proyecto nuevo de tipo “Aplicación de Windows Phone” (Ilustración 91)

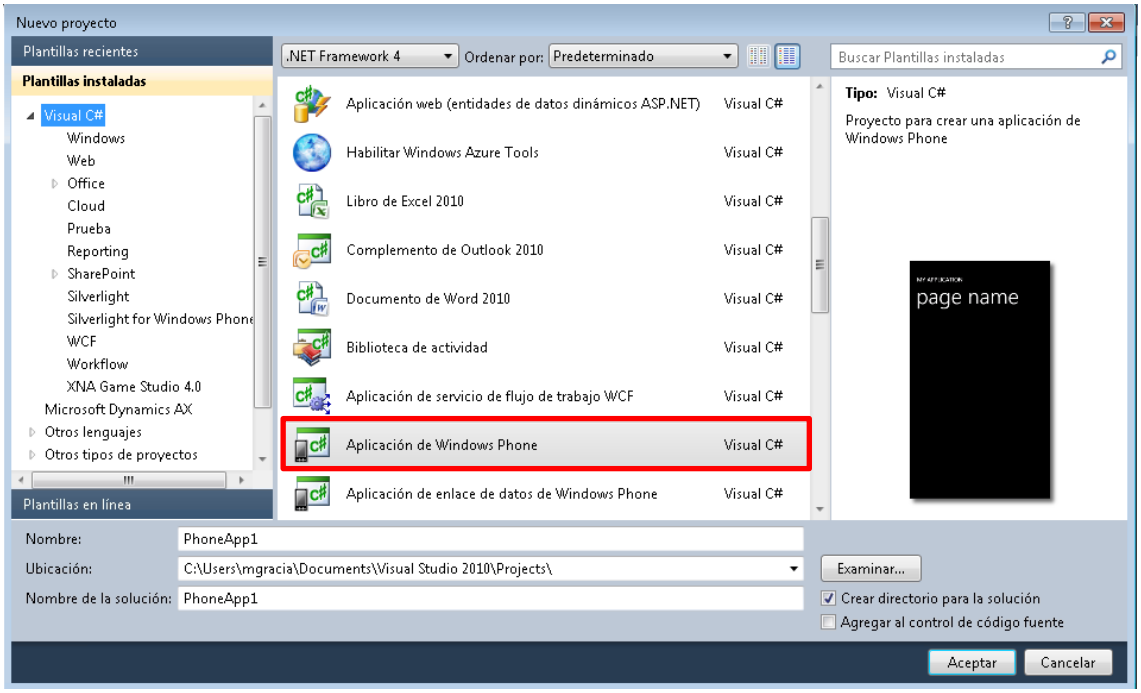



Ilustración 91. Creación de aplicación de Windows Phone en Visual Studio

Una vez seleccionado el tipo de proyecto hay que seleccionar la versión de la aplicación de Windows Phone (Ilustración 92), en este caso 7.1


Ilustración 92. Selección de la versión de Windows Phone

En el proyecto de Visual Studio crea los objetos mínimos y necesarios para crear una sencilla aplicación con una pantalla principal (MainPage.axml), un icono (ApplicationIcon.png) y una imagen (SplashScreenImage.jpg) que se presenta durante la carga de la APP.

## 8.1. Conceptos generales sobre el aspecto visual

Para el diseño visual de la aplicación se intentará hacer una APP lo más estándar en cuanto al cumplimento de las prácticas recomendadas por Microsoft.

### 8.1.1. Nombre de la aplicación

El nombre de la aplicación se encuentra localizado en el archivo WMAppManifest.xml (Ilustración 93):

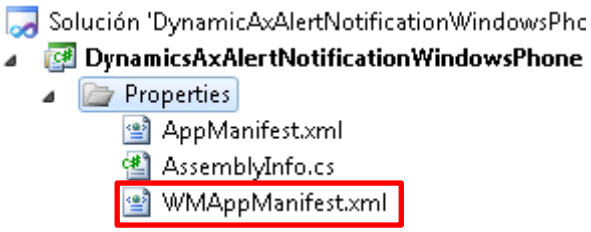



Ilustración 93. WMAppManifest

hay dos puntos en el que indicar el nombre, ya que el primero se mostrará en el listado de aplicaciones y el segundo en el propio icono en el caso de anclarlo en la pantalla principal.

```
<?xml version="1.0" encoding="utf-8"?>
<Deployment xmlns="http://schemas.microsoft.com/windowsphone/2009/deployment" AppPlatformVersion="7.1">
  <App xmlns="" ProductID="{49c9df9c-784e-471d-acf0-616e317238ce}" Title="Alertas Dynamics AX" RuntimeType="
    <IconPath IsRelative="true" IsResource="false">ApplicationIcon.png</IconPath>
    <Capabilities>...</Capabilities>
    <Tasks>...</Tasks>
    <Tokens>
      <PrimaryToken TokenID="DynamicsAxAlertNotificationWindowsPhoneToken" TaskName="_default">
        <TemplateType5>
          <BackgroundImageURI IsRelative="true" IsResource="false">ApplicationIcon.png</BackgroundImageURI>
          <Count>0</Count>
          <Title>Alertas Dynamics</Title>
        </TemplateType5>
      </PrimaryToken>
    </Tokens>
  </App>
</Deployment>
```

Ilustración 94. Edición del título de la aplicación

El nombre que aparece en el listado de aplicaciones se indica en la propiedad “Title” (Ilustración 94) dentro de la etiqueta APP:


El segundo, se especifica en la etiqueta Title (Ilustración 94) que forma parte de TemplateType5, en este caso dado que el icono tiene un tamaño limitado se ha acortado el nombre de la APP dejándolo en “Alertas Dynamics”.

Si se presiona durante unos segundos el icono de la aplicación desde el listado aparece un menú

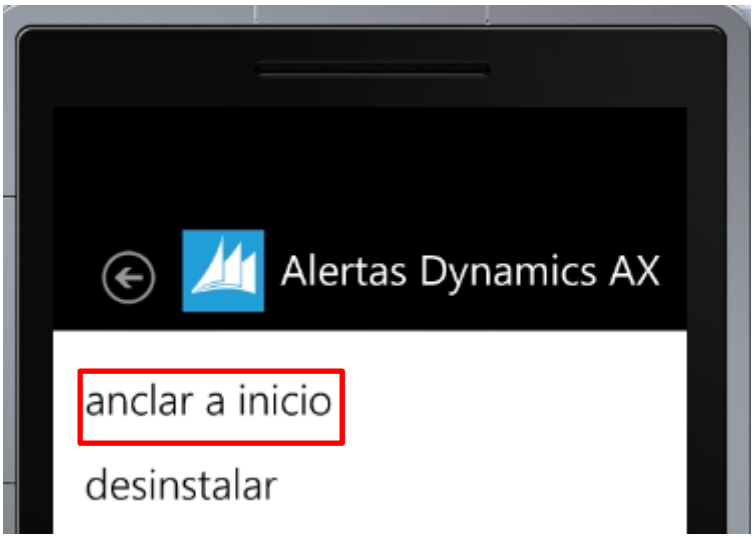



Ilustración 95. Anclar a inicio el Tile de la aplicación

Se selecciona anclar a inicio (Ilustración 95), y se puede observar el resultado (Ilustración 96)

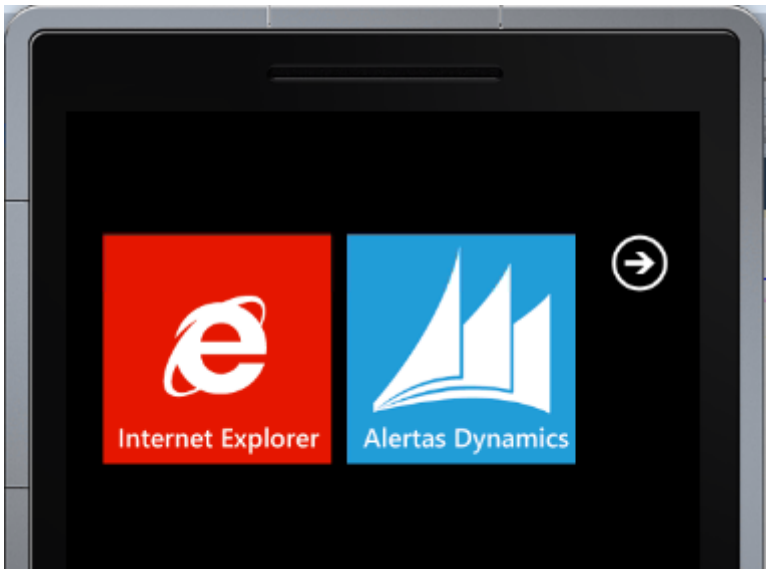



Ilustración 96. Tile en la pantalla de inicio de Windows Phone

En el proyecto que se presenta el icono de la aplicación juega una parte importante en la notificación de tipo Tile, ya que aparecerá en la parte superior derecha un número que representa el número de alertas pendientes de leer y un segundo icono para enfatizar el envío de nuevas alertas desde Dynamics AX.

### 8.1.2. Tile y splashscreen.

El icono o tile debe tener un tamaño de 252x252 píxeles. Se puede editar el icono con cualquier programa de tratamiento de imágenes y guardarlo con el mismo nombre “ApplicationIcon.png” y formato dentro de la carpeta del proyecto sobre-escribiendo el existente.

Lo mismo se puede hacer con la imagen de fondo que aparece cuando carga la aplicación (Ilustración 97), en este caso se trata de una imagen de 480x800 píxeles, al igual que en el caso del tile, se puede editar y guardar la imagen sobre-escribiendo la existente “SplashScreenImage.jpg”

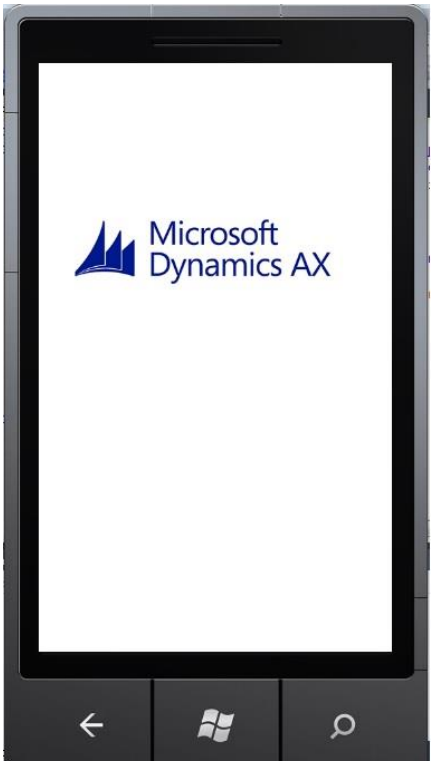



Ilustración 97. Splash screen

### 8.1.3. Navegación y botones de acción.

Windows Phone lleva dos botones de navegación por defecto (Ilustración 98), uno para navegar hacia atrás y el otro para ir a la pantalla inicial del teléfono:

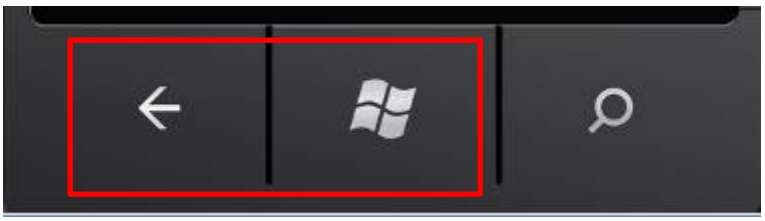


Ilustración 98. Botones de navegación

Para navegar entre las distintas pantallas de la aplicación o añadir un menú con botones para realizar acciones se usará el objeto ApplicationBar (13)http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff431813(v=vs.105).aspx). Este objeto añade una barra en la base del formulario.

Para agregarlo hay que editar el fichero axml asociado al formulario, en este fichero se edita el aspecto gráfico del formulario. En el siguiente ejemplo se ve el ApplicationBar (Ilustración 99) incluido en la pantalla principal de la aplicación:

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
        <shell:ApplicationBarIconButton IconUri="/Images/refresh.png" Click="refresh_Click" Text="actualizar"/>
        <shell:ApplicationBarIconButton IconUri="/Images/settings.png" Click="settings_Click" Text="configurar"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Ilustración 99. Barra de menú en axml

Las propiedades de la barra de menú son:

1. IsVisible, para hacerla visible hay que ponerla a True
2. IsMenuEnabled, si se indica a True al expandir la barra de menus hace aparecer los puntos de menú.

En la barra de menú se agregan los botones, por cada s indica:

1. IconUri, el icono que representa el punto de menú.
2. Click, indica el método al lanzarse el evento de seleccionar el punto de menú.
3. Text, texto que aparecerá debajo del punto de menú cuando este se expande.

### *8.2. Comunicación con el servicio web.*

La aplicación de Windows Phone deberá usar el servicio web explicado anteriormente para poderse identificar y consultar los datos de las alertas generados en Dynamics AX.

Para ello se debe agregar la referencia del servicio al proyecto, desde “Service references” se selecciona “Agregar referencia de servicio…”

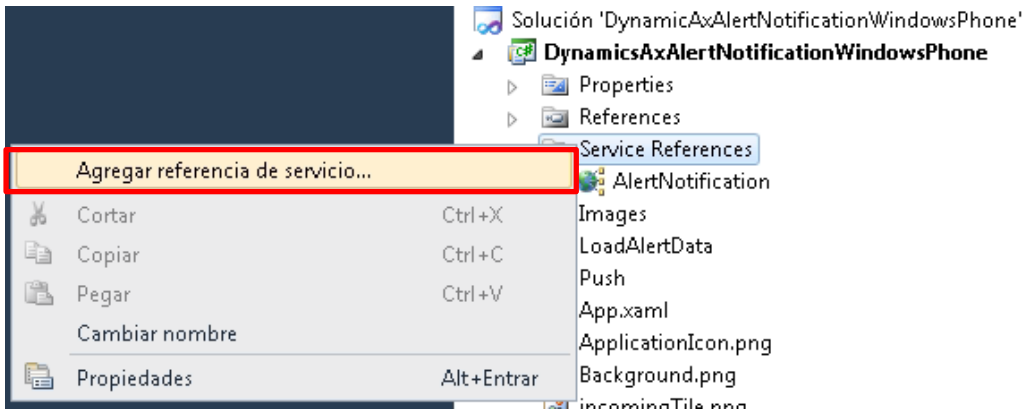



Ilustración 100. Agregar referencia de servicio web en la aplicación de Windows Phone

Y en la configuración se indica la dirección del servicio web (Ilustración 101) y el nombre que se usará, en este caso “AlertNotification”

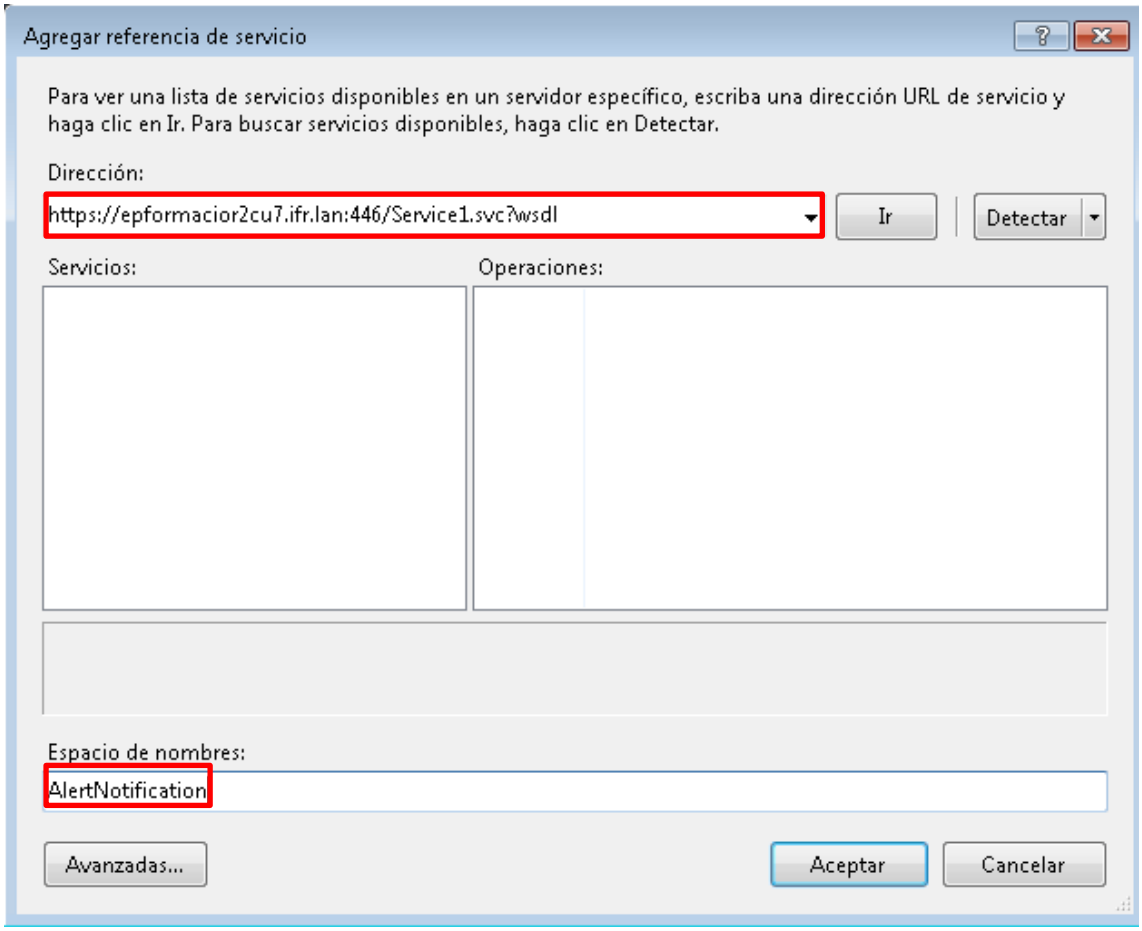



Ilustración 101. Configurar la referencia del servicio web programado en WCF

Una vez agregada la referencia al servicio web ya se pueden acceder a las funciones publicadas para la consulta de alertas y el usuario del móvil puede actualizar el canal URI del servicio MPNS en Dynamics AX.

Al iniciar la aplicación siempre se seguirá el mismo algoritmo (Ilustración 102) que se muestra a continuación:

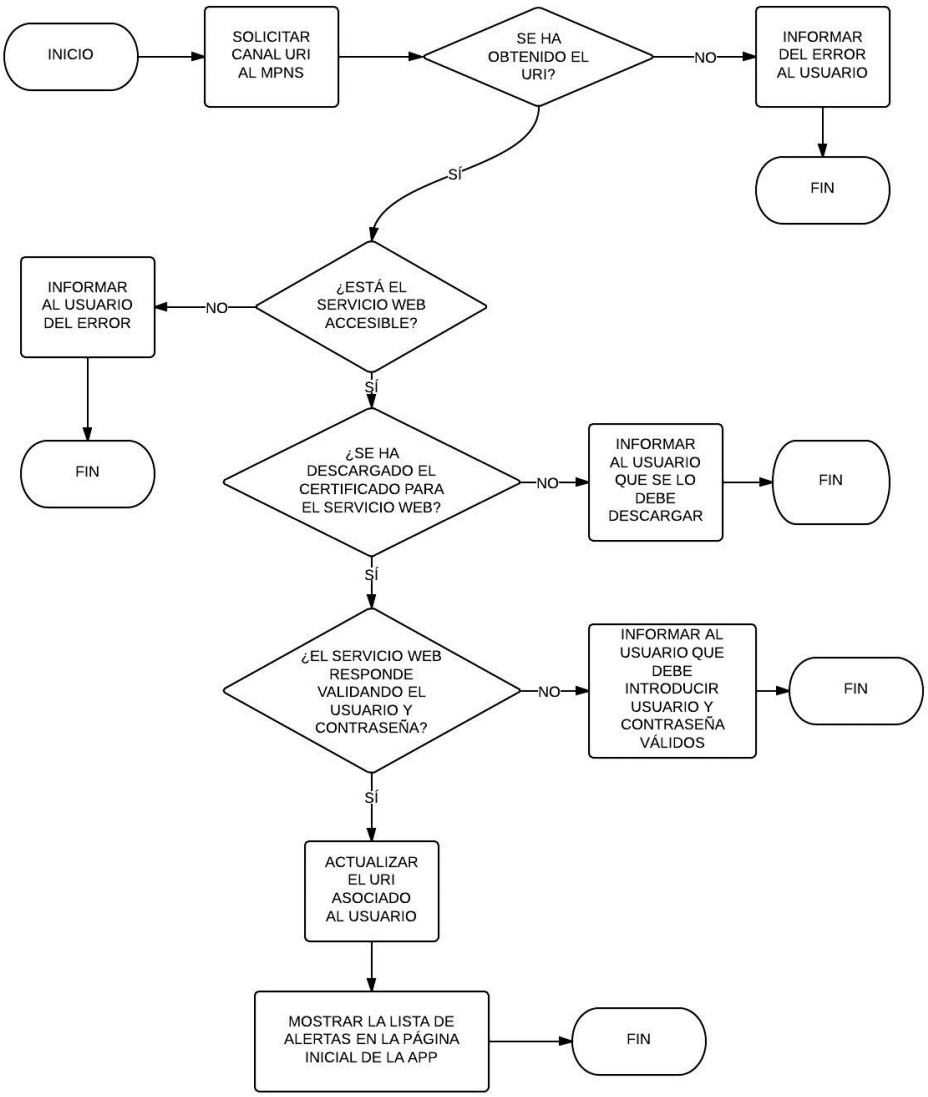



**Ilustración 102. Algoritmo al iniciar la aplicación de Windows Phone**

## *8.3. Obtener e instalar el certificado.*

Para poder acceder al servicio web se ha de instalar el certificado creado para tal fin y así cifrar las comunicaciones.

Para instalar el certificado se usará el navegador web para acceder al sitio web donde éste se ha dejado comprimido en un archivo zip (Ilustración 103).

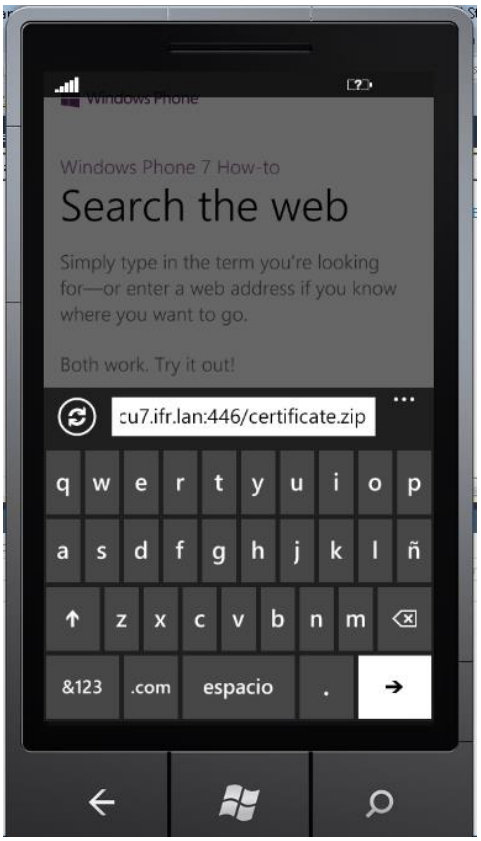



Ilustración 103. Navegar a la url con el certificado del servicio web

La dirección en este caso es https://epformacior2cu7.ifr.lan:446/certificate.zip

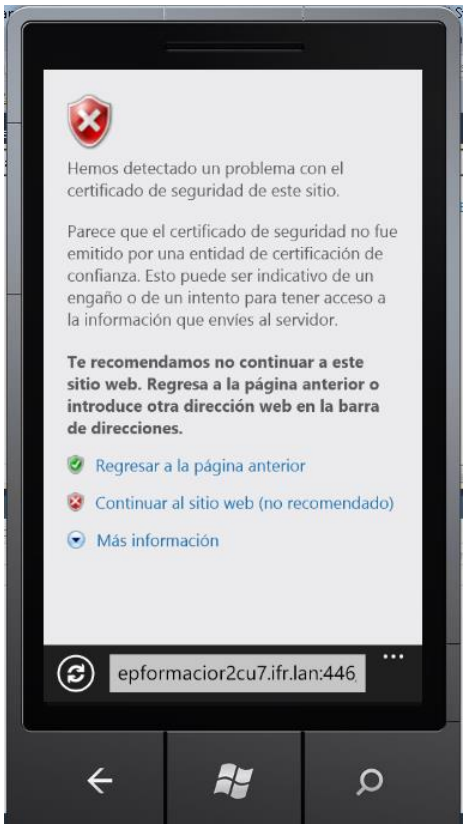



Ilustración 104. Aviso del navegador de certificado

Al acceder al sitio web sale un aviso de problema con el certificado (Ilustración 104), dado que éste aún no se ha instalado, se selecciona “Continuar al sitio web”. Con lo que ya aparece el fichero para descargar

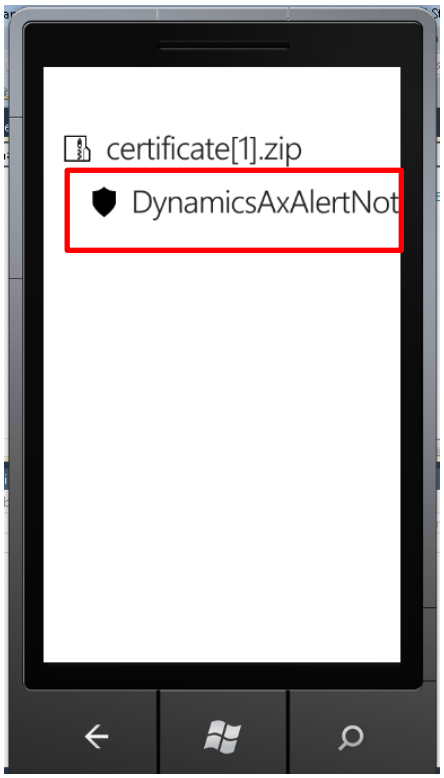



Ilustración 105. Seleccionar archivo de certificado

Se selecciona (Ilustración 105) para iniciar la descarga e instalar (Ilustración 107) el certificado:

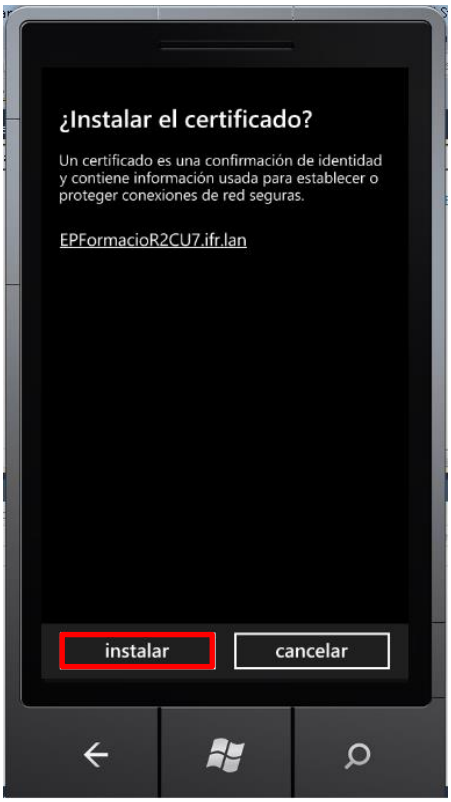



Ilustración 106. Instalación de certificado


Ilustración 107. Confirmación de instalación del certificado

### 8.3.1. Controlar la excepción si no hay certificado.

El servicio web no está accesible en caso de no tener el certificado instalado, con lo que al acceder a las funciones del servicio se produce una excepción que hay que controlar (Ilustración 108).

La excepción se controlará en las funciones que actualizan el URI del servicio MPNS y la que devuelve el listado de alertas, para ello hay que editar el archivo Reference.cs generado al agregar la referencia del servicio web.

```
public DynamicsAxAlertNotificationWindowsPhone.AlertNotification.WebService_PushResult EndupdateUserPushURI
    object[] _args = new object[0];
    try
    {
        DynamicsAxAlertNotificationWindowsPhone.AlertNotification.WebService_PushResult _result = ((Dynamic
        return _result;
    }
    catch (System.ServiceModel.EndpointNotFoundException e)
    {
        DynamicsAxAlertNotificationWindowsPhone.AlertNotification.WebService_PushResult _result = new WebSe
        _result.ProcessOk = false;
        _result.ProcessTxt = "Configure la aplicación y descargue el certificado";

        return _result;
    }
```

```
public DynamicsAxAlertNotificationWindowsPhone.AlertNotification.WebService_PushResult EndgetAlertTypeList
    object[] _args = new object[0];
    try
    {
        DynamicsAxAlertNotificationWindowsPhone.AlertNotification.WebService_PushResult _result = ((Dynami
        return _result;
    }
    catch (System.ServiceModel.EndpointNotFoundException e)
    {
        DynamicsAxAlertNotificationWindowsPhone.AlertNotification.WebService_PushResult _result = new WebS
        _result.ProcessOk = false;
        _result.ProcessTxt = "Configure la aplicación y descargue el certificado";

        return _result;
    }
```

Ilustración 108. Control de la excepción de certificado

Ambas funciones devuelven un objeto de tipo WebService_PushResult, la excepción lo devolverá indicando:

1. ProcessOk , false para indicar que ha habido un error.
2. ProcessTxt, texto que aparecerá en la APP indicando al usuario que debe descargarse el certificado.


Ilustración 109. Mensaje de aviso de certificado en Windows Phone

### *8.4. Autentificar el usuario.*

Tal y como se ha explicado anteriormente el usuario se debe autentificar para poder acceder a los datos y consultar las alertas que tenga vinculadas. En caso contrario se le notificará en la APP que debe configurar un usuario y contraseña válidos (Ilustración 110).


Ilustración 110. Mensaje de aviso de usuario o contraseña incorrectos

### 8.4.1. Aspecto visual de la pantalla.

Se creará una nueva pantalla donde configurar y poder guardar el usuario y contraseña, el acceso se añade en la pantalla principal de la aplicación mediante un punto de menú.

El objetivo es, una vez introducidos los datos, navegar a la pantalla principal y comunicarlos mediante el servicio web al Dynamics AX, que éste los valide, y en caso de ser correctos actualizar el canal URI vinculándolo al usuario. A partir de entonces el usuario podrá consultar los datos de las alertas.

Para agregar una nueva pantalla desde el proyecto se debe seleccionar agregar un nuevo elemento (Ilustración 111):

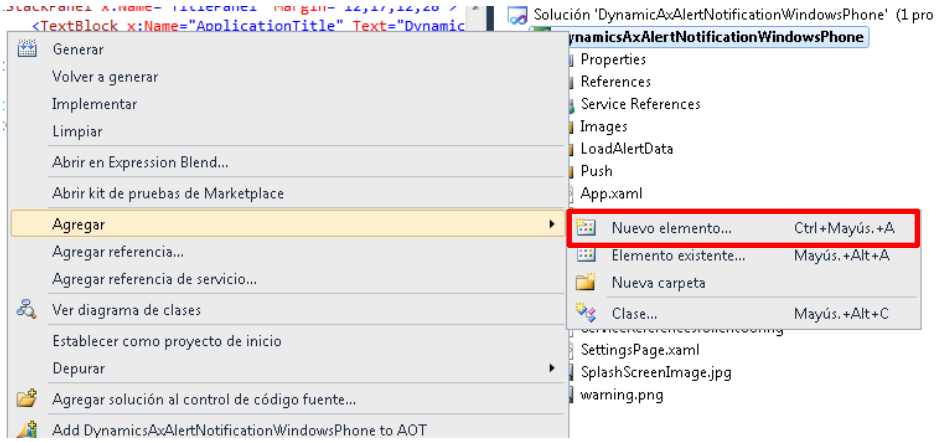



Ilustración 111. Agregar nueva pantalla a la aplicación desde Visual Studio

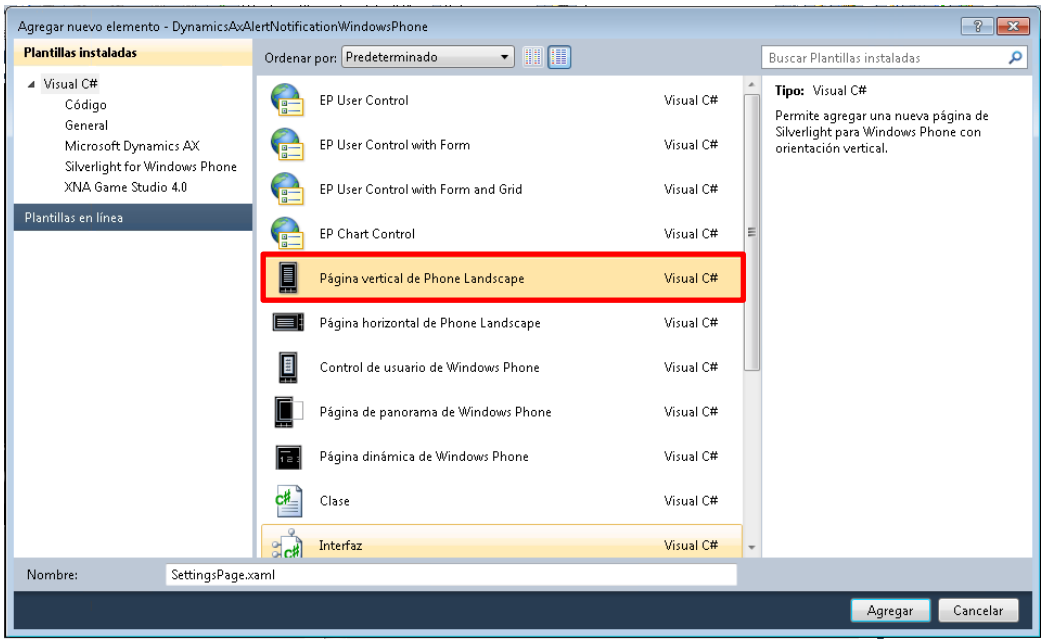



Ilustración 112. Selección tipo de página vertical

El elemento a escoger es una página vertical (Ilustración 112), se le da nombre, en este caso SettingsPage.axml. En el diseño tiene el siguiente aspecto (Ilustración 113):


Ilustración 113. Diseño de la página de configuración

En la parte superior, en el objeto TitlePanel de tipo StackPanel se agrega un objeto de tipo TextBlock con el título de la pantalla, en este se inicializa la propiedad Text con el valor “Configuración” (Ilustración 114)

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="PageTitle" Text="Configuración" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>
```

**Ilustración 114. Código para definir del título de la pantalla de configuración**

En la parte central, se añaden dos campos de tipo texto para introducir el usuario y contraseña, además de los títulos para orientar al usuario. Estos objetos se agregan en el objeto ContentPanel de tipo Grid, éste es añadido automáticamente en el diseño cuando se ha seleccionado al crear la pantalla, abarca toda la parte central, entre el título en la parte superior y la barra de menú en la parte inferior.

Para el campo usuario se agrega un control de tipo TexBox (Ilustración 115), el nombre para éste es TextBoxUsername, en la parte superior se agrega otro de tipo TextBlock, la propiedad Text se inicializa con “Usuario”:

```
<TextBlock HorizontalAlignment="Left" Margin="65,12,0,521" Name="textBlock1" Text="Usuario" Width="169" />
<TextBox Height="78" HorizontalAlignment="Left" Margin="60,34,0,0" Name="textBoxUsername"
```

**Ilustración 115. Código para agregar un textbox para la introducción del usuario**

Para el campo contraseña se agrega un control de tipo PasswordBox, el nombre para éste es passwordBoxPassword, no es de tipo TextBox ya que de esta manera el texto introducido es sustituido por puntos para hacerlo ilegible al ser visualizado, al igual que en el usuario se agrega otro objeto de tipo TextBlock, la propiedad Text se inicializa con “Contraseña”:

```
<TextBlock Height="27" HorizontalAlignment="Left" Margin="65,118,0,0" Name="textBlock2" Text="Contraseña"
<PasswordBox Height="78" HorizontalAlignment="Left" Margin="60,151,0,0" Name="passwordBoxPassword"
```

**Ilustración 116. Código para agregar un PasswordBox para la introducción de la contraseña**

Finalmente en la parte inferior se agrega una barra de menú para añadir dos menú ítems, uno representa la acción “Aceptar” con lo que se guardan los datos y se vuelve a la pantalla principal, el otro representa la acción “Cancelar”, no se guardan los datos e igualmente se navega a la pantalla principal:

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
        <shell:ApplicationBarIconButton IconUri="/Images/check.png" Click="doneButton_Click" Text="aceptar"/>
        <shell:ApplicationBarIconButton IconUri="/Images/cancel.png" Click="cancelButton_Click" Text="cancelar"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

**Ilustración 117. Código para agregar una barra de menú para guardar los cambios**

En cada uno de los puntos de menú se ha especificado la imagen, el método a ejecutar el evento de seleccionarlos y el texto que aparece en la parte inferior.

En el caso de aceptar, el código del evento es el siguiente (Ilustración 118):

```
void doneButton_Click(object sender, EventArgs e)
{

    settings.UsernameSetting = textBoxUsername.Text;
    settings.PasswordSetting = passwordBoxPassword.Password;
    settings.Save();

    NavigationService.Navigate(new Uri("/MainPage.xaml", UriKind.Relative));
}
```

Ilustración 118. Código para guardar los cambios en la pantalla de configuración

Se guarda el objeto settings (éste se detalla más adelante) con los textos introducidos en el textBox (usuario) y PasswordBox (contraseña). Con NavigationService (Ilustración 119) se navega a la pantalla principal MainPage.axml

En el caso de cancelar, el código del evento es prácticamente igual, sólo tiene el código para navegar a la pantalla principal:

```
void cancelButton_Click(object sender, EventArgs e)
{
    NavigationService.Navigate(new Uri("/MainPage.xaml", UriKind.Relative));
}
```

Ilustración 119. Código para navegar a la pantalla principal

### *8.4.2. Como guardar los datos de usuario y contraseña.*

Para guardar los datos de usuario y contraseña se usa el objeto IsolatedStorageSettings (14)http://msdn.microsoft.com/en-us/library/bdts8hk0(v=vs.95).aspx), este objeto permite crear archivos en los cuales guardar datos de parametrización de una forma sencilla. El sistema de archivos es virtual y en todo momento es manejado por el sistema operativo, nunca por el usuario. Dado que el archivo es de tipo xml se puede guardar todo aquello que sea serializable. Tal y como indica el nombre del objeto, este se encuentra aislado del resto de aplicaciones y sólo es accesible desde la propia aplicación que lo crea y guarda, en todo momento se pueden recuperar los datos cuando ésta se cierra y se vuelve a abrir.

En el archivo SettingsPage.xaml.cs donde está definida la clase para la pantalla de configuración, se agrega la clase AppSettings (Ilustración 120)

```
public class AppSettings
{
    // Our settings
    public IsolatedStorageSettings settings;

    // The key names of our settings
    const string UsernameSettingKeyName = "UsernameSetting";
    const string PasswordSettingKeyName = "PasswordSetting";

    // The default value of our settings
    const string UsernameSettingDefault = "";
    const string PasswordSettingDefault = "";
```

Ilustración 120. Clase AppSettings para guardar datos de usuario y contraseña

En esta clase se declara:

1. El objeto settings de tipo IsolatedStorageSettings para guardar la parametrización.
2. Dos constantes de tipo string para definir los nombres clave para poder vincular los valores introducidos al usuario y contraseña.
3. Dos constantes para inicializar los valores por defecto para usuario y contraseña, en este caso de tipo string en blanco para ambos.

Los métodos:

1. El constructor (Ilustración 121):

```
public AppSettings()
{
    // Get the settings for this application.
    if (!System.ComponentModel.DesignerProperties.IsInDesignTool)
    {
        settings = IsolatedStorageSettings.ApplicationSettings;
    }
}
```

Ilustración 121. Constructor de la clase AppSettings

   Se crea o inicializa el objeto settings a partir del objeto IsolatedStorageSettings asociado a la aplicación.
2. AddOrUpdateValue, este método sirve para inicializar o actualizar uno de los dos valores, usuario o contraseña

```
public bool AddOrUpdateValue(string Key, Object value)
{
    bool valueChanged = false;

    // If the key exists
    if (settings.Contains(Key))
    {
        // If the value has changed
        if (settings[Key] != value)
        {
            // Store the new value
            settings[Key] = value;
            valueChanged = true;
        }
    }
    // Otherwise create the key.
    else
    {
        settings.Add(Key, value);
        valueChanged = true;
    }
    return valueChanged;
}
```

Ilustración 122. Código para actualizar usuario o contraseña

   el parámetro de entrada Key ha de tener una de las dos constantes que se han usado

para definir la clave de usuario o contraseña, value contiene el valor que se vinculará a la clave.

3. GetValueOrDefault (Ilustración 123), método genérico que devuelve el valor vinculado a la clave declarada en la clase. En caso de no existir valor devuelve el valor por defecto.

```
public T GetValueOrDefault<T>(string Key, T defaultValue)
{
    T value;

    // If the key exists, retrieve the value.
    if (settings.Contains(Key))
    {
        value = (T)settings[Key];
    }
    // Otherwise, use the default value.
    else
    {
        value = defaultValue;
    }
    return value;
}
```

Ilustración 123. Código para devolver el valor del usuario o contraseña

4. Save (Ilustración 124), guarda los valores introducidos.

```
public void Save()
{
    settings.Save();
}
```

Ilustración 124. Código para guardar los cambios

5. UsernameSetting (Ilustración 125), se usa este método para asignar o devolver el valor vinculado a la clave del usuario.

```
public string UsernameSetting
{
    get
    {
        return GetValueOrDefault<string>(UsernameSettingKeyName, UsernameSettingDefault);
    }
    set
    {
        if (AddOrUpdateValue(UsernameSettingKeyName, value))
        {
            Save();
        }
    }
}
```

Ilustración 125. Código para asignar o devolver el usuario

6. PasswordSetting (Ilustración 126), se usa este método para asignar o devolver el valor vinculado a la clave de la contraseña

```
public string UsernameSetting
{
    get
    {
        return GetValueOrDefault<string>(UsernameSettingKeyName, UsernameSettingDefault);
    }
    set
    {
        if (AddOrUpdateValue(UsernameSettingKeyName, value))
        {
            Save();
        }
    }
}
```

Ilustración 126. Código para asignar o devolver la contraseña

Ahora que ya se dispone del objeto AppSettings, éste se ha de inicializar al abrir el formulario, y los controles que se han declarado para indicar el usuario y contraseña se han de vincular a los métodos UsernameSettings y PasswordSettings. De esta manera al abrir el formulario se enseñarán los datos guardados para la aplicación y estos pueden ser inicializados, modificados o guardados.

Para ello en el fichero SettingsPage.axml, donde se define el diseño del formulario se crea un recurso de tipo AppSettings (Ilustración 127):

```
<phone:PhoneApplicationPage.Resources>
    <local:AppSettings x:Key="appSettings"></local:AppSettings>
</phone:PhoneApplicationPage.Resources>
```

Ilustración 127. Código para agregar la clase AppSettings como recurso

Se vincula al control textBoxUsername el método UsernameSetting del objeto appSettings (Ilustración 128):

```
<TextBox Height="78" HorizontalAlignment="Left" Margin="60,34,0,0" Name="textBoxUsername"
Text="{Binding Path=UsernameSetting, Mode=OneWay, Source={StaticResource appSettings}}" Vertic
```

Ilustración 128. Código para vincular el control de texto para la introducción del usuario a la clase AppSettings

Para ello hay que indicarle en la propiedad Text el nombre de la función y el recurso que se usa.

Se hace lo mismo para el control passwordBoxPassword (Ilustración 129):

```
<PasswordBox Height="78" HorizontalAlignment="Left" Margin="60,151,0,0" Name="passwordBoxPassword"
Password="{Binding Path=PasswordSetting, Mode=OneWay, Source={StaticResource appSettings}}" VerticalAl
```

Ilustración 129. Código para vincular el control de texto para la introducción de la contraseña a la clase AppSettings

Sólo que en este caso en vez de usar la propiedad Text, se ha de usar la propiedad Password, indicando el nombre del método PasswordSetting y el recurso appSettings.

### 8.5. Pantalla principal.

La pantalla principal de la aplicación se crea con el nombre MainPage.axml para el diseño y MainPage.axml.cs para la clase vinculada por defecto cuando se crea el proyecto por primera vez. Los objetivos de la pantalla principal son varios:

1. Inicializar el canal push con el servicio MPNS.
2. Comunicar el canal push al servicio web.
3. Mostrar una lista de alertas categorizada por tabla vinculadas al usuario.
4. Poder navegar a la pantalla que muestre en detalle la alerta seleccionada.
5. Poder navegar a la pantalla de configuración de la aplicación.
6. Poder refrescar la información.

#### 8.5.1. Aspecto visual de la pantalla.

El aspecto gráfico de la pantalla es el siguiente (Ilustración 130):

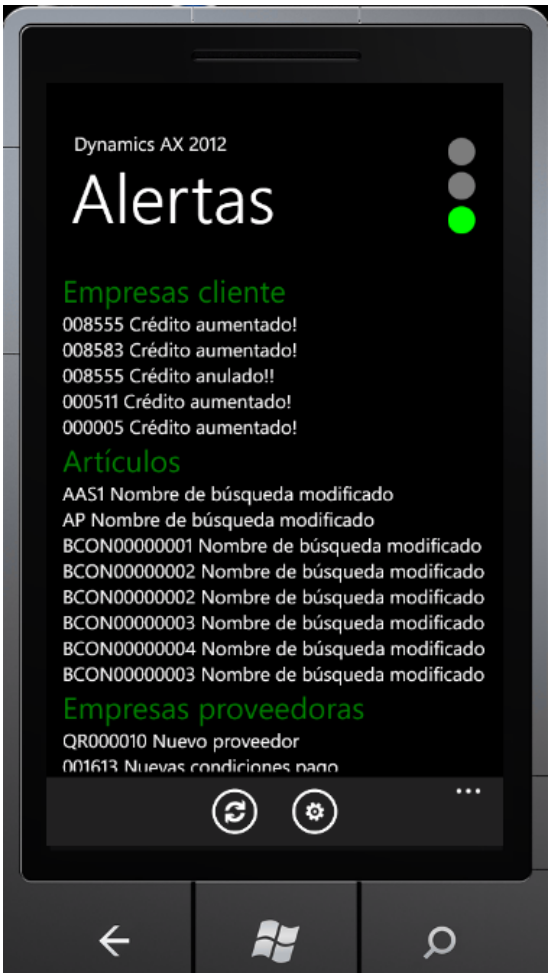



Ilustración 130. Aspecto visual de la pantalla inicial

En la parte superior, en el objeto TitlePanel de tipo StackPanel se agregan dos objetos de tipo TextBlock para mostrar el título de la pantalla (Ilustración 131), el primero con la propiedad Text incializado con “Dynamics AX 2012”, y el segundo de un tamaño de fuente mayor con la propiedad Text inicializado con “Alertas”:

```
<StackPanel x:Name="TitlePanel" Margin="12,17,12,28">
    <TextBlock x:Name="ApplicationTitle" Text="Dynamics AX 2012" Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="Alertas" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}" Width="440" />
</StackPanel>
```

**Ilustración 131. Código para definir el título de la pantalla**

También se añade una imagen a semejanza de un semáforo, para indicar en verde que la app se ha podido conectar con MPNS, o indicando el rojo que todavía no se ha conectado (esta parte se detalla en el punto 8.5.2.

En la parte central de la pantalla se han de mostrar las alertas categorizadas, para ello se usará un objeto de tipo LongListSelector (Ilustración 132), ya que este control permite mostrar largas listas agrupadas (15). Por lo tanto, en el axml donde está definido todo el diseño se añadirá:

```
<toolkit:LongListSelector Name="GroupedList" SelectionChanged="GroupedList_SelectionChanged" Grid.ColumnSpan="2"
    <toolkit:LongListSelector.ItemTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Description}" />
        </DataTemplate>
    </toolkit:LongListSelector.ItemTemplate>
    <toolkit:LongListSelector.GroupHeaderTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Name}" FontSize="32" Foreground="Green" />
        </DataTemplate>
    </toolkit:LongListSelector.GroupHeaderTemplate>
</toolkit:LongListSelector>
```

**Ilustración 132. Código para agregar el objeto LongListSelector en la pantalla**

El objeto LongListSelector con la propiedad Name inicializada a GroupedList, esto permitirá luego llamar al control desde la clase que inicializará la lista, la propiedad SelectionChanged tiene el nombre del método GroupedList_SelectionChanged, este será el evento que se ejecutará al seleccionar un ítem de alguna de las agrupaciones, de esta manera, el usuario ha de poder consultar la alerta en detalle navegando a una nueva pantalla.
GroupHeaderTemplate contiene el control que representa la agrupación, éste está enlazado con Name, esta variable se inicializará con el nombre de la agrupación que devuelva Dynamics AX, la fuente se hace más grande, tamaño 32 y de color verde para que destaque.
ItemTemplate contiene el control que representa las alertas dentro de la agrupación, éste está enlazado con Description, que se inicializará con una breve descripción de la alerta que devuelva Dynamics AX.

Finalmente en la parte inferior se agrega una barra de menú con dos menú ítems (Ilustración 133), uno para poder navegar a la pantalla de configuración y otro para refrescar los datos, por lo que en el diseño se agrega un objeto de tipo ApplicationBar:

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
        <shell:ApplicationBarIconButton IconUri="/Images/refresh.png" Click="refresh_Click" Text="actualizar"/>
        <shell:ApplicationBarIconButton IconUri="/Images/settings.png" Click="settings_Click" Text="configurar"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

**Ilustración 133. Código para agregar una barra de menú**

Donde se definen los dos menú ítems, con sus iconos (con la propiedad IconUri, se especifica el archivo de imagen a utilizar), eventos (se usa el evento Click) y texto a mostrar (con la propiedad Text, que mostrará lo especificado debajo del icono).

Para refrescar, el evento Click es refresh_Click, que hace una llamada al método refreshData (Ilustración 134), el cual usa el objeto LoadAlert para volver a solicitar los datos al servicio web:

```
private void refresh_Click(object sender, EventArgs e)
{
    this.refreshData();
}


private void refreshData()
{
    AppSettings settings = (AppSettings)this.Resources["appSettings"];

    LoadAlertData.LoadAlert loadAlert = new LoadAlertData.LoadAlert(settings.UsernameSetting, settings.PasswordSetting, GroupedList);
    loadAlert.loadData();
}
```

Ilustración 134. Código para refrescar la pantalla

Para navegar a la pantalla de configuración, el evento Click es settings_Click, que realiza la llamada a SettingsPage.axml (Ilustración 135)

```
private void settings_Click(object sender, EventArgs e)
{
    (App.Current.RootVisual as PhoneApplicationFrame).Navigate(
                            new Uri("/SettingsPage.xaml", UriKind.Relative));
}
```

Ilustración 135. Código para navegar a la pantalla de configuración

### *8.5.2. Inicializar el canal push.*

Tal y como se ha comentado en el punto 8.5 y en algoritmo del punto 4.2.2. al iniciar la pantalla principal, si el usuario se ha autentificado de forma correcta, se debe inicializar el canal push. La razón por la cual se hace vez que se entra en la pantalla es para asegurar que se tiene correctamente actualizado el URI asociado al usuario en Dynamics AX, dado que es la pantalla principal, y por lo tanto es ejecutada de forma más frecuente.

Hay que tener en cuenta que no siempre que se solicita el canal push se le da al usuario un nuevo URI, ya que si éste ya tiene uno y no ha caducado (la duración de un canal es de 30 días) se le asigna el mismo, en Dynamics AX cuando se informa el URI, sólo si es distinto al que tiene el usuario, es actualizado.

Para realizar esta tarea se crea una clase nueva, PushAX, que se encargará de:

1. Solicitar el canal push.
2. Actualizar el URI en Dynamics AX usando el servicio web.
3. Actualizar la lista de alertas.

La razón por la cual esta clase se encarga no sólo de solicitar el canal push y actualizar el URI sino también de actualizar la lista de alertas, es porque tal y como se mostró en el algoritmo al iniciar la pantalla, una vez se actualiza el canal URI hay que refrescar los datos.

Hay que tener en cuenta que la comunicación con el servicio web es asíncrona, por lo tanto, solo cuando se ejecute el evento al actualizar el canal URI, se puede ejecutar la clase que se usará para actualizar los datos de las alertas. Si se ejecutara uno después del otro, podría suceder que, antes de ejecutarse el evento que actualiza el canal URI se ejecutara la clase que actualiza los datos.

La clase PushAx se deberá llamar al inicializarse la pantalla principal (Ilustración 136), para lo cual se usará el evento OnNavigatedTo, desde aquí se llamará al método initPushAndRefreshData que construirá y ejecutará la clase.

```
private void initPushAndRefreshData()
{
    AppSettings settings = (AppSettings)this.Resources["appSettings"];

    Push.PushAX push = new Push.PushAX(settings.UsernameSetting, settings.PasswordSetting, Dispatcher, GroupedList);
    push.initPushChannel();
}

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    try
    {
        this.initPushAndRefreshData();
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }

}
```

Ilustración 136. Código para llamar a la clase PushAx

La clase PushAx contiene las siguientes variables públicas

```
public class PushAX
{
    /// Holds the push channel that is created or found.
    HttpNotificationChannel pushChannel;

    // The name of our push channel.
    const string channelName = "DynamicsAXAlertNotification";

    public string userId { get; set; }
    public string password { get; set; }
    public Dispatcher dispatcher { get; set; }
    public LongListSelector groupedList { get; set; }
    public AlertNotification.WebService_PushResult result = new AlertNotification.WebService_PushResult();

    public PushAX()...

    public PushAX(string _userId, string _password, Dispatcher _dispatcher, LongListSelector _groupedList)
```

Ilustración 137. Clase PushAX

1. pushChannel de tipo HttpNotificationChannel, variable para inicializar el canal push.
2. channelName de tipo string, variable con el nombre del canal, necesario para inicializarlo, es constante.
3. userId de tipo string, contiene el usuario configurado en la apliación, necesario para poder autentificarse con DynamicsAX.
4. password de tipo string, contiene la constraseña configurada en la aplicación, necesario para poder autentificarse con DynamicsAX.

5. groupedList de tipo LongListSelector, necesario para poder actualizar la lista mostrada en la pantalla inicial.
6. Result de tipo WebService_PushResult, para devolver si se ha ejecutado correctamente la clase.

El constructor se usa para inicializar las variables locales, el método al que se llama al navegar a la pantalla principal es initPushChannel:

```
public void initPushChannel()
{
    result.ProcessOk = true;

    // Try to find the push channel.
    pushChannel = HttpNotificationChannel.Find(channelName);

    // If the channel was not found, then create a new connection to the push service.
    if (pushChannel == null)
    {
        pushChannel = new HttpNotificationChannel(channelName);

        // Register for all the events before attempting to open the channel.
        pushChannel.ChannelUriUpdated += new EventHandler<NotificationChannelUriEventArgs>(PushChannel_ChannelUriUpdated);
        pushChannel.ErrorOccurred += new EventHandler<NotificationChannelErrorEventArgs>(PushChannel_ErrorOccurred);

        pushChannel.Open();

        // Bind this new channel for Tile events.
        pushChannel.BindToShellTile();

        // Bind this new channel also for Toast events.
        pushChannel.BindToShellToast();
    }
    else
    {
        // The channel was already open, so just register for all the events.
        pushChannel.ChannelUriUpdated += new EventHandler<NotificationChannelUriEventArgs>(PushChannel_ChannelUriUpdated);
        pushChannel.ErrorOccurred += new EventHandler<NotificationChannelErrorEventArgs>(PushChannel_ErrorOccurred);
        pushChannel.ShellToastNotificationReceived += new EventHandler<NotificationEventArgs>(pushChannel_ShellToastNotificationReceived);

        this.updateURIChannelOnAxUser(pushChannel.ChannelUri.ToString());
    }
}
```

**Ilustración 138. Código para solicitar el URI al servicio MPNS**

Este método solicita el canal push con el nombre que se le ha dado, en este caso "DynamicsAXAlertNotification", si no se encuentra se inicializa y se enlaza para esperar notificaciones de tipo Tile y Toast, en caso de encontrarse, se actualiza el URI en Dynamics AX (Ilustración 139) ejecutando el método updateURIChannelOnAXUser.
En cualquiera de los dos casos hay que controlar el evento ChannelUriUpdated, este evento se ejecutará si el servicio MPNS decide actualizar el URI ya bien sea porque se ha solicitado un canal nuevo, o porque haya caducado y el servicio decida darle un nuevo identificador. Por lo tanto en el evento se deberá llamar al método updateURIChannelOnAXUser.

```
private void PushChannel_ChannelUriUpdated(object sender, NotificationChannelUriEventArgs e)
{
    dispatcher.BeginInvoke(() =>
    {
        this.updateURIChannelOnAxUser(e.ChannelUri.ToString());
    });
}

private void PushChannel_ErrorOccurred(object sender, NotificationChannelErrorEventArgs e)...

private void updateURIChannelOnAxUser(string _uRIChannel)
{
    AlertNotification.Service1Client client = new AlertNotification.Service1Client();

    try
    {
        client.updateUserPushURICompleted += new EventHandler<AlertNotification.updateUserPushURICompletedEventArgs>(client_updateUserPushURIOnAxUser);
        client.updateUserPushURIAsync(userId, password, _uRIChannel);
    }
    catch (System.ServiceModel.EndpointNotFoundException e)
    {
        result.ProcessOk = false;
        result.ProcessTxt = e.ToString();
        MessageBox.Show(e.ToString());
    }
    catch (Exception e)
    {
        result.ProcessOk = false;
        result.ProcessTxt = e.ToString();
        MessageBox.Show(e.ToString());
    }
}
```

Ilustración 139. Código para actualizar en canal URI en Dynamics AX

Este método usará el servicio web para llamar al método updateUserPushURIAsync, pasando como parámetros el usuario, contraseña y el URI, para que éste se vincule al usuario en Dynamics AX. El evento vinculado client_updateUserPushURIOnAxUser permitirá controlar cuando se ha ejecutado el método, con lo cual si el resultado es satisfactorio se refrescará la pantalla principal, en caso contrario, se mostrará el error en la pantalla del móvil

```
private void client_updateUserPushURIOnAxUser(object sender, AlertNotification.updateUserPushURICompletedEventArgs e)
{
    dispatcher.BeginInvoke(() =>
    {
        result = e.Result;
        if (result.ProcessOk)
        {
            this.refreshData();
        }
        else
        {
            MessageBox.Show(result.ProcessTxt);
        }
    });
}
```

Será el propio método del servicio web, quien, mediante la variable Result de tipo WebService_PushResult devuelve el resultado de la llamada, ProcessOk estará a true si todo ha ido correctamente, con lo cual se llamará al método refreshData para actualizar los datos de la pantalla principal, en caso contrario se muestra un mensaje por pantalla con el mensaje que haya devuelto la llamada al servicio web, éste se encuentra en ProcessTxt que es de tipo string.

Un error que puede devolver el servicio web es un fallo al identificarse el usuario, con lo que aparecería un mensaje como el que se muestra en el punto 8.4

### *8.5.3. Mostrar una lista de alertas categorizada por tabla.*

Una vez el usuario se autentificado y el URI del canal push se ha vinculado a éste correctamente, se ha de refrescar la lista de alertas que se deben consultar en Dynamics AX mediante el servicio web.

Para inicializar la lista se hará mediante una nueva clase, LoadAlert, cargando los datos en el objeto longListSelector que se ha mostrado en el punto 1. La clase se llamará desde PushAx, cuando se actualice correctamente el URI en Dynamics AX (Ilustración 140):

```
private void client_updateUserPushURIOnAxUser(object sender, AlertNotification.updateUserPushURICompletedEventArgs e)
{
    dispatcher.BeginInvoke(() =>
    {
        result = e.Result;
        if (result.ProcessOk)
        {
            this.refreshData();
        }
        else
        {
            MessageBox.Show(result.ProcessTxt);
        }
    });
}

private void refreshData()
{
    LoadAlertData.LoadAlert loadAlert = new LoadAlertData.LoadAlert(userId, password, groupedList);
    loadAlert.loadData();
}
```

Ilustración 140. Llamada a LoadAlert al actualizar el canal URI

Y cuando el usuario refresque la información de la pantalla principal (Ilustración 141) usando el menú ítem “actualizar” de la barra de menú

```
private void refresh_Click(object sender, EventArgs e)
{
    this.refreshData();
}

private void refreshData()
{
    AppSettings settings = (AppSettings)this.Resources["appSettings"];

    LoadAlertData.LoadAlert loadAlert = new LoadAlertData.LoadAlert(settings.UsernameSetting, settings.PasswordSetting, GroupedList);
    loadAlert.loadData();
}
```

Ilustración 141. Llamada a LoadAlert al actualizar la pantalla principal

En ambos casos se pasará como parámetros el usuario y contraseña para autentificarse con Dynamics AX y el control GroupedList que es de tipo LongListSelector.

La clase LoadAlert (Ilustración 142) tiene las siguientes variables locales

```
public class LoadAlert
{
    public string userId { get; set; }
    public string password { get; set; }
    public LongListSelector groupedList { get; set;}
    public AlertNotification.WebService_PushResult result { get; set; }

    public LoadAlert()
    {
    }

    public LoadAlert(string _userId, string _password, LongListSelector _groupedList)
    {
        userId = _userId;
        password = _password;
        groupedList = _groupedList;
    }
```

Ilustración 142. Clase LoadAlert

1. userId de tipo string, contiene el usuario configurado en la apliación, necesario para poder autentificarse con DynamicsAX.
2. password de tipo string, contiene la constraseña configurada en la aplicación, necesario para poder autentificarse con DynamicsAX.
3. groupedList, se inicializará con el objeto LongListSelector definido en el diseño, para poder actualizar la lista que se mostrará al usuario con las alertas categorizadas que devuelva DynamicsAX.
4. result de tipo WebService_PushResult, para poder indicar si ha ido todo correctamente o si ha habido un error.

El constructor de la clase inicializa las variables.

Para poder enlazar los objetos de la lista de la aplicación del móvil con los datos que devuelve el servicio web se crearán dos clases:

1. AccountRegister (Ilustración 143), contendrá la descripción breve de las alertas:

```
public class AccountRegister
{
    public string AccountNum { get; set; }
    public string Description { get; set; }
    public Int64 RecId { get; set; }

    public AccountRegister(string _accountNum, string _description, Int64 _recId)
    {
        AccountNum = _accountNum;
        Description = _accountNum +" "+_description;
        RecId = _recId;
    }
    public AccountRegister()
    {
        AccountNum = "";
        RecId = 0;
        Description = "";
    }
}
```

Ilustración 143. Clase AccountRegister

   - AccountNum, es el identificador asociado al registro que generó la alerta.
   - Description, es la descripción breve definida en la alerta en Dynamics AX, éste se vincula al ItemTemplate del objeto GroupedList, para que se muestre en pantalla.
   - RecId, contiene el recId de la alerta, esto ayudará a buscar el registro para poder mostrar la información en detalle.

2. AlertCategory (Ilustración 144), contendrá la descripción de la agrupación y una lista de tipo AccountRegister vinculada a ésta.

```
public class AlertCategory : System.Collections.IEnumerable
{
    public string Name { get; private set; }
    public System.Collections.Generic.List<AccountRegister> Items { get; private set; }

    public AlertCategory(string categoryName)
    {
        Name = categoryName;
        Items = new System.Collections.Generic.List<AccountRegister>();
    }

    public void AddAccountRegister(AccountRegister _accountRegister)
    {
        Items.Add(_accountRegister);
    }

    public System.Collections.IEnumerator GetEnumerator()
    {
        return this.Items.GetEnumerator();
    }
}
```

Ilustración 144. Clase AlertCategory

- Name, es el nombre de la agrupación, éste se vincula al GroupHeaderTemplate del objeto GroupedList, para que se muestre en pantalla.
- Items, es la lista de tipo AccountRegister, de esta manera, para cada agrupación se asocia un conjunto de alertas.

Cuando la clase LoadAlert inicialice los objetos AccountRegister y AlertCategory, estos se enlazarán con el objeto LongListSelector que mostrará en pantalla los datos que devuelve DynamicsAX, para ello se crea el método público loadData

```
public void loadData()
{
    AlertNotification.Service1Client client = new AlertNotification.Service1Client();

    try
    {
        client.getAlertTypeListCompleted += new EventHandler<AlertNotification.getAlertTypeListCompletedEventArgs>(client_getAlertTypeListCompleted);
        client.getAlertTypeListAsync(userId, password);
    }
    catch (System.ServiceModel.EndpointNotFoundException e)
    {
        result.ProcessOk = false;
        result.ProcessTxt = e.ToString();
        MessageBox.Show(e.ToString());
    }
    catch (Exception e)
    {
        result.ProcessOk = false;
        result.ProcessTxt = e.ToString();
        MessageBox.Show(e.ToString());
    }
}
```

Ilustración 145. Método LoadData para llamar al servicio web y actualizar la lista

Que llama al método getAlertType del servicio web, pasando como parámetros el usuario y password, una vez se captura el evento (recordar que la conexión es asíncrona), hay que inicializar AccountRegister y AlertCategory:

```
void client_getAlertTypeListCompleted(object sender, AlertNotification.getAlertTypeListCompletedEventArgs e)
{
    List<AlertCategory> alertCategories = new List<AlertCategory>();

    AlertCategory category;
    AccountRegister accountRegister;

    if (e.Error != null)
    {
        Exception ex = e.Error;
        MessageBox.Show(ex.Message);
        return;
    }
    if (e.Result == null)
    {
        MessageBox.Show("No hay incidencias");
        return;
    }
```

Lo primero a comprobar es que el objeto que devuelve el servicio web (de tipo WebService_PushResult) no es null o que no haya error, ya que en caso contrario el método finaliza devolviendo el mensaje de error (en caso que se haya producido) o indicando que no hay alertas (en caso que Result, que contiene la lista de alertas esté a null, es decir, vacía). Si no se da ninguno de los dos casos, se procede a leer Result para inicializar la lista a mostrar en la pantalla del terminal móvil:

```
result = e.Result;

if (result.ProcessOk == true && result.AlertTypeCategoryList != null)
{
    foreach (AlertNotification.WebService_AlertTypeCategory alertType in result.AlertTypeCategoryList)
    {
        category = new AlertCategory(alertType.alertCategoryName);

        foreach (AlertNotification.WebService_AlertId alertId in alertType.AlertIdList)
        {
            accountRegister = new AccountRegister(alertId.AlertId, alertId.AlertDescription, alertId.AlertRecId);
            category.AddAccountRegister(accountRegister);
        }

        alertCategories.Add(category);
    }

    groupedList.ItemsSource = alertCategories;
}
else if (result.ProcessOk == false)
{
    MessageBox.Show(result.ProcessTxt);
}
```

**Ilustración 146. Método para cargar el objeto LongListSelector con la lista de las alertas**

Se inicializa la variable global result con lo que devuelve el evento. Para cada WebService_AlertTypeCategory se inicializa un nuevo AlertCategory, y para cada WebService_AlertId que se encuentra dentro de la categoría se vincula un nuevo AccountRegister a AlertCategory.

Es decir, recorremos las categorías y alertas vinculadas para ir construyendo el objeto alertCagories, este finalmente es asignado a groupedList (el control LongListSelector)que se ha pasado por parámetro, mostrando así al usuario una lista de alertas categorizadas.

# 9. Conclusiones

El objetivo del proyecto no ha sido tan sólo como mostrar cómo se puede aprovechar el protocolo push para poder comunicar alertas desde Dynamics AX 2012 a una aplicación para Windows Phone, si no desarrollar también una aplicación que se pueda usar en el ámbito profesional. Para ello el reto ha sido poder programar un sistema que sea capaz de funcionar en cualquier sistema de Dynamics AX 2012 aunque este contenga múltiples modificaciones. Con la solución aportada, la comunicación de alertas es capaz de enviar la información al usuario aunque se trate de alertas relacionadas con tablas nuevas desarrollas fuera del estándar.

Otro de los retos ha sido integrar todo el desarrollo, en lo máximo posible, dentro de la propia aplicación de Dynamics AX, es el propio ERP el responsable de enviar la notificación push usando el servicio de MPNS, es el propio ERP el responsable de pasarle la información a la aplicación móvil, ésta al final, no se preocupa de saber cuál es la estructura de datos que usa Dynamics AX para gestionar las alertas. Esto mejora el mantenimiento de este desarrollo, migrar la aplicación a una versión superior o hacer modificaciones se hace más sencillo, dado que no hay que trabajar con múltiples y distintas herramientas. No sólo eso, si no que se puede aprovechar el desarrollo para otros dispositivos dado que el código no está orientado a funcionar específicamente con una aplicación para móvil, si no que se ha desarrollado, en la parte de Dynamics AX un sistema para la consulta de alertas (a excepción de la parte que hace referencia al envío de notificaciones push).

Durante el desarrollo del proyecto hay temas que se podrían mejorar o que se podrían aprovechar de cara a un futuro para ampliar nuevas funcionalidades:

1. El uso Windows Azure para la autentificación contra un servicio de AIF montado en Dynamics AX tal y como se describe en el whitepaper “Developing secure mobile apps for Microsoft Dynamics AX 2012” (16). No se ha podido usar debido los requerimientos, como el uso de Active Directory Federation Services (el sistema se ha montado sobre la infraestructura de la empresa IFR y no se ha dado permiso para la instalación de este software).
2. Estudiar el uso de claim users para la autentificación, Dynamics AX permite crear un nuevo tipo de usuario que no tiene que estar dado de alta en el dominio donde se trabaja, pero toda la documentación disponible hace referencia al uso de este tipo de usuarios para la aplicación web “Enterprise portal”.
3. Ampliar la funcionalidad para poder usar iOS y Android.
4. Aprovechar el trabajo realizado para realizar una aplicación para Tablet y Windows 8

A nivel personal ha sido muy satisfactorio el poder haber programado por primera vez una aplicación para Smartphone con cierta complejidad, haciéndolo intervenir con un ERP como es Dynamics AX2012 pudiendo sacarle partido al AIF.

También ha sido muy importante para mí, poder realizar el proyecto final de carrera esperando ver como los clientes de IFR usan la APP para integrar los smartphones de sus empleados con el ERP.

# 10. Bibliografía


1. http://www.erpsoftwareblog.com/2014/03/how-many-companies-use-microsoft-dynamics-erp/. [En línea]

2. http://es.wikipedia.org/wiki/Hardcode. [En línea]

3. http://msdn.microsoft.com/en-us/library/jj677285.aspx#BKMKOverviewOfPartit. [En línea]

4. http://msdn.microsoft.com/en-us/library/aa632254.aspx. [En línea]

5. http://msdn.microsoft.com/en-us/library/aa659581(v=ax.10).aspx. [En línea]

6. http://msdn.microsoft.com/en-us/library/aa570087(v=ax.10).aspx. [En línea]

7. http://es.wikipedia.org/wiki/Simple_Object_Access_Protocol. [En línea]

8. http://msdn.microsoft.com/en-us/library/windowsphone/develop/gg521150(v=vs.105).aspx. [En línea]

9. http://technet.microsoft.com/es-es/library/cc753127(v=ws.10).aspx. [En línea]

10. http://blogs.msdn.com/b/davidhardin/archive/2010/12/30/wp7-and-self-signed-ssl-certificates.aspx. [En línea]

11. http://msdn.microsoft.com/en-us/library/windowsphone/develop/gg521147. [En línea]

12. (http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff941100(v=vs.105).aspx). [En línea]

13. http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff431813(v=vs.105).aspx. [En línea]

14. http://msdn.microsoft.com/en-us/library/bdts8hk0(v=vs.95).aspx. [En línea]

15. http://blogs.windows.com/buildingapps/2012/10/01/how-to-create-an-infinite-scrollable-list-with-longlistselector/. [En línea]

16. **Jagruti Pandya, Rob Drollinger.** *sitio web de Microsoft.* [En línea] Mayo de 2013. http://www.microsoft.com/en-us/download/confirmation.aspx?id=38413.