Uso de los servicios web de Odoo a través de xmlrpc con C#:

Se ha incluido en el repositorio del proyecto una carpeta de nombre odoo_interface, esta carpeta contiene un proyecto hecho con Mono C# en Ubuntu, pero en principio compatible con C# .net, este proyecto consta de dos subproyectos, en concreto OpenERPClient, que es el proyecto que contiene la librería de conexión a Odoo a través de xmlrpc y odoo_interface, que contiene un ejemplo de uso de la librería.

Métodos disponibles en OpenERPClient:

- **OpenERPConnect(string url, string dbname, string login, string pwd):** Constructor de la clase de conexión, debe recibir como argumentos, la url de conexión a Odoo (ej: http://localhost:8069), el nombre de la base de datos a la que nos queremos conectar, el login del usuario con el que nos conectamos y su contraseña, en texto plano.
- **Login():** Método que se encarga de hacer login en Odoo, con los datos pasados al constructor de la clase. Una vez terminado el login, dispondremos en una propiedad de la instancia con nombre **UserId,** del identificador (int) del usuario de Odoo.
- **Create(string model, XmlRpcStruct fieldValues):** Método que nos permite crear un registro en Odoo, recibe como primer argumento el nombre del modelo en el que queremos hacer el "insert" y como segundo argumento, el conjunto de valores que queremos insertar, en un formato similar a [(clave, valor)] donde la clave se debe de corresponder con un campo del modelo indicado de Odoo y el valor, con un dato en el tipo del campo, se puede ver un ejemplo de creación en el proyecto de odoo_interface como de todo lo demás. Para añadir un elemento en un XmlRpcStruct se usa, el método Add, de la forma record.Add("clave", valor). Este método nos devolverá el identificador (long) del registro que se acaba de crear en Odoo.
  Los tipos de campos posibles en Odoo son:
  - **integer**: long o int de C#
  - **float**: float, double o decimal de C#
  - **char**: string de C#
  - **text**: string de C# pero permite multilinea, el tipo char no.
  - **boolean**: bool de c#
  - **date**: string de C# que representa a una fecha en formato inglés, %Y-%m-%d
  - **datetime***:* string de C# que representa a una fecha y hora en formato inglés y GMT+0, %Y-%m-%d %H:%M:%S
  - **binary**: Representa datos binarios, imágenes documentos etc... codificados en base64.
  - **many2one**: long o int de C# que se corresponde con un identificador de un registro de Odoo, ya que los campos many2one representan las foreign keys de la base de datos.
  - **one2many**: Es la contrapartida de los many2one, no tiene ninguna representación en la base de datos, pero sin en Odoo, con insertar el many2one asociado ya se rellena de forma automática, si se quisiera insertar el propio one2many, hay que hacerlo de una forma especial que explico en el siguiente tipo de campo.
  - **many2many**: Representan a las relaciones many2many de la base de datos, la inserción como en los one2many se hace de una forma especial, no suele ser muy común:
    - **[[6, 0, [ids]]]**: Borra las asociaciones actuales del campo y le asocia los identificadores contenidos en ids (int[])
    - **[[4, id]]**: Asocia el identificador contenido en id (int) al campo.
    - **[[3, id]]**: Desasocia el identificador contenido en id (int) del campo.
    - **[[2, id]]**: Desasocia y borra el identificador contenido en id (int) del campo y de la base de datos.
    - **[[0, 0, fields]]:** Sólo se utiliza con los one2many y nos permite crear los registros relacionados desde el campo one2many sin tener que crearlos individualmente y asociarlos por el many2one, fields sería de tipo **XmlRpcStruct** de la forma que describimos arriba.
    - **[[1, id, fields]]**: Sólo se utiliza en los one2many como el anterior y nos permite cambiar los valores de un registro del one2many, desde el propio campo, sin tener que hacerlo con un Write, que explicaremos más adelante, en el propio registro. De nuevo, fields vuelve a ser de tipo **XmlRpcStruct** e id (int) representa el identificador del registro que queremos modificar.
- **Search(string model, Object[] filters):** Método que nos permite consultar a través de filtros la base de datos de Odoo, como primer argumento, como antes, recibe el nombre del modelo en el que queremos hacer el "select", como segundo argumento recibe los diferentes filtros que queremos aplicar, a modo de "where" con un formato similar a [[campo, operador, valor],[...]] donde campo, se corresponde con el nombre de un campo del modelo de Odoo, operador con unos de los operadores de búsqueda disponibles en Odoo, que listaré al final y valor, con el valor por el que queremos buscar, este campo valor, con ciertos operadores, cogerá el carácter '%' como un comodín de búsqueda al estilo '*' de Sql. Este método, nos devolverá un array de longs, long[], con el listado de todos los identificadores (int) que cumplen el filtro pasado como argumento.
  Los operadores posibles de búsqueda son:

- =//**!=:** El valor del campo pasado es exactamente igual//no es igual, al valor pasado. El posible usarlo con cualquier tipo de campo. Ej: [[“name”, “=”, “Prueba”]] // [[“name”, “!=”, “Prueba”]]

- >//</<=//>=**:** Usado para comparar campos numéricos o de tipo fecha, tiene el significado lógico. Ej: [[“qty”, “>”, 0]] // [[“qty”, “<”, 65.5]] // [[“date”, “>=”, “2014-12-12 00:00:01”]] *(Las fechas, Odoo debe recibirlas en formato inglés, lleven horas o no y si llevan horas, estás, deberían estar en la zona horaria GMT+0, %Y-%m-%d o %Y-%m-%d %H:%M:%S)*

- **like:** Se usa para buscar en campos de tipo string, se corresponde con el like de Sql, pero añadiendo de forma automática el comodín de Odoo (%) ,al principio y al final del valor pasado. Ej: [[“name”, “like”, “ueb”]]

- **ilike:** Exactamente igual que like, pero con la diferencia de que no distingue minúsculas y mayúsculas. Ej: [[“name”, “ilike”, “prueb”]]

- =**like**: Similar a like pero nos permite pasarle el comodín (%) donde nos interese, y cuantas veces nos interese. Ej: [[“name”, “=like”, “Pr%eb%”]]

- **in//not in**: Se corresponde con los operadores del mismo nombre de Sql y nos permite buscar en campos one2many y many2many, o buscar en cualquier campo por un conjunto de valores. El argumento valor con este operador debe ser un array. Ej: [[“address_ids”, “in”, [2,4]]] // [[“state”, “not in”, [“draft”,”cancel”]]]

- **child_of:** En campos many2one (Fks) que relacionen con un modelo de Odoo que se puede representar en árbol, véanse cuentas contables, categorías de productos, empresas, ubicaciones de almacén..., nos permite filtrar por toda la jerarquía, por ejemplo [['location_id', 'child_of', [6]]], nos devolverá todos los registros que tengan como location_id 6, o cualquiera de sus hijos en el árbol.

– **Read(string model, long[] ids, string[] fields):** Método que nos permite obtener los campos que nos interesan de los registros pasados como argumento, como antes, model es el nombre del modelo de Odoo del que queremos obtener la información, ids una array de longs con cada uno de los ids de los que queremos obtener información y fields un array de strings con el listado de los campos que queremos que nos devuelva. Este método, nos devolverá un array de tipo XmlRpcStruct (XmlRpcStruct[]), con un formato similar a [[(“campo1”, valor),(“campo2”,valor),(“id”,1)],[(“campo1”, valor),(“campo2”,valor),(“id”,2)]], aparte de los campos pedidos, siempre no volverá a traer el id. Este XmlRpcStruct[], se pude recorrer con un foreach y luego, para cada elemento, acceder a el estilo Hashtable, record[“campo1”].
– **Unlink(string model, long[] ids):** Método que nos permite borrar uno o varios registros en Odoo, como en todos, como primer argumento recibe el nombre del modelo de Odoo, en el que queremos borrar los registros y como segundo argumento, un array de longs con los identificadores a borrar. Este método devuelve un bool, informando del éxito de la operación.
– **Write(string model, long[] ids, XmlRpcStruct fieldValues):** Este método se usa para actualizar registros en Odoo, a modo de update de Sql, recibe como siempre el nombre del modelo a actualizar, la lista de identificadores de los registros a actualizar y un XmlRpcStruct como el que se comentó en el método Create, pero, sólo con los campos que nos interesa actualizar en los registros indicados. Como el método Unlink, éste, nos devuelve un bool informando del éxito de la operación de actualización.
– **Execute(string model, string method, long[] ids):** Este método nos permite ejecutar cualquier método, de cualquier modelo de Odoo, siempre y cuando su firma, sólo esté obligada a recibir los argumentos de conexión (cr, uid) y el listado de identificadores que lo ejecutan (ids), esto engloba a la inmensa mayoría. Si se necesitaran más argumentos, este método debería sobrecargarse. Como siempre, como primer argumento recibe el modelo del que queremos ejecutar el método, como segundo, el nombre de la función a ejecutar y como tercer argumento, el array de identificadores (long), con los que queremos ejecutarlo. Nos devolverá un Object, que posteriormente habrá que hacerle un cast, al tipo correcto, si lo necesitamos utilizar, para esto, tendremos que valernos del conocimiento del método a la que estamos llamando.

Modelos de Odoo de interés, para la integración con Vstock:

– **product.product**: Este modelo representa a los productos en Odoo, que tanto pueden ser mercancías, consumibles o servicios. Los albaranes se componen de otro modelo que veremos más tarde, stock.move, que va a tener un campo many2one que relaciona con product.product, así, indica que producto se está moviendo. Se compone principalmente de los campos:

- id (integer): Identificador del registro en la base de datos.
- name (char): Nombre del producto
- description (text): descripción larga del producto
- type (char): Se representa técnicamente como char, pero es un combobox que nos permite indicar el tipo del producto, los valores que puede tener son: “consu” para Consumibles, que son productos que si se van a albaranar, pero de los que no se controla disponibilidad de stock, “service” son productos que representan servicios y nunca se van a albaranar y “product” que son productos que representan mercancías y en los que si se va a controlar disponibilidad de stock.
- list_price (float): Precio de venta base
- standard_price (float): Precio de coste

- volume (float): Volumen
- weight (float): Peso bruto
- weight_net (float): Peso neto
- ean13 (char): Código Ean13
- default_code (char): Código que representa al producto.
- qty_available (float): Stock del producto en almacén.
- virtual_available (float): Stock del producto en almacén, menos cantidad pendiente de salir, más cantidad pendiente de entrar.
- incoming_qty (float): Cantidad del producto pendiente de entrar.
- outgoing_qty (float): Cantidad del producto pendiente de salir.
- track_incoming (boolean): Marca que es obligatorio poner un número de serie/lote en las entradas de material.
- track_outgoing (boolean): Marca que es obligatorio poner un número de serie/lote en las salidas de material.
- track_all (boolean): Marca que es obligatorio poner un número de serie/lote para cualquier movimiento de mercancía.
- uom_id (many2one): FK que relaciona el producto con su unidad de medida, por lo general va a ser Unidad, por lo que no haría falta mapearlas en principio.

– **res.partner**: Este modelo representa a las empresas y direcciones en Odoo. Desde el modelo de albaranes que definiré más abajo, se accedería a este modelo para obtener los datos del cliente/proveedor. Se compone principalmente de los campos:
- id (integer): Identificador del registro en la base de datos.
- name (char): Nombre fiscal de la empresa
- parent_id (many2one): Si el registro fuera una dirección, a través de este campo accedemos a la empresa, a la que pertenece esa dirección.
- child_ids (one2many): Es la contrapartida del anterior, si el registro fuera una empresa a través de este campo, obtenemos todas sus direcciones.
- ref (char): Código que identifica a la empresa. No se suele utilizar.
- lang (char): Idioma en el que se le envían los informes a esa empresa, son strings del tipo: es_ES o en_US.
- vat (char): CIF/Nif del cliente/proveedor
- comment (text): Se utiliza para dejar comentarios asociados al cliente/proveedor
- customer (boolean): Marca la empresa como cliente.
- supplier (boolean): Marca la empresa como proveedor.
- street (char): Dirección
- street2 (char): Segunda linea de dirección
- zip(char): Código postal
- city (char): Ciudad
- state_id (many2one): FK hacia un modelo de nombre res.partner.state, que en su campo name contiene el nombre de la provincia.
- country_id (many2one): FK hacia un modelo de nombre res.country, que en su campo name contiene el nombre del país.
- contact_address (text): Campo función que nos devuelve la dirección completa, con todos los campos concatenados, con un formato similar a: Nombre de empresa
  Calle
  Calle2
  Ciudad C.P.
  País

– **stock.picking**: Modelo de Odoo que representa a los albaranes. Se compone principalmente de los campos:
- id (integer): Identificador del registro en la base de datos.
- name (char): Número del albarán
- origin (char): Número del pedido de venta o compra que creó el albarán
- backorder_id (many2one): Es una FK sobre este mismo modelo, que en el caso de una recepción parcial o entrega parcial, asocia el albarán procesado con el albarán con la cantidad pendiente de procesar.
- note (text): Campo texto donde se pueden escribir anotaciones del albarán.
- state (char): Se representa técnicamente con un char, pero es un combobox que va informando del estado del albarán, nunca se suele escribir de forma directa, lo suele escribir el sistema, según el albarán avanza en su flujo de trabajo, pero si es útil para hacer búsquedas. Los valores permitidos son:
  - “draft”: Estado borrador, indica que el albarán ha sido creado o está siendo creado y que todavía no se confirmó, no afecta al stock en este estado.
  - “cancel”: Estado cancelado, indica que el albarán ha siso cancelado, sigue en el sistema a modo de histórico, pero no es posible volver a abrirlo, no tiene ninguna repercusión en el stock.

- “waiting”: Estado esperando otra operación, indica que el albarán ha sido confirmado y se encuentra esperando otra operación, sea un abastecimiento, una compra, una producción, etc... mientras la operación que espera no se realice, no permite avanzar su flujo a excepción de que se fuerce. No tiene ninguna repercusión en el stock.
- “confirmed”: Estado esperando disponibilidad, el albarán ha sido confirmado pero no tiene stock disponible para procesarse, como antes, su flujo queda parado hasta que haya stock y se vuelva a intentar asignar o se fuerce. No tiene ninguna repercusión en el stock.
- “partially_available”: Estado parcialmente disponible, indica que se ha podido reservar parte de las mercancías, pero no sus totalidad, como el anterior, no se puede avanzar hasta que este toda y se trate de volver a asignar o se fuerce. La cantidad que se consigue reservar, figurará restada en salidas e incrementada en entradas, en el campo virtual_available de los correspondientes productos.
- “assigned”: Estado listo para transferir, indica que la mercancía está disponible en su totalidad o se forzó y ya está listo para ser finalizado. La cantidad total de cada movimiento, figurará restada en salidas e incrementada en entradas, en el campo virtual_available de los correspondientes productos.
- “done”: Estado realizado, marca como finalizado el albarán. La cantidad total de cada movimiento será efectiva en el stock y figurará ya en el campo de stock real, qty_available, de los correspondientes productos.

- min_date (datetime): Fecha estimada de salida/entrada mínima
- max_date (datetime): Fecha estimada de salida/entrada máxima.
- move_lines (one2many): Campo que relaciona el albarán con sus movimientos o lineas, contendrá el listado de registros de movimientos, stock.move, que se engloban en el albarán.
- partner_id (many2one): Campo que relaciona el albarán con la dirección de envío, en el caso de albaranes de salida y con la dirección origen, en el caso de albaranes de entrada.
- picking_type_id (many2one): Campo que relaciona con una tabla maestra de tipos de albaranes, es el campo que nos va a indicar de que tipo es el albarán, es interesante para las búsquedas se suele utilizar en los filtros de búsqueda como [[“picking_type_id.code”, '=', “outgoing”]], por ejemplo, en este caso para filtrar los albaranes de salida, los valores posibles del campo code son:
  - “incoming”: Indica que son albaranes de entrada
  - “outgoing”: Indica que son albaranes de salida
  - “internal”: Indica que son albaranes internos, de movimientos dentro del almacén.
- picking_type_code (char): Atajo para buscar por el código del tipo de albarán, el filtro [[“picking_type_code”, '=', “outgoing”]] sería equivalente al que vimos en el campo anterior.
- shipping_identifier (char): Código identificador del envío

– **stock.move**: Modelo de Odoo que representa a las lineas de un albarán, es el modelo que hace variar el stock de los productos, representa el movimiento de una mercancía de una ubicación a otra, física (otra ubicación interna, otro almacén) o virtual (ubicaciones donde se crea o destruye material, véase Clientes, Proveedores, Producción...). Se compone principalmente de los campos:

- id (integer): Identificador del registro en la base de datos.
- date_expected (datetime): Fecha estimada de recepción/envío, la usa el albarán para calcular los campos de min_date y max_date.
- product_id (many2one): FK que relaciona con el modelo product.product, visto con anterioridad, indica que producto se está moviendo.
- product_uom_qty (float): Campo que indica la cantidad del producto que se mueve.
- product_uom (many2one): Fk que relaciona con la unidad de medida en la que se mueve la mercancía, por lo general, va a ser la misma que la definida en el campo uom_id del producto.
- location_id (many2one): Ubicación de origen del movimiento, supongo que para vstock no será un dato relevante.
- location_dest_id (many2one): Ubicación de destino del movimiento, de nuevo supongo que para vstock no será relevante, en principio no será necesario mapearlas.
- picking_id (many2one): FK que relaciona el movimiento con el albarán que lo contiene, es la representación en la base de datos, del campo contrario que vimos en el modelo stock.picking, de nombre move_lines.
- note (text): campo texto que nos permite añadir anotaciones relacionadas con el movimiento.
- state (char): Tiene los mismos estados que stock.picking a excepción de “partially_available”, de nuevo, no se suele nunca escribir manualmente, hay funciones que se encargan de hacerlo respetando su flujo de trabajo.
- partially_available (boolean): Substituye al estado “partially_available”, que comentamos antes que no tenía, indicán si hay una parte reservada o no.
- scrapped (boolean): En Odoo, es posible marcar parte o todo un movimiento como 'Desecho', esto lo que va a hacer es duplicar el movimiento desde el que se llamó a esa función y en el duplicado, poner la cantidad desechada y esta marca a True, además de cambiar la ubicación de destino del movimiento, por la ubicación de desechos, esta marca indica que es un movimiento de ese tipo.

Métodos de los modelos de Odoo, de interés para la integración con Vstock:

- **stock.picking:**
  - action_confirm: Confirma el albarán, mueve el albarán y sus movimientos de estado “draft” a “confirmed”
  - action_assign: Comprueba la disponibilidad de stock para procesar el albarán. Moverá según la condición que se dé, el albarán y sus movimientos a estado “assigned”, “waiting” o “partially_waiting”.
  - force_assign: Fuerza la disponibilidad del albarán, haya stock o no, lo avanza a estado “assigned”, al igual que sus movimientos, dejándolos listos para procesar, eso si, pudo dejar el stock en negativo sino había suficiente para el/los movimientos procesados.
  - action_cancel: cancela el albarán y los movimientos asociados.
  - action_done: Finaliza el albarán y los movimientos asociados, haciendo efectivo el/los cambios de stock.
  - Para hacer un procesado parcial, deberemos usar un asistente que también se define como un modelo de Odoo, con nombre “stock.transfer_details”, y seguir estos pasos:
    1. Crearlo simplemente rellenado el campo picking_id, con el identificador del albarán que queremos procesar.
    2. Crear un “stock.transfer_details_items” por cada movimiento que queramos procesar, entero o parcialmente del albarán, rellenando, los campos: transfer_id con el identificador del “stock.transfer_details” creado en el punto anterior, product_id con el identificador el producto del movimiento, product_uom_id con el identificador de la unidad de medida del movimiento, quantity con la cantidad que queremos procesar del movimiento, sourceloc_id con el identificador de la ubicación de origen del movimiento, destinationloc_id con el identificador de la ubicación de destino del movimiento y el campo date, con la fecha y hora de procesado en formato inglés.
    3. Ejecutar la función do_detailed_transfer del modelo “stock.transfer_details”, pasándole como ids el id creado en el primer paso.
    4. Esto nos procesará el albarán actual con la condiciones impuestas y nos creará una copia con la mercancía que falte por procesarse, relacionándolo con el albarán actual, a través del campo backorder_id.

Notas

***** Si van a disponer de una copia de Odoo para hacer pruebas, es de gran ayuda activar el modo desarrollador, una vez hecho login, nos vamos arriba a la derecha donde aparece escrito nuestro nombre de usuario, clickamos encima y pulsamos en “Acerca de Odoo”, en el popup que se abre, pulsamos en “Activar modo desarrollador”, la web se refrescará y debajo del menú superior, nos aparecerá un combobox con diferente utilidades, pero lo más interesante, es que si entramos en la vista formulario de algún registro, por ejemplo un albarán, si nos situamos con el ratón encima de cualquier campo, nos mostrará un tooltip, con el nombre del campo, su tipo y dependiendo del tipo, con que objeto relaciona o que valores acepta.