



**Implantación de OpenERP y  
programación de un conector  
con básculas MAPAL**

Jose Ramón Terol Borrás

13 de Septiembre de 2010





---

Escuela Técnica Superior de Ingeniería Informática

I.T. Informática De Sistemas

Implantación de OpenERP y  
programación de un conector  
con básculas MAPAL

Departamento de Sistemas Informáticos y Computación

Director del Proyecto: Jon Ander Gómez

Autor del proyecto: Jose Ramón Terol Borrás

Valencia, 13 de Septiembre de 2010





## Licencia

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al final del mismo.

Copyright (c) 2010 Juan Jesús Sanz Ortolá / Jose Ramón Terol Borrás

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

## Notación

En el presente documento se sigue la siguiente notación: Los términos extranjeros, inglés por ejemplo, aparecerán *en cursiva*. Los nombres de los ficheros aparecerán *en cursiva con fuente monoespaciada*. Los extractos de código fuente, ya sea XML o python aparecerán en un marco como este con fuente monoespaciada:

```
código fuente
```

Por último, cuando se indique un comando a ejecutar en línea de comandos, se usará una fuente monoespaciada y el comando vendrá precedido del símbolo del dólar:

```
$ sudo aptitude install programa
```



## Índice de contenido

Licencia .....	5
Notación.....	6
Capítulo 1.....	9
Introducción.....	9
1.- Antecedentes.....	9
2.- Alcance y objetivo.....	12
Capítulo 2.....	14
Planteamiento del proyecto.....	14
1.- Estudio básico de las opciones disponibles.....	14
2.- Funcionamiento de la opción elegida, OpenERP.....	25
3.- Arquitectura de OpenERP.....	26
Capítulo 3.....	29
Descripción de la solución adoptada.....	29
1.- Esquema general de la solución.....	29
2.- El entorno de producción, instalación.....	31
3.- El entorno de desarrollo, instalación.....	51
4.- La programación del módulo.....	55
1.- Módulo OpenERP.....	55
2.- La vista.....	83
3.- El controlador.....	102
Conclusiones.....	106
Bibliografía .....	108
GNU Free Documentation License.....	109





## Capítulo 1

### Introducción

#### 1.- *Antecedentes*

La intención de este Proyecto de Final de Carrera es demostrar y reforzar una parte de los conocimientos adquiridos durante la carrera de Ingeniería Técnica en Informática de Sistemas. Nuestra pretensión no es aportar la solución definitiva al problema presentado, sino aportar una solución válida y con la planificación necesaria para ser ampliada y mejorada en un futuro si es requerido por cualquiera de nuestros clientes o potenciales clientes.

También es necesario ceñirse a lo demandado por el cliente, ya que no disponemos de tiempo ni fondos suficientes para realizar un gran proyecto válido para absolutamente todas las tipologías de clientes. No obstante es necesario dejar las puertas abiertas a futuras modificaciones sin que ello suponga un excesivo gasto de tiempo en rediseñar y adaptar los fundamentos de la aplicación.

Es por ello que este trabajo estudiará las soluciones más importantes existentes en el mercado, así como sus capacidades para ser modificadas, mejoradas y adaptadas a los requisitos demandados por el cliente.

Gracias al avance de las tecnologías de la información ya no es competencia exclusiva de las grandes organizaciones el tener la información del funcionamiento de un negocio en formato electrónico que permita al empresario conocer al momento cuál es la marcha



actual de su negocio. Esto ha impulsado al mercado a crear gran número de soluciones comerciales a tal requerimiento, la mayoría de ellas cerradas a modificaciones por terceros y con un coste en licencias significativo.

Además cada solución creada se limita a un tipo de cliente concreto en el caso de soluciones sectoriales específicas de relativo bajo coste. En el caso de soluciones de aplicación más amplia gozamos de la ventaja de una adaptación muy ajustada a las necesidades de la empresa, con la desventaja de un coste en algunos casos demasiado elevado para la mayoría de PYME que conforman el tejido empresarial español.

En lo referente a soluciones sectoriales, de éstas se encargan algunas pequeñas empresas dedicadas a la programación que han reunido a grupos de clientes con las mismas necesidades y han desarrollado una solución a medida. Esta solución es modificada a demanda del grupo de clientes y no es aplicable ni posee una interfaz parecida a las aplicaciones creadas por otras empresas incluso ofreciendo a la misma tipología de clientes una solución al mismo problema. Todo esto crea una dependencia total hacia una empresa desarrolladora que, eventualmente, podría desaparecer dejando a todos sus clientes con una aplicación que en unos años será inutilizable debido a la falta de adaptación a los cambios en normativas, leyes y mercados.

Pensamos que la utilización de software libre es la mejor solución para esta complicada situación, ya que permite el abaratamiento de costes en desarrollo, así como un programa de mayor calidad al tener



la oportunidad de mejorar código en lugar de reescribir una solución ya existente. Dispondríamos de una interfaz de usuario unificada, con el consiguiente ahorro de costes en el área de formación de empleados y adaptación.

Todo esto, más allá de quitar trabajo a las empresas del sector de la programación, permitirá que más usuarios puedan acceder a programas de calidad, consiguiendo un importante ahorro en la adaptación básica de estos programas, como puede ser contabilidad, traducciones, mejoras en la interfaz, mejoras en rendimiento, etc. Aumentando así el beneficio, ya que estas adaptaciones se pueden hacer en común (menos trabajo) y aplicarlas en masa a todos sus clientes o a un grupo de ellos según sea necesario.

Ya son bastantes las empresas que trabajan en proyectos comunes gracias a la extensibilidad de la aplicación por medio de módulos que encajan perfectamente en el sistema mediante la definición de un estándar (incluso los módulos base son módulos al fin y al cabo).

Además, la competencia en este ámbito de la programación (como en la mayoría), está basada en multinacionales enormes como SAGE, Microsoft y SAP. No existe forma de competir con una facturación de millones de euros y la capacidad de dedicar recursos a una adaptación o mejora que tienen estas empresas. La única solución para las PYME dedicadas al sector es unirse en proyectos de software libre que, gracias a las contribuciones de todos los socios, están a la altura e incluso superan a los grandes programas comerciales tanto en prestaciones como en recursos dedicados a los mismos.



## 2.- Alcance y objetivo

Este proyecto consistirá en la adaptación de un ERP de propósito general a las necesidades básicas de una pequeña empresa, en realidad a una parte de ella. Nos centraremos en la integración de la facturación, gestión de almacén y gestión de ventas mediante la extracción de datos de ventas de las básculas adquiridas por el cliente.

El objetivo es aprovechar los módulos ya existentes como son la gestión de almacén, la contabilidad y la facturación, y crear un módulo para la tarea de extracción de datos de las básculas. Las básculas son de la empresa MAPAL, que no dispone de ninguna aplicación de extracción de datos, simplemente tiene una interfaz web de gestión básica que no provee capacidad de actualización ni modificación después de ensamblada la máquina.

Después de hablar con el fabricante se dejó en el aire la posibilidad de crear un estándar de interacción báscula-programa mediante la tecnología XML, sin embargo teníamos que trabajar con lo ya disponible, así que en este proyecto programaremos un conector que descargue la página web de la báscula y extraiga los datos necesarios para la creación de documentos internos del programa de gestión.

También hay que dejar el módulo abierto a la posibilidad de modificar directamente los datos de la báscula sin necesidad de acceder a la interfaz web, así como modificaciones en masa, gestión avanzada de tiendas, etc. Sin embargo no es esto lo que en este momento necesita el cliente.

En resumen, los objetivos que debemos alcanzar una vez finalizado el presente proyecto son los siguientes:



- Administración de la facturación del cliente, tanto facturación manual a clientes empresas como facturación automática diaria por vendedor y tienda a raíz de los datos extraídos de las básculas interconectadas.
- Gestión de inventario, mediante la creación de albaranes de salida de mercancía por el total de unidades o kilos de género vendidos en cada tienda, de forma que las existencias computadas por el programa sigan lo más fielmente posible las existencias reales dentro de la tienda.
- Gestión básica de clientes y proveedores. Más adelante explicaremos la decisión de por qué tratamos como un cliente más a cada uno de los vendedores de la tienda.
- Extracción de vendedores de las básculas MAPAL. Con el fin de separar la facturación realizada por cada uno de los vendedores primero tendremos que importar al ERP los vendedores que las básculas tienen datos de alta.
- Extracción de los artículos de las básculas MAPAL. Esto es necesario para poder llevar cuenta de las existencias de cada uno de estos artículos.
- Extracción de datos de ventas de las básculas MAPAL. El objetivo final que permitirá llevar cuenta de todas las ventas realizadas en la tienda, así como las cantidades de cada artículo.



## Capítulo 2

### Planteamiento del proyecto

#### 1.- *Estudio básico de las opciones disponibles*

Explicaremos, en primer lugar, opciones disponibles de la competencia que no podemos utilizar ya que son privadas, pero que nos darán una guía de qué es lo que en un principio está demandando el mercado. Para ello empezaremos hablando del líder en el mercado español, Sage.



Sage es una empresa especializada en servicios para empresas en cuanto a programas de gestión, ERP, CRM, etc. Tiene multitud de productos centrados en resolver una parte del problema cada uno. Así, en el ámbito de las PYME, tenemos FacturaPlus, ContaPlus, NominaPlus, TPVPlus, PYMEPlus, Autónomos y otros. El enfoque es entregar un paquete al cliente que resuelva todas las necesidades de su sector (frecuentemente muchas más de las que tiene en realidad el cliente).

El principal problema que presenta este enfoque es la integración, hay multitud de productos que hacen el mismo trabajo, por ejemplo las herramientas ERP y CRM engloban servicios como contabilidad,



## 1.- Estudio básico de las opciones disponibles

---

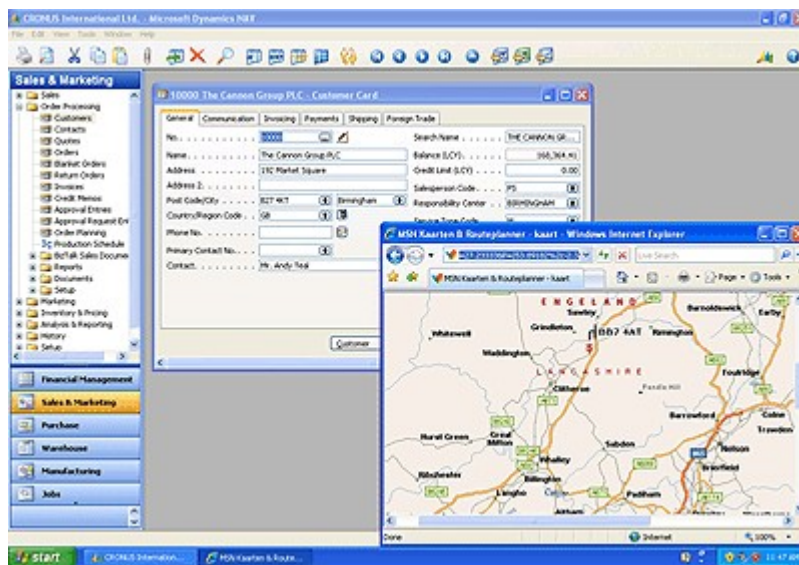
facturación, gestión de nóminas, gestión de clientes y otros tantos que además tienen sus aplicaciones separadas. Además no es posible modificar su código, hacer ampliaciones ni adaptarlo a las necesidades de un cliente particular si la empresa matriz no lo ve conveniente.



Lo que nosotros buscamos es una aplicación unificada que responda a demanda tanto a estas necesidades por separado como en su conjunto si se desea. Es por ello que nos decantaremos por una solución ERP.

Microsoft Dynamics NAV es el ERP que comercializa Microsoft. Su segmento de mercado son las empresas que puedan pagar alrededor de 2000€ por usuario sólo en licencias, a esto habría que añadir el coste de mantenimiento, adaptación al usuario, etc. Para poder adentrarnos en la posible utilización de este ERP es necesario certificarse como partner mediante cursos que otorgan diferentes niveles de conocimientos, así como pagar una cuota con el consiguiente coste para la empresa que desee dedicarse a implantarlo.





Debido a nuestro bajo potencial económico debimos desechar esta opción ya que no es nuestro deseo depender al 100% de una tercera empresa para el desarrollo de nuestra normal actividad.

Así pues decidimos estudiar las opciones comerciales que existen en el área del software libre, que permite a cualquiera aprender los entresijos de las aplicaciones, contribuir libremente, modificarlas para uso personal, vender las modificaciones y, además, disponer del código fuente de la aplicación eliminando la dependencia de la empresa creadora.





## 1.- Estudio básico de las opciones disponibles

En primer lugar comentaremos el ERP de Infosial llamado anteriormente FacturaLux y cuyo nombre actual es AbanQ. Este programa está desarrollado por una empresa de Albacete y tiene a sus espaldas numerosos casos de éxito. La particularidad del programa es que todos sus módulos se guardan en una base de datos MySQL y un cliente genérico y multiplataforma se conecta a ella para utilizar tanto los datos maestros como los diferentes módulos del programa.

The screenshot displays the AbanQ financial software interface, titled 'Area Financiera:Principal'. It features a menu bar with options like 'Financiera', 'Ejercicio', 'P.G.C', 'Ventana', and 'Ver Módulos'. The main workspace is divided into several panels, each showing a different accounting module with search and data table options.

- Cuentas especiales ( EJERCICIO 1 )**: Search by 'Código'. Table with columns 'Código' and 'Descripción'.

Código	Descripción
1	CAJA Cuentas de caja
2	CAMNEG Cuentas de diferencias negativas de cambio
3	CAMPOS Cuentas de diferencias positivas de cambio
4	CLIENT Cuentas de clientes
5	COMPRA Cuentas de compras
6	DIVPOS Cuentas por diferencias positivas en divisa entr
7	EURNEG Cuentas por diferencias negativas de conversi
8	EURPOS Cuentas por diferencias positivas de conversi
9	IRPF Cuentas de retenciones IRPF
10	IRPFPR Cuentas de retenciones para proveedores IRPF
11	IVAACR Cuentas acreedoras de IVA en la regularización
12	IVADEU Cuentas deudoras de IVA en la regularización
13	IIVAFPP Cuentas de IVA repercutido
- Conceptos ( EJERCICIO 1 )**: Search by 'ID'. Table with columns 'ID' and 'Descripción'.
- Códigos de Balances ( EJERCICIO 1 )**: Search by 'Código'. Table with columns 'Código', 'Naturaleza', 'Código nivel 1', and 'Descr'.

Código	Naturaleza	Código nivel 1	Descr
1	A	ACTIVO	
2	A-A	ACTIVO	A) AC
3	A-B	ACTIVO	B) INM
4	A-B-I	ACTIVO	B) INM
5	A-B-II	ACTIVO	B) INM
6	A-B-II-1	ACTIVO	B) INM
7	A-B-II-2	ACTIVO	B) INM
8	A-B-II-3	ACTIVO	B) INM
9	A-B-II-4	ACTIVO	B) INM
10	A-B-II-5	ACTIVO	B) INM
11	A-B-II-6	ACTIVO	B) INM
12	A-B-II-7	ACTIVO	B) INM
13	A-R-I-A	A/R	R) INM
- Epígrafes ( EJERCICIO 1 )**: Search by 'Código'. Table with columns 'Código' and 'Descripción'.

Código	Descripción
1	1. FINANCIACION BASICA
2	10. CAPITAL
3	11. RESERVAS
4	12. RESULTADOS PENDIENTES DE APLICACION
5	13. INGRESOS A DISTRIBUIR EN VARIOS EJERC
6	14. PROVISIONES PARA RIESGOS Y GASTOS
7	15. EMPRESTITOS Y OTRAS EMISIONES ANALOG
8	16. DEUDAS LARGO PLAZO CON EMPRESAS DEL
9	17. DEUDAS A LARGO PLAZO POR PRESTAMOS
10	18. FIANZAS Y DEPOSITOS RECIBIDOS A LARGO
11	19. SITUACIONES TRANSITORIAS DE FINANCIAC
12	2. INMOVILIZADO
13	20. GASTOS DE ESTABLECIMIENTO
- Cuentas ( EJERCICIO 1 )**: Search by 'Código'. Table with columns 'Código', 'Ejercicio', 'Epígrafe', and 'Descr'.

Código	Ejercicio	Epígrafe	Descr
1	100	0001	10. 100. C
2	1000	0001	10. 1000.
3	1001	0001	10. 1001.
4	1002	0001	10. 1002.
5	1003	0001	10. 1003.
6	101	0001	10. 101. F
7	102	0001	10. 102. C
8	110	0001	11. 110. P
9	111	0001	11. 111. R
10	112	0001	11. 112. R
11	113	0001	11. 113. R
12	114	0001	11. 114. R
13	115	0001	11. 115. R
- Subcuentas ( EJERCICIO 1 )**: Search by 'Código'. Table with columns 'Código' and 'Descripción'.

Código	Descripción
1	1000000000 100. Capital social
2	1000000001 1000. Capital ordinario
3	1001000000 1001. Capital privilegiado
4	1002000000 1002. Capital sin derecho a voto
5	1003000000 1003. Capital con derechos restringidos
6	1010000000 101. Fondo social
7	1020000000 102. Capital
8	1100000000 110. Prima de emision de acciones
9	1110000000 111. Reservas de revalorización
10	1120000000 112. Reserva legal
11	1130000000 113. Reservas especiales
12	1140000000 114. Reservas para acciones de la sociedad dominante
13	1150000000 115. Reservas para acciones minoritarias

AbanQ está programado en QT 3.3 y utiliza editores integrados para prácticamente cualquier actividad de modificación o personalización. Su mayor problema es que la tecnología QT 3.3 es de alrededor de 2003, que está muy lejos del QT 4.6.3 (Junio de 2010).



## 1.- Estudio básico de las opciones disponibles

---

Esto limita el soporte (hay cosas arregladas en versiones posteriores de QT y la solución es actualizar) y da una apariencia un tanto primitiva al programa en general. También se convierte en una tarea tediosa algo tan simple como modificar un informe, asunto que en las últimas versiones de la tecnología utilizada es mucho más básico, funcional y rápido.

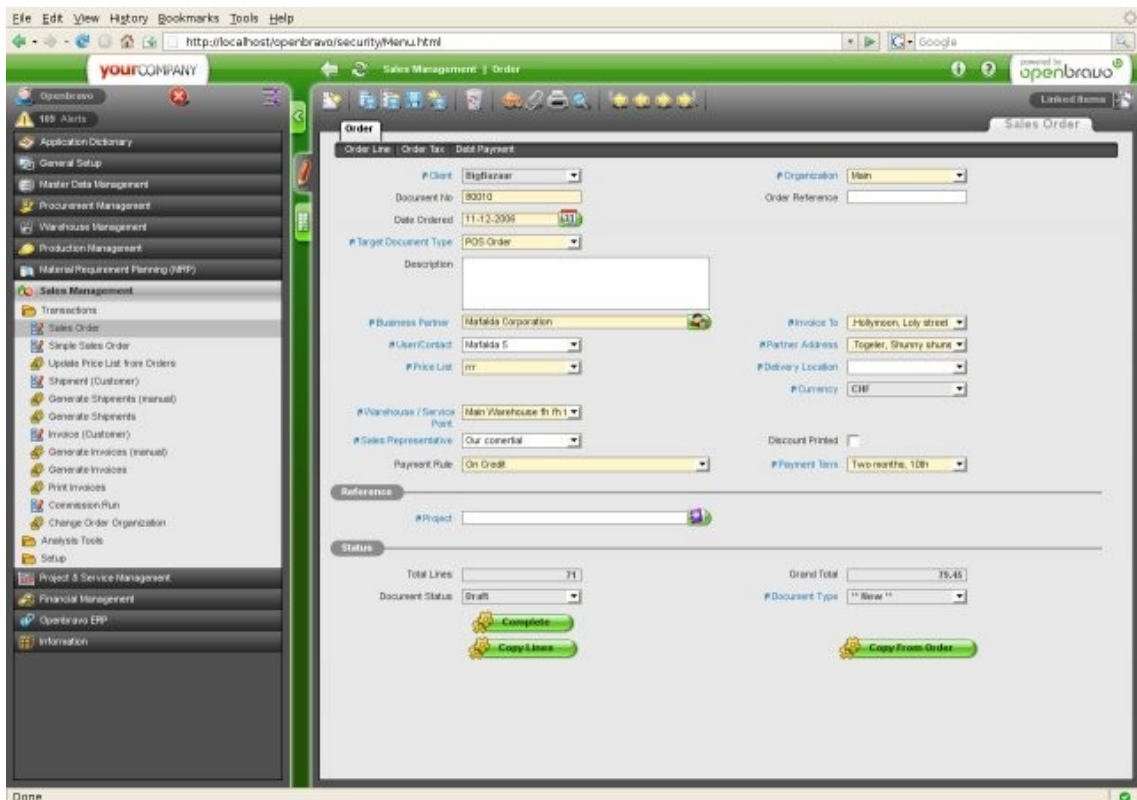
La falta de usuarios y cierta falta de atención por parte de Infosial hacia la comunidad ha propiciado la decadencia de este programa que, sin embargo, es una gran idea y totalmente funcional. No obstante nuestra intención es utilizar los beneficios de las últimas tecnologías y es por este motivo que este programa no será utilizado en la implantación final.

Openbravo fue otra opción a estudiar, de hecho, esta fue la opción que más nos convenció junto con OpenERP y la que pensábamos utilizar en la implantación final, hasta que llegó la hora de desarrollar.



Openbravo es un proyecto iniciado por la empresa Openbravo SLU y que ha conseguido grandes fondos para su desarrollo. Las tecnologías utilizadas en Openbravo son: Java, XML, HTML, CSS, AJAX, POSTGRESQL/ORACLE, SQLC... Además de desarrollar tecnologías propias con la ayuda de éstas.

Openbravo dispone de absolutamente todo lo que necesitamos tanto para este proyecto en particular como para proyectos futuros. Es multiusuario, con permisos completamente personalizables para cada uno de ellos, utiliza la tecnología cliente-servidor, la interfaz es web, sus módulos por defecto incluyen contabilidad, gestión financiera, contabilidad analítica, gestión multialmacén, recursos humanos,



facturación, planificación de compras y aprovisionamientos, módulos para TPV en un programa separado aunque integrado. Contiene herramientas de auditoría y amplios informes. Podemos destacar su escalabilidad, modularidad y que es altamente personalizable.

Detrás de todo esto hay una empresa altamente implicada con una comunidad creciente que había apoyado el proyecto casi desde la publicación del código de la versión libre en el 2006. Sin embargo en



## 1.- Estudio básico de las opciones disponibles

---

este aspecto y durante las fechas en las que empezamos a aprender y a involucrarnos con la comunidad nos llegaron ciertas críticas hacia la nueva dirección de la empresa con respecto a la comunidad de desarrolladores independientes. Éstas se centraban en los intentos de la empresa por tener todo controlado, incluso los módulos publicados por terceros dificultando su instalación si no se utilizaba su red de distribución, así como incluyendo todas las aportaciones de la comunidad a su rama principal devolviendo más bien poco a la misma. La lectura del artículo de despedida de la comunidad de desarrolladores de Openbravo de uno de los principales contribuyentes al código, persona muy influyente, nos hizo reflexionar en si apoyar este proyecto.

Además de todo esto, la curva de aprendizaje en cuanto al desarrollo es, según nuestra experiencia, extremadamente pronunciada. La aplicación se compone del diccionario de aplicación, que guarda la información de cada uno de los componentes del sistema y su comportamiento; el WAD (Wizard for Application Development) que genera código ejecutable a partir del diccionario mencionado anteriormente; el Framework MVC (Modelo Vista Controlador) de Openbravo, compuesto por multitud de aplicaciones cuya misión es dar forma a los datos para que puedan ser vistos desde un navegador web. Una vez finalizado este camino, los datos pasan a través del ejecutable de Openbravo que se encarga de servir los datos a los usuarios.

El enfoque es bueno, sin embargo a estas alturas de desarrollo de la aplicación algunas tareas aún son tediosas, repetitivas y complejas. Esto nos empujó a intentar encontrar alguna aplicación madura en la que el desarrollo fuese más simple y rápido, a la par de limpio.



# Compiere®

Compiere es el precursor de Openbravo. Este ERP fue descartado desde un principio por dos motivos principalmente: este proyecto ya no tiene casi actividad comparado con los otros y tenemos Openbravo que tomando su base tiene una tecnología mucho más actual.



Por último tenemos OpenERP, que es el que finalmente hemos escogido para realizar nuestra implementación. OpenERP utiliza Python en el lado del servidor, junto con Postgresql. En cuanto a la interfaz de usuario existe un cliente GTK oficial (también hay otros clientes no oficiales) y un servidor web que permite acceder desde cualquier navegador. Para la comunicación cliente-servidor OpenERP utiliza la tecnología XML-RPC. OpenERP es un ERP muy activo, con una gran comunidad alrededor suyo y con múltiples empresas en todo el mundo que le dan soporte. En España tenemos unas cuantas empresas que trabajan en la localización para España y en el Plan General Contable, traducciones y otras tareas.



## 1.- Estudio básico de las opciones disponibles

---

La metodología de desarrollo en OpenERP utiliza XML y Python exclusivamente en el desarrollo de cualquier módulo. No es necesario acceder a la base de datos directamente (se hace todo a través de funciones), la creación de tablas, relaciones, interfaces visuales y todo lo necesario para poder desarrollar un módulo funcional se realiza a través de un mismo lenguaje de programación. En cuanto a los informes, internamente el programa utiliza RML, que no es más que un archivo XML de ReportLabs (un programa escrito también en Python), aunque gracias a las contribuciones de la comunidad se han desarrollado más opciones. Entre ellas están la edición de informes a través de OpenOffice, Jasper Reports y en HTML, CSS y Javascript. Además existe un *plugin* para OpenOffice que no está liberado (es de pago), y que permite informes más avanzados.

En resumen, con sólo Python y XML podemos realizar todas las tareas para las que, en Openbravo, había que aprender Java, XML, HTML, CSS, Ajax, WAD, Diccionario MDD, MVC-FF, Javascript, SQL/PL-SQL.

El servidor OpenERP utiliza XML-RPC para todas las comunicaciones con las aplicaciones cliente. Esto permite utilizar los métodos de OpenERP desde cualquier otra interfaz que soporte el protocolo (por ejemplo una página web). Tiny ha desarrollado dos clientes para OpenERP, un cliente GTK y un cliente-servidor web. Terceros han desarrollado otros clientes (existe uno en QT), sin embargo su presencia no es importante en la actualidad.



### **2.- Funcionamiento de la opción elegida, OpenERP**

OpenERP está escrito totalmente en Python, un lenguaje de programación de alto nivel que no necesita ser compilado para ejecutarse ya que es interpretado. No obstante el sistema compila "al vuelo" los "scripts" de forma que el rendimiento del programa mejora notablemente. La extensión de los archivos compilados es .pyc, mientras que la extensión de las fuentes interpretables es .py.

La comunidad internacional que da soporte a Python se preocupa por mejorar el lenguaje día a día, así como de dar soporte a la mayoría de problemas comunes a los que se puede enfrentar un programador. De esta forma podemos encontrar bibliotecas de Python para casi cualquier tarea que deseemos realizar. Su uso es simple y efectivo, teniendo un grupo de herramientas que llegan a instalar dependencias y bibliotecas como un gestor de paquetes cualquiera del tipo *apt*.

El gran alcance de este lenguaje de programación llega hasta la web, llegando a ser posible substituir con este lenguaje el omnipresente PHP de la programación web. Además de disponer de varios frameworks (como Symphony en PHP), siendo el más famoso de todos ellos Django. Gracias a esto es posible disfrutar para cualquier tipo de programa (desde una aplicación web hasta un programa de escritorio) de unas capacidades casi infinitas. Todo esto, además, con una sintaxis muy legible y con muy pocas líneas de código en comparación con la mayoría de otros lenguajes.

El servidor de OpenERP está construido a base de módulos de Python. En la carpeta principal de OpenERP encontramos el módulo base que es el que arranca el programa, y que se encarga de cargar el





## 2.- Funcionamiento de la opción elegida, OpenERP

---

resto de módulos y de iniciar el servidor. A su vez, dentro de esta carpeta encontraremos el directorio "addons" (aunque esto es completamente configurable) en el que se encuentra el resto de módulos funcionales del programa con una carpeta para cada uno. De todos estos módulos, el más importante es "osv", compuesto de tres archivos Python de los que heredan todos los demás objetos. Estos archivos proveen los recursos necesarios para seguir la arquitectura MVC (Modelo, vista, controlador) que proporciona separación entre las diferentes partes de la programación de los módulos.

Así tenemos, basados en el modelo MVC que, la base de datos es el modelo, los objetos de OpenERP (clases u objetos de Python) que proporcionan los módulos son el controlador, y los archivos XML que hay dentro de los módulos definen las vistas.

### 3.- *Arquitectura de OpenERP*

Visto lo anterior, pasaremos a explicar cual es la arquitectura de trabajo de OpenERP, basada en Cliente/Servidor. El servidor y el cliente se comunican utilizando el protocolo XML-RPC (en las versiones más actuales se ha añadido el protocolo NET-RPC, que es más rápido). Mediante este protocolo se llama a funciones desde el cliente, pasándole los argumentos y todo lo necesario mediante HTTP y codificado en XML.

Todo el trabajo de gestión de los datos se realiza en la parte del servidor, el cliente simplemente muestra estos datos. La lógica de funcionamiento también reside en el servidor, de forma que al realizar cualquier acción el cliente pregunta al servidor qué hacer y éste le contesta con la acción a realizar. Hay tres tipos de acciones: abrir una



nueva ventana, imprimir un documento o ejecutar un asistente.

En la parte del servidor residen los objetos, que pueden residir en diferentes máquinas, o estar replicados en ellas, con fines de balanceo de carga. La capa ORM (object relational mapping) del servidor ofrece funcionalidad adicional a PostgreSQL consistente en:

- Consistencia, pruebas de validación.
- Trabajo con objetos (métodos, referencias, etc)
- Seguridad a nivel de fila (por usuario, grupo y rol).
- Acciones complejas en un grupo de recursos.
- Herencia.

Los servicios web ofrecidos del lado del servidor son los siguientes:

- SOAP
- XML-RPC
- NET-RPC

El servidor OpenERP tiene un componente llamado "motor de flujos de trabajo" que describe el flujo de documentos necesario para realizar una tarea, como hacer una venta a través de un presupuesto, albarán y, finalmente, factura. Estos flujos pueden variar entre empresas debido a la dinámica de funcionamiento y son personalizables.

También hay un motor de informes que permite la impresión de diferentes documentos, desde facturas hasta estadísticas. Todos los informes son generados en PDF, aunque esto puede cambiar. En



OpenERP tenemos, por el momento, las siguientes formas de generar un informe:

- Informes personalizados, que son creados directamente desde la interfaz del cliente sin necesidad de programar nada. Estos informes son una representación directa de los objetos de negocio.
- Informes personalizados utilizando *Openerport*, estos informes se crean a partir de dos archivos XML, una plantilla que indica los datos disponibles en el informe y una hoja de estilo XSL:RML.
- Informes grabados en código.
- Plantillas de OpenOffice.org Writer.

También podemos encontrar en el lado del servidor los objetos de negocio, que son todos los datos que contiene el programa y que se describen utilizando el módulo ORM. Éstos se distinguen entre asistentes y *widgets*. Los asistentes son grafos de ventanas y acciones personalizables (esto es similar a los flujos de trabajo, pero más orientado al usuario), y los *widgets* representan la forma en la que se muestra un campo en la interfaz de usuario. Los más comúnmente soportados (aunque hay más) son las cajas de entrada de texto, las cajas de texto, los números en coma flotante, las fechas, las casillas seleccionables, así como botones para llamar a acciones y los *widgets* de referencias entre tablas (uno a muchos, muchos a uno, etc).



## Capítulo 3

### Descripción de la solución adoptada

#### 1.- *Esquema general de la solución*

Los requisitos que debe cumplir la solución adoptada (los demandados por el cliente) son, a grandes rasgos, el llevar una cuenta de las ventas diarias, así como de las existencias en almacén teniendo como datos fuente los que generan las básculas diariamente. Es por ello que utilizaremos los siguientes módulos de OpenERP:

- **Producto (*product*):** este módulo define los objetos correspondientes a los artículos que se van a vender, así como los precios de compra y de venta y las diferentes tarifas si las hubiese. Es un módulo necesario para poder instalar el módulo de ventas, y necesario también para poder llevar la gestión de existencias en almacén.
- **Existencias (*stock*):** el módulo de existencias permitirá saber en todo momento qué cantidad de cada producto tenemos disponible en almacén. Además de ser un requisito para la implantación es requerido también por el módulo de ventas.
- **Módulo de ventas (*sale*):** con el módulo de ventas llevaremos cuenta de la facturación de la empresa. Este módulo tiene unas cuantas funciones más, sin embargo tan sólo vamos a utilizar las facturas de ventas.



## 1.- Esquema general de la solución

---

- **Módulos de localización al español:** son los módulos que empiezan con "l10n" y que proveen de las características específicas para España, como pueden ser los topónimos, el plan general contable, el CIF/NIF para empresas y algunas otras cosas.
- **Otros:** además de los módulos indicados hay otros instalados que se requieren para el correcto funcionamiento de los módulos que vamos a utilizar, sin embargo no utilizaremos directamente sus funciones. Estos son, entre otros, el módulo "base" y el módulo "account".
- **Módulo de balanzas MAPAL:** este es el módulo que programaremos para acceder a los datos de las balanzas MAPAL. Su función será extraer los datos de la web de las balanzas a petición del usuario, convertirlas en un objeto de negocio más de la base de datos y generar una lista de pedido en almacén desde la que se puede generar un movimiento de almacén y/o una factura directamente. De este modo tendremos todas las necesidades cubiertas.

## 2.- El entorno de producción, instalación

Debido a la estabilidad y estado de conservación a lo largo del tiempo de GNU/Linux, decidimos utilizar este sistema operativo para la instalación del servidor OpenERP. Además Python es nativo de Linux (aunque funcione perfectamente en Microsoft Windows y la mayoría de sistemas operativos existentes) y su instalación y configuración en este sistema es prácticamente automática. El uso del servidor también sería, en parte, para uso "personal" (no se dispone en estos momentos de un



## 2.- El entorno de producción, instalación

ordenador cliente) así que nos decantamos por la distribución Ubuntu, de Canonical.

Para empezar, debemos instalar el sistema gestor de bases de datos PostgreSQL, para ello utilizaremos la orden "sudo aptitude install postgresql". Al realizar la instalación nos pedirá la contraseña del usuario "root" para la administración del sistema gestor de bases de datos. Después de la instalación el primer paso será crear un nuevo usuario con permisos de administración de la base de datos que utilizaremos para OpenERP, aunque sin permisos de administración para el resto de bases de datos. El sistema para la creación tanto de la base de datos de OpenERP como del usuario que la gestionará es el siguiente:

```
$ sudo su postgres
```

```
$ createuser -pwprompt openerp
```

Esta orden nos pedirá la contraseña del usuario dos veces y nos preguntará si el usuario es superusuario, a lo que responderemos que no, y si podrá crear bases de datos, a lo que responderemos que sí, además de si puede crear nuevos usuarios lo cual no es necesario.

[Ctrl] + D (para salir)

Después de esto instalaremos las bibliotecas de Python y XML que OpenERP necesita para funcionar, las líneas de órdenes a ejecutar son las siguientes:

```
$ sudo apt-get install python python-libxslt1  
python-lxml python-xml
```



## 2.- El entorno de producción, instalación

```
$ sudo apt-get install python-psycopg python-psycopg2 python-imaging python-reportlab
```

```
$ sudo apt-get install python-pyparsing python-pydot graphviz python-matplotlib python-numpy python-tz gs-gpl python-pychart python-egenix-mxdatetime python-vobject
```

A continuación pasaremos a describir para qué sirven los paquetes más importantes instalados anteriormente:

- **python:** Python es el lenguaje de programación en el que OpenERP está escrito. Es un lenguaje multiplataforma e interpretado, aunque por motivos de rendimiento el sistema puede generar un código intermedio precompilado.
- **python-libxslt1 ,python-lxml y python-xml:** son los conectores de Python que permiten utilizar las bibliotecas de procesamiento de archivos en formato XML. Es necesario para que OpenERP pueda procesar las vistas, los datos iniciales, etc.
- **python-psycopg python-psycopg2:** Es el conector de Python con PostgreSQL, es necesario para que OpenERP pueda operar con la base de datos.
- **python-reportlab:** Todos los informes de OpenERP están basados en ReportLab, con esta biblioteca se pueden interpretar estos archivos escritos en formato RML e imprimir los informes desde OpenERP.





## 2.- El entorno de producción, instalación

- **Python-imaging:** Es una biblioteca que permite el procesamiento de imágenes con Python.
- **python-egenix-mxdatetime** **python-vobject:** son utilizadas por OpenERP para gestionar el tiempo y los calendarios.
- **python-pyparsing** **python-pydot** **graphviz** **python-matplotlib** **python-numpy** **python-tz** **gs-gpl** **python-pychart:** OpenERP utiliza estas bibliotecas para poder generar y mostrar todo tipo de gráficas.

Necesitamos control absoluto sobre OpenERP, así que no utilizaremos la instalación del mismo que permite el sistema gestor de paquetes "apt" que incorpora Ubuntu mediante el cual la orden sería, simplemente, "\$ **sudo aptitude install** openerp-server". También queremos la última versión de OpenERP y controlar personalmente su actualización a nuevas versiones o su no actualización a las mismas, así que procederemos a la descarga e instalación manual de OpenERP.

Las órdenes son las siguientes:

```
$ mkdir openerp
```

```
$ cd openerp
```

```
$ wget
```

```
http://www.openerp.com/download/stable/source/openerp-server-5.0.12.tar.gz
```

```
$ tar xvfz openerp-server-*.tar.gz
```

```
$ rm openerp-server-*.tar.gz
```



En este momento ya tenemos OpenERP descargado y listo para funcionar a falta de un archivo de configuración, que crearemos ahora:

```
$ sudo nano /etc/openerp-server.conf
```

En el archivo de configuración es necesario indicar algunas opciones para el correcto funcionamiento del servidor. Hay que tener en cuenta que hay más opciones en configuraciones más avanzadas de OpenERP que pueden ser útiles en algunos casos. Por ejemplo, podemos utilizar "direct\_print = True" para evitar la previsualización en PDF de los archivos a imprimir, o podemos cambiar la exactitud de los decimales en el precio con la opción "price\_acuracy = 3" (que establecería la precisión en tres decimales por defecto), pero siempre prevalecerán las opciones del servidor sobre las del cliente. Con las opciones que indicaremos a continuación el servidor funcionará correctamente y de la forma que deseamos. Por ello vamos a pasar a describir las necesarias:

- **[options]:** indica el principio de la sección de opciones de configuración, será el principio del archivo. La notación de cada uno de los parámetros de configuración es "parámetro = valor" en todos los casos.
- **port:** indica en qué puerto va a escuchar nuestro servidor por el protocolo xml-rpc. El puerto por defecto es el 8069.
- **netport:** es el puerto por el que el servidor atenderá conexiones con el protocolo net-rpc, un protocolo más rápido que xml-rpc. El valor por defecto de esta opción es 8070.



## 2.- El entorno de producción, instalación

---

- **xmlrpc:** esta opción habilita en el servidor la utilización del protocolo xml-rpc. Los valores posibles son "True" o "False". En general lo pondremos en "True".
- **netrpc:** al igual que el anterior, indica si se servirá en el protocolo net-rpc a los clientes. El valor por defecto también es "True".
- **admi\_passwd:** es la contraseña de superadministrador por defecto de OpenERP. Esta contraseña la cambiaremos desde el cliente GTK.
- **db\_host:** indica la dirección IP o el nombre de red de la máquina en la que se encuentra la base de datos.
- **db\_port:** es el puerto en el que la base de datos PostgreSQL del servidor de bases de datos está escuchando.
- **db\_name:** aquí va el nombre de la base de datos que utilizará OpenERP por defecto.
- **db\_user:** cadena de caracteres en la que se indicará el nombre de usuario con el que el servidor OpenERP se conectará a la base de datos.
- **db\_password:** es la contraseña del usuario para conectarse a la base de datos. Hay que indicar que OpenERP gestionará sus propios usuarios de forma independiente y que estos no se verán reflejados en la base de datos.
- **db\_maxconn:** aquí indicaremos el número máximo de conexiones a la base de datos. El valor por defecto es 64.



## 2.- El entorno de producción, instalación

---

- **translate\_modules:** indicamos qué módulos serán traducidos (en caso de existir una traducción disponible). Por norma general indicaremos que todos los módulos deben ser traducidos si es posible con el parámetro "[all]".
- **root\_path:** indicaremos el directorio en el que están los archivos osm, orm y report. En nuestra instalación lo hemos colocado en /opt/openerp/bin.
- **addons\_path:** aquí indicaremos la ruta en la que se encuentran los módulos de OpenERP. En nuestro servidor lo situaremos en /opt/openerp/bin/addons.

Para evitar problemas de seguridad crearemos un usuario que es el que ejecutará el servidor OpenERP y moveremos la carpeta del servidor a su lugar final y con los permisos adecuados.

```
$ useradd openerp
```

```
$ sudo mkdir /opt/openerp
```

```
$ sudo mv openerp-server-* /opt/openerp/.
```



## 2.- El entorno de producción, instalación

```
$ sudo chown openerp:openerp -R /opt/openerp/
```

```
DAEMON=/usr/bin/openerp-server
NAME=openerp-server
DESC=openerp-server
USER=openerp
test -x ${DAEMON} || exit 0
set -e
case "${1}" in
  start)
    echo -n "Starting ${DESC}: "
    start-stop-daemon --start --quiet --pidfile /var/run/${NAME}.pid \
      --chuid ${USER} --background --make-pidfile \
      --exec ${DAEMON} -- --config=/etc/openerp-server.conf
    echo "${NAME}."
    ;;
  stop)
    echo -n "Stopping ${DESC}: "
    start-stop-daemon --stop --quiet --pidfile /var/run/${NAME}.pid \
      --oknodo
    echo "${NAME}."
    ;;
  restart|force-reload)
    echo -n "Restarting ${DESC}: "
    start-stop-daemon --stop --quiet --pidfile /var/run/${NAME}.pid \
      --oknodo
```

Ahora crearemos un script de inicio de servicio en el sistema, en /etc/init.d/ y lo añadiremos al inicio para que el servicio se levante en caso de reiniciar la máquina. Para ello crearemos un archivo con el siguiente contenido en "/etc/init.d/openerp-server":



```
sleep 1
start-stop-daemon --start --quiet --pidfile /var/run/${NAME}.pid \
    --chuid ${USER} --background --make-pidfile \
    --exec ${DAEMON} -- --config=/etc/openerp-server.conf
echo "${NAME}."
;;
*)
N=/etc/init.d/${NAME}
echo "Usage: ${NAME} {start|stop|restart|force-reload}" >&2
exit 1
;;
esac
exit 0
```

Ahora lo añadimos al arranque del sistema con el siguiente comando:

```
$ sudo update-rc.d openerp-server defaults
```

Y crearemos también el lanzador en "/usr/bin" llamado openerp-server con el siguiente contenido:

```
#!/bin/sh
cd /opt/openerp/openerp-server -5.0.12/bin
exec /usr/bin/python ./openerp-server.py $@
```

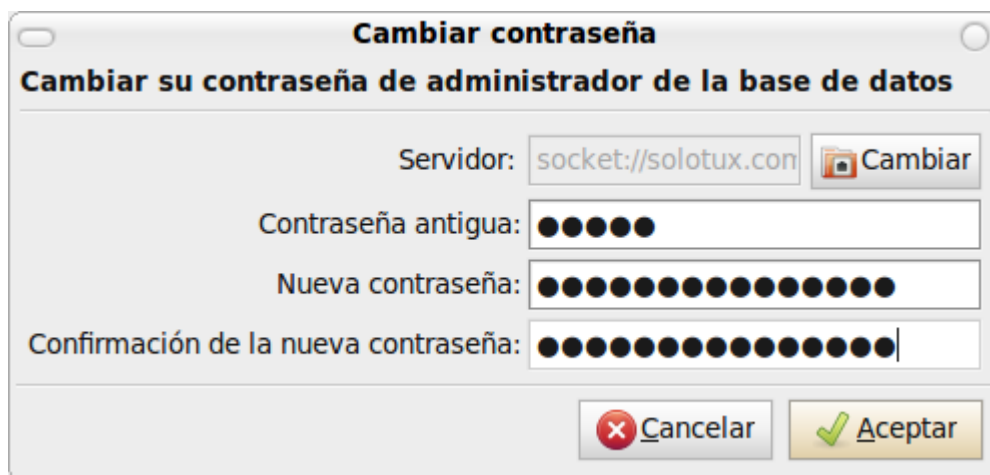
Con todo esto ya tenemos el servidor de OpenERP funcionando, aunque sin ningún dato, ninguna base de datos y ningún módulo instalado. El siguiente paso es crear la base de datos para la empresa en OpenERP e instalar y configurar los módulos necesarios. Para ello instalaremos el cliente GTK con la orden:



## 2.- El entorno de producción, instalación

```
$ sudo aptitude install openerp-client
```

En primer lugar debemos cambiar la contraseña de superusuario, que es quien gestionará las bases de datos de OpenERP desde el mismo programa. En el cliente GTK iremos al menú "archivo", "bases de datos", "contraseña de administrador". Nos pedirá la contraseña antigua, siendo ésta por defecto la palabra "admin" sin las comillas. Una vez introducida dos veces la nueva contraseña, hacer clic en aceptar para que los cambios se guarden.



Una vez ejecutado el cliente lo utilizaremos para crear la base de datos de la nueva empresa. Iremos al menú "archivo" y en el apartado "bases de datos" pulsamos sobre la opción "nueva base de datos".

Crear una nueva base de datos

Crear nueva base de datos

Servidor OpenERP:

Contraseña del super-administrador:  (admin, por defecto)

Nuevo nombre de la base de datos:

Cargar datos de demostración:

Idioma por defecto:

Contraseña del administrador:

Confirmar contraseña:

Después de crear algunas tablas un asistente preguntará qué perfil de instalación. Ya que queremos instalar la mínima cantidad de módulos posible elegimos el perfil mínimo. Aunque en este punto existen diferentes preconfiguraciones que son comunes a algunas tipologías de empresa y que podrían ser útiles para operadores poco avanzados que no conozcan en profundidad qué módulos instalar en cada situación. Sin embargo nosotros instalaremos lo justo y necesario, sin poner módulos de más ni de menos.

Instalación

Seleccione un perfil

Perfil :

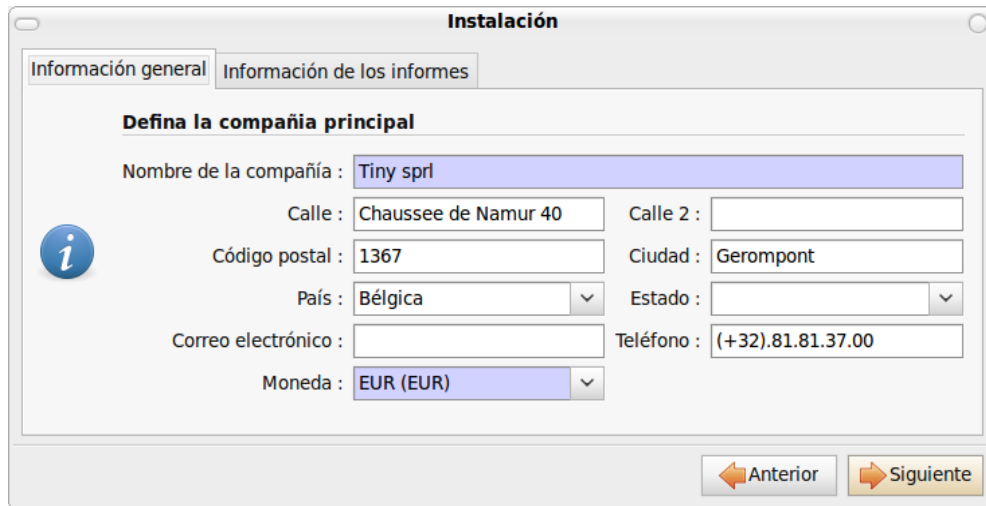
i

Un perfil instala una preselección de módulos para una necesidad específica. Estos perfiles han sido creados para ayudarle a descubrir los diferentes aspectos de OpenERP. Esto es sólo un punto de partida, OpenERP dispone de 300+ módulos.

Posteriormente podrá instalar más módulos desde el menú Administración.



A continuación introducimos la información de la empresa:



**Instalación**

Información general | Información de los informes

**Defina la compañía principal**

Nombre de la compañía : Tiny sprl

Calle : Chaussee de Namur 40    Calle 2 :

Código postal : 1367    Ciudad : Gerompont

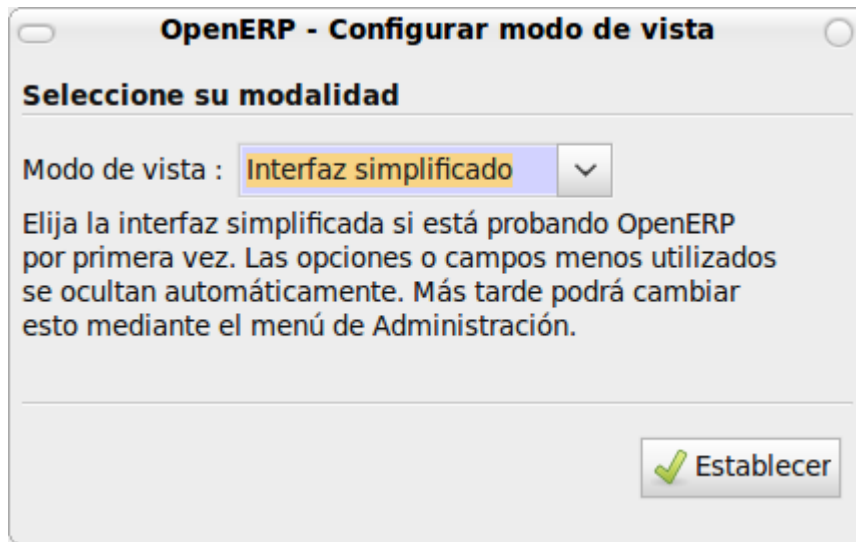
País : Bélgica    Estado :

Correo electrónico :     Teléfono : (+32).81.81.37.00

Moneda : EUR (EUR)

Anterior    Siguiete

En el siguiente paso del asistente seleccionaremos la interfaz simplificada. La diferencia entre interfaz simplificada e interfaz extendida es que en la simplificada se ocultan las opciones menos utilizadas o más avanzadas. En la práctica esto tiene poco efecto para las configuraciones más simples de OpenERP, sin embargo puede ser útil en algunas situaciones.



Después de preguntar los datos de la empresa el asistente pedirá datos de los usuarios a utilizar, en nuestro caso utilizaremos dos (uno para cada usuario del programa aparte del administrador). También tenemos la opción de configurar los roles de usuario llegados a este punto, sin embargo, debido a que aún no tenemos más módulos que los de una instalación mínima, no se han creado aún los roles correspondientes a los mismos. Así que obviaremos este paso para configurarlo luego y ya estará instalada la aplicación base.



## 2.- El entorno de producción, instalación

**OpenERP - Configurar usuario**

**Definir nuevos usuarios**

Nombre :  Activo :

Usuario :  ?Contraseña :

Usuario Grupos Roles

Dirección :  Compañía :

Acción inicial :  Acción de menú :

Idioma :  Zona horaria :

Firma :

Una vez instalada la base de datos con la configuración básica procederemos a la instalación de los módulos que vamos a necesitar. En el menú lateral podemos encontrar la sección de administración. Una vez dentro desplegaremos el menú de la derecha, la opción "administración de módulos" y haremos doble clic en la sección "módulos no instalados".



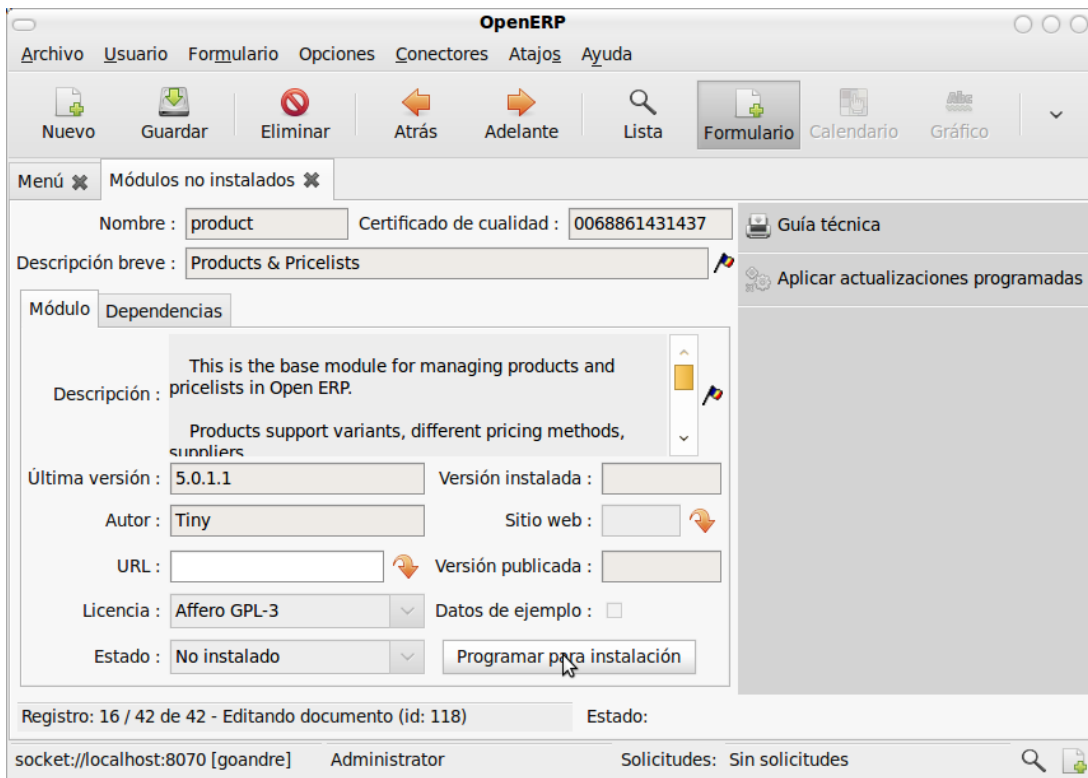
## 2.- El entorno de producción, instalación



En este punto accederemos a una lista de los módulos disponibles y que no están instalados, iremos marcando todos los módulos para programar su instalación, para después pulsar sobre el botón “aplicar actualizaciones programadas” y éstas empezarán a instalarse. Los módulos a instalar son:

- account
- l10n\_chart\_ES
- product
- base\_iban
- sale

- stock
- l10n\_ES\_partner
- l10n\_ES\_toponyms
- balanzas



### 3.- El entorno de desarrollo, instalación

Ya que el proyecto iba a funcionar en Ubuntu, la distribución que utilizaremos será esa misma. Para ello montaremos un servidor de pruebas en un proyecto de Eclipse. El IDE elegido es Eclipse ya que tiene uno de los mejores complementos para desarrollar en Python, PyDev.



### 3.- El entorno de desarrollo, instalación

---

Antes que nada, es necesario explicar el por qué hemos elegido el IDE Eclipse como entorno de desarrollo. Entre sus principales ventajas, algunas de ellas compartidas con OpenERP, podemos destacar:

- Es software libre.
- Es multiplataforma.
- Gran comunidad de desarrolladores.
- Provee el mismo entorno de desarrollo para la mayoría de lenguajes (C, C++, Java, Python, PHP...)
- Cuenta con PyDev, el mejor entorno existente hasta el momento para programar en Python.
- Entre los desarrolladores, aparte de la comunidad, encontramos empresas de la talla de: IBM, Adobe, Borland, Google, HP, Intel, Motorola, Nokia, Oracle y SAP.

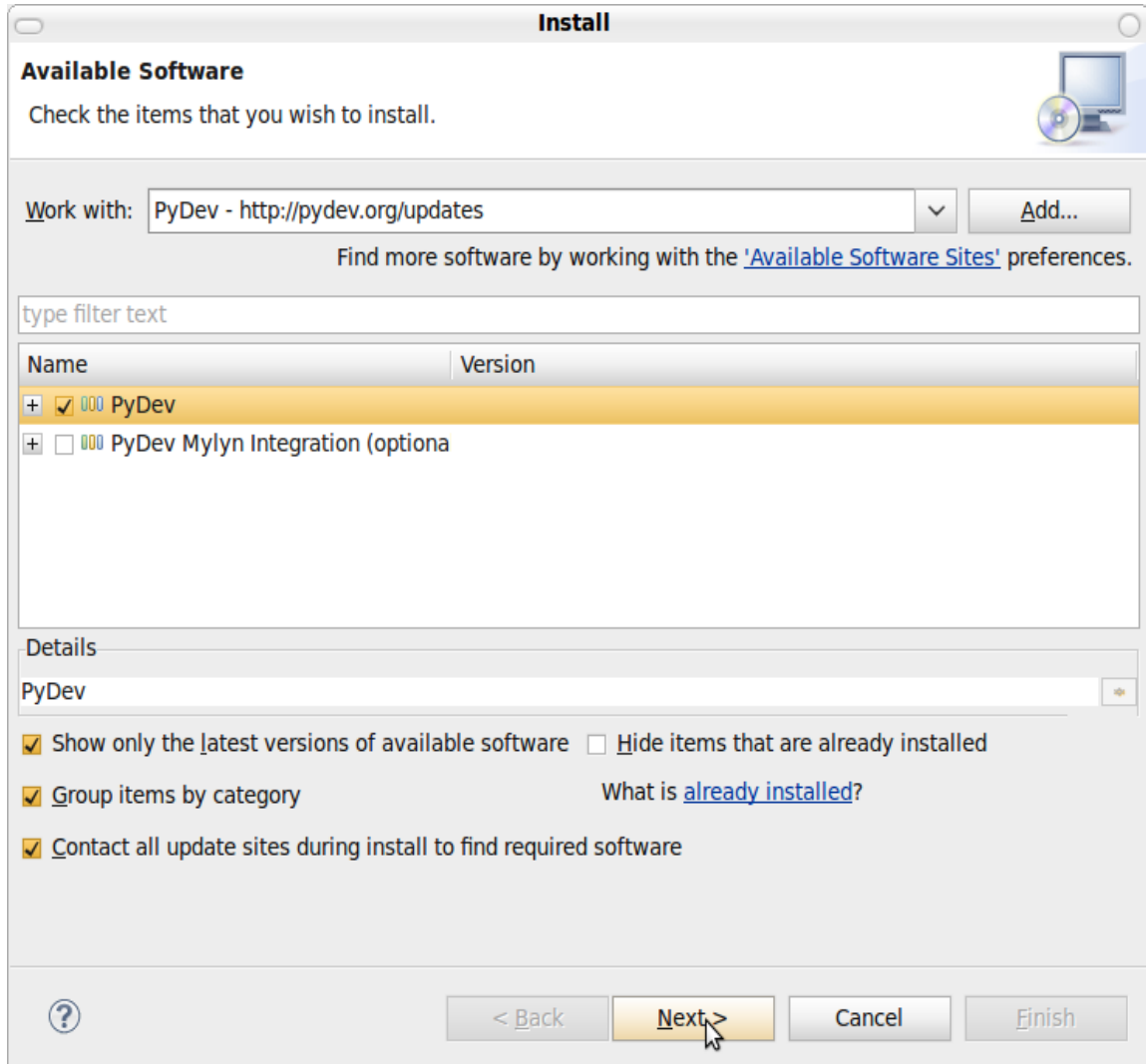
En primer lugar, para proceder a la instalación del entorno de desarrollo podemos utilizar el gestor de paquetes de Ubuntu e instalar el IDE con sólo una orden:

```
$ sudo aptitude install eclipse
```

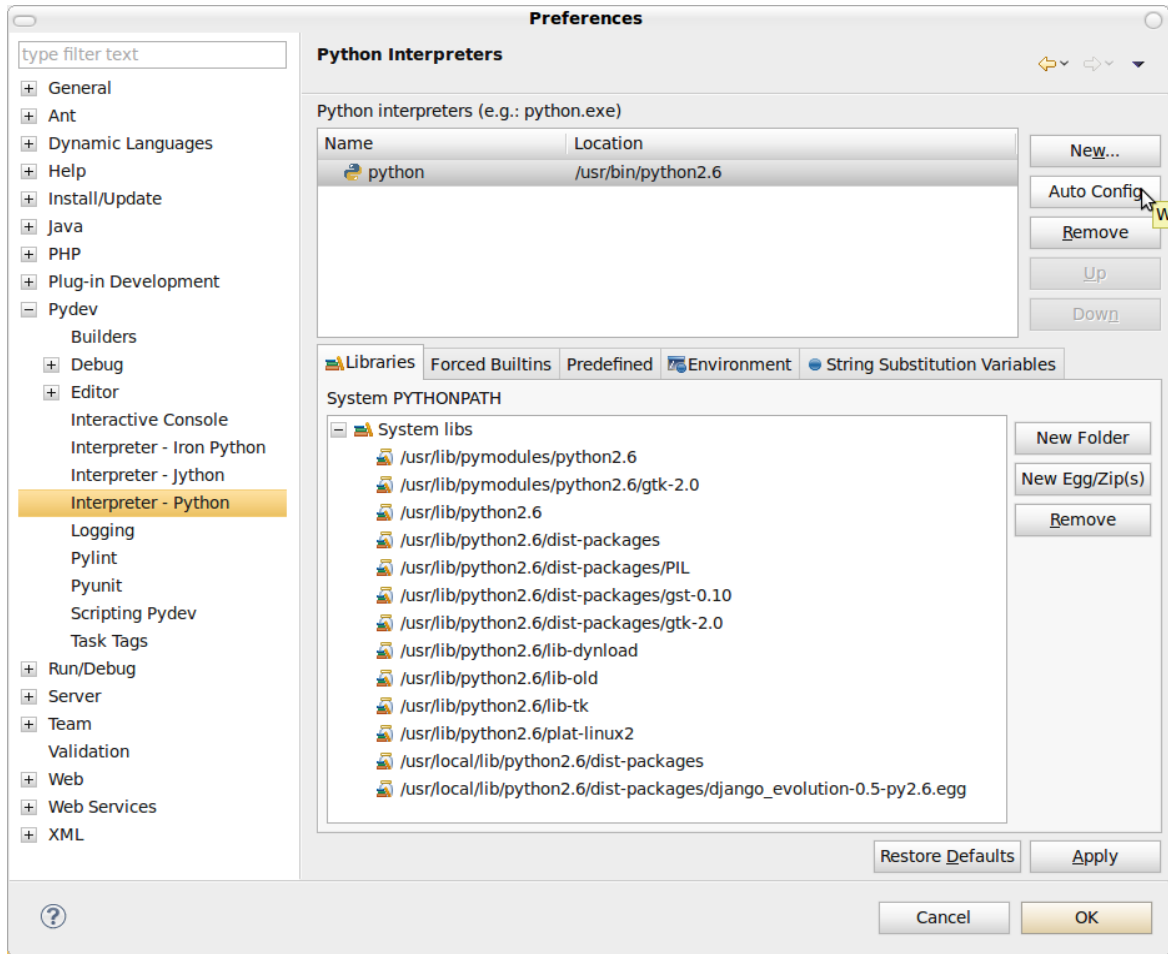
Para instalar PyDev en Eclipse hay que entrar en el menú "help, install new software". Una vez ahí añadimos un nuevo repositorio "<http://pydev.org/update>" para poder descargar desde ahí el complemento. Ahora seleccionaremos para instalar PyDev y Eclipse lo descargará e instalará dejándolo preparado para funcionar.



### 3.- El entorno de desarrollo, instalación



Una vez instalado PyDev hay que configurar el interprete de Python, para ello iremos a las preferencias de Eclipse, y en la sección de Python pulsamos el botón de configuración automática del editor.



El sistema para instalar el servidor de desarrollo a modo de proyecto de PyDev es similar a la instalación en el servidor de producción. Crearemos entonces un nuevo proyecto de PyDev, y dentro de este proyecto volcamos todo el contenido del archivo descargado de *OpenERP*.

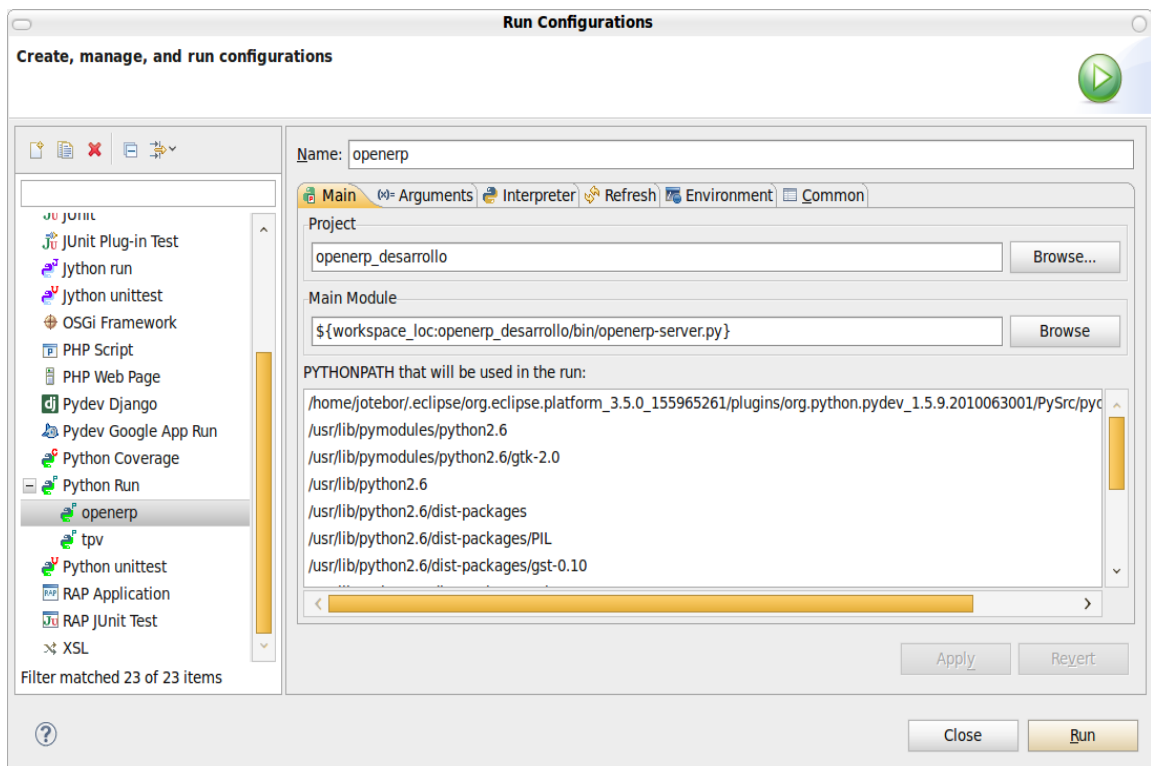
Para poder lanzar el servidor y realizar tareas de depuración desde Eclipse hay que entrar en el menú "Run, Run configurations". En el apartado de Python creamos un nuevo lanzador en el que *openerp.py* sea el módulo principal. Al lanzar el servidor se creará



automáticamente el archivo de configuración dentro de la carpeta *OpenERP*, que debemos editar para introducir el usuario y contraseña de la base de datos del servidor de desarrollo.

En el apartado *arguments* del "run configurations" es recomendable indicar dónde irá el log y lo pondremos en modo de depuración:

```
--logfile=$HOME/openerp_desarrollo.log --log-  
level=debug_rpc debug
```



Una vez configurado el entorno ejecutaremos el cliente y lo configuraremos exactamente igual que en la sección anterior, con los módulos básicos, para poder desarrollar en un entorno lo más parecido



al entorno de usuario final.

## 4.- La programación del módulo

### 1.- Módulo OpenERP

Cualquier módulo de OpenERP es como un paquete de Python, debe tener un archivo llamado “\_\_init\_\_.py”. Este archivo se ejecuta al inicio del programa y su función es importar los archivos que deben ser cargados al inicio. Lo colocaremos dentro de una carpeta con el nombre del módulo a crear, en nuestro caso “balanzas”, dentro de la carpeta “addons”.

```
import balanzas contenido de nuestro __init__.py
```

El siguiente archivo a crear es el archivo “\_\_terp\_\_.py”, que contiene la información del módulo que vamos a programar, además de las dependencias que deben ser satisfechas en la instalación del módulo, y las vistas que se deben actualizar para su correcto funcionamiento.

```
{
    "name" : "Modulo para la gestión de balanzas",
    "version" : "0.1",
    "author" : "SoloTux",
    "category" : "Generic Modules/Others",
    "website": "http://www.solotux.com",
    "description": """ Este modulo gestiona la conexion de OpenERP con
diferentes tipos de balanzas. Balanzas soportadas:
                        - MAPAL (Lectura)          """
    "depends" : ["base", "product", "stock"],
    "init_xml" : [],
    "update_xml" : ['balanzas_view.xml'],
    "active": False,
    "installable": True
}
```



Una vez creados estos archivos escribiremos el tercer archivo común en todos los módulos. Este archivo tiene generalmente (aunque no es necesario) el mismo nombre que el módulo, aunque con la extensión “.py”. Dentro de este archivo es donde va el código del módulo.

Ahora es el momento de analizar qué es lo que necesitamos. Hay que tener en cuenta la separación MVC (modelo vista controlador), en nuestro módulo. Lo que necesitamos es crear un modelo que integre todos los datos existentes (o no) en la página web de la balanza. Pueden existir balanzas con más o menos datos, aunque en esencia los más importantes siempre estarán (que son las ventas), así que versión a versión habrá que ir adaptando el modelo a las nuevas balanzas si esto fuera necesario de forma que un único modelo pueda ser utilizado para todas ellas independientemente de los datos que éstas contengan.

En caso de encontrarnos con balanzas con capacidades especiales y fuera de lo común, es decir, con configuraciones y modos de operación que no sean comunes en el resto de balanzas (casos especiales) hemos decidido implementar esas características especiales en módulos específicos de las mismas utilizando la herencia. De este modo lo que ya hayamos programado no se perderá y podremos adaptar el código a ese caso especial.

En este momento hay que centrarse en qué datos va a contener el modelo, cómo estarán estos relacionados entre ellos y olvidar por el momento la forma en la que vamos a obtener esos datos (controlador) y la forma en la que se los presentaremos al usuario (vista).



Para crear la estructura mental de datos que debe contener este módulo necesitamos cuatro tablas de datos a las que hemos llamado: `balanzas_balanza`, `balanzas_ventas_diarias`, `balanzas_ventas_totales` y `balanzas_vendedor`.

- **balanzas\_balanza:** esta tabla contendrá los datos de cada una de las balanzas que vayamos a utilizar, a estas balanzas nos conectaremos mediante el protocolo HTTP, de forma que tenemos que guardar su dirección IP y el puerto al que responde el servidor incorporado en la máquina. En este momento el módulo tan sólo soporta una marca y modelo de balanza, en una determinada versión del software. Sin embargo en un futuro deberá poder procesar datos de diferentes balanzas. Es por ello que guardaremos estos datos también. A cada balanza van asociados unos determinados vendedores, un determinado grupo de productos y una balanza padre (o ninguna). Estas balanzas en concreto van conectadas en serie, de forma que tan sólo accedemos a la balanza padre para obtener los datos unificados de todas ellas. Pero esto puede cambiar, de forma que guardaremos este dato por escalabilidad. Además, para acceder rápidamente a los datos más solicitados, pondremos el total de ventas global y el total de ventas del último día consultado.



- **balanzas\_ventas\_diarias:** esta tabla contendrá una fila por balanza, día y vendedor con un resumen de las ventas de ese día. Además de las ventas es necesario conocer el neto de ventas y el total de los abonos realizados. También pondremos el total de ventas acumulado por ese vendedor en esa balanza.
- **balanzas\_ventas\_totales:** aquí pondremos una estructura similar a la tabla `balanzas_ventas_diarias`, pero dispondrá del total acumulado por vendedor y balanza. Esta tabla la creamos por motivos de rendimiento ya que el módulo está pensado para un microprocesador modesto.
- **balanzas\_vendedor:** es necesario llevar un registro de qué vendedores pertenecen a cada balanza, debido a la forma de obtención de datos de las balanzas utilizaremos el nombre de este vendedor y el identificador de la balanza para crear la relación.
- **balanzas\_caja:** esta tabla representará el contenido en dinero que debe haber en la caja de cada balanza en la fecha dada. Esto es necesario para poder implementar el arqueo de caja.

Para poder materializar nuestro modelo en objetos de OpenERP. Las variables principales del objeto que da forma a cada una de las tablas son las siguientes:

- **name:** es una variable privada de la clase que representa el nombre del objeto.
- **columns:** es una variable privada en forma de diccionario donde la clave es el nombre de la columna en la tabla y el valor es la descripción de los datos que contiene.

Cada uno de estos objetos será representado en PostgreSQL creando una tabla de nombre "name" con las columnas "columns". Para comenzar la construcción de nuestro modelo necesitamos una clase que represente a una balanza. El código que representa los objetos que anteriormente indicamos es el siguiente.

Para balanzas\_balanza:

```
_name = 'balanzas.balanza'
_description = 'Balanza'
_columns = {
    'name': fields.char('Dirección IP/Nombre de dominio', size=50,
required=True),
    'puerto': fields.char('Puerto', size=4, required=True),
    'marca': fields.char('Marca', size=50),
    'modelo': fields.char('Modelo', size=50),
    'version': fields.char('Versión', size=50),
    'vendedores_ids':fields.one2many('balanzas.vendedo r', 'balanza_id',
'Vendedores', required=False),
    'productos_ids':fields.one2many('product.product', 'balanza_id',
'Productos', required = False),
    'padre' : fields.boolean('Principal', help="La balanza principal es la
que sera controlada desde el panel global."),
    'ventas_diarias_ids':fields.one2many('balanzas.ven tas_diarias',
'balanza_id', 'Ventas diarias', required=False),
    'ventas_totales_ids':fields.one2many('balanzas.ven tas_totales',
'balanza_id', 'Ventas totales', required=False),
    'caja_inicial':fields.float('Caja inicial', digits=(6,2)),
    'caja_ids':fields.one2many('balanzas.caja', 'balanza_id', 'Caja',
required=False)
```

Para `balanzas_vendedor`:

```
_name = 'balanzas.vendedor'
_description = 'Vendedor que pertenece a una bascula'
_columns = {
    'balanza_id':fields.many2one('balanzas.balanza', 'Balanza',
    required=False),
    'name':fields.char('Nombre', size=20, required=False, readonly=False),
    'numero':fields.char('Numero', size=2, required=False, readonly=False),
}
```

Para `balanzas_ventas_diarias`:

```
_name = 'balanzas.ventas_diarias'
_description = 'Ventas diarias por vendedor'
_columns = {
    'vendedor_id':fields.many2one('balanzas.vendedor', 'Vendedor'),
    'total': fields.float('Total', digits=(6,2)),
    'abonos': fields.float('Abonos', digits=(6,2)),
    'neto': fields.float('Neto', digits=(6,2)),
    'acumulado': fields.float('Acumulado', digits=(6,2)),
    'date': fields.date('Fecha'),
    'vista_id':fields.many2one('balanzas.vista', 'Vista Global',
    required=False),
    'balanza_id':fields.many2one('balanzas.balanza', 'Balanza',
    required=False),
}
```

Para `balanzas_caja`:

```
_name = 'balanzas.caja'
_description = 'Caja diaria'
_columns = {
    'vendedor_id':fields.many2one('balanzas.vendedor', 'Vendedor'),
    'caja': fields.float('Caja', digits=(6,2)),
    'date': fields.datetime('Fecha y Hora'),
    'vista_id':fields.many2one('balanzas.vista', 'Vista Global',
required=False),
    'balanza_id':fields.many2one('balanzas.balanza', 'Balanza',
required=False)
}
```

¿Por qué hemos hecho una tabla llamada `balanzas_vista`? El cliente solicitó claramente que toda la información procedente de las balanzas estuviera disponible en sólo una pantalla. Para conseguir este objetivo en OpenERP existe una solución llamada “dashboard”. Sin embargo no es posible interactuar con estas vistas ya que están pensadas para mostrar gráficos y estadísticas. De modo que la solución es crear un objeto “vista” con el cual se relacionan todos los demás objetos que queremos mostrar en esta vista:

```
'ventas_diarias_ids':fields.one2many('balanzas.ventas_diarias',
'vista_id', 'Ventas diarias', required=False),
    'ventas_totales_ids':fields.one2many('balanzas.ventas_totales'
, 'vista_id', 'Ventas totales', required=False),
    'albaranes_ids':fields.one2many('stock.picking', 'vista_id',
'Albaranes', required=False),
    'caja_ids':fields.one2many('balanzas.caja', 'vista_id',
'Caja', required=False)
```





De esta forma conseguiremos relacionar todos los objetos que queremos utilizar con la vista y así tendremos la opción de interactuar con los mismos. Al verse esta "tabla" reflejada en la base de datos nos encontraríamos con duplicidad de datos, siendo esto ineficiente e innecesario. OpenERP cuenta con una opción para solventar este inconveniente, es la opción "auto = False" que hace que no se guarde la información contenida en esta vista en nuestra base de datos.

Con estas pocas líneas de código hemos conseguido representar todos los datos que necesitamos guardar por el momento referente a las balanzas en nuestra base de datos.

Para alimentar el modelo de datos es necesario crear métodos que se encarguen de procesar la información correspondiente a cada modelo y crear los objetos para ser representados en OpenERP. OpenERP ofrece una serie de métodos a través de su ORM (object relational mapping) para poder interactuar con todos los objetos de negocio, ya que todos heredan el ORM.

Hay que tener en cuenta que los métodos no pueden ser llamados directamente tal y como se hace en cualquier programa normal hecho en Python. Lo que OpenERP hace es crear una agrupación común de métodos (que comúnmente se denomina "pool") para cada objeto. Es por ello que hay que llamar a los métodos a través del "pool" del propio objeto con la siguiente línea:

```
self.pool.get.(nombre_del_objeto).método_a_llamar(parametros)
```

A continuación pasaremos a explicar qué objetos son los que nosotros necesitamos indicando el nombre del método y la función del mismo:



#### 4.- La programación del módulo

---

- **browse:** este método transformará las filas que le pidamos de PostgreSQL a un objeto de Python. Los parámetros que debemos pasar a la función son:
  - el cursor a la base de datos.
  - el id del usuario que realiza la petición.
  - el contexto (idioma, zona horaria, etc), por defecto es "None".
  - id o lista de ids de los objetos a convertir.
  - Esta función devuelve el objeto o la lista de objetos convertidos.
  - Ejemplo:

```
familias_existentes=self.pool.get('product.category').  
browse(cr, uid, familias_existentes)
```
- **create:** esta función crea objetos de OpenERP y los inserta en la base de datos en forma de filas. Los parámetros de esta función son:
  - el cursor a la base de datos.
  - El id del usuario que realiza la petición.
  - El diccionario con formato "parámetro : valor" que contiene el objeto u objetos que vamos a crear.
  - La función devuelve el identificador o lista de identificadores del objeto u objetos creados.



- Ejemplo: 

```
self.pool.get('product.product').create(cr, uid, { 'name': productos[i][0], 'categ_id': id_categoria, 'list_price': productos[i][2].replace(u',', u'.'), 'balanza_id': id_bascula, 'default_code': productos[i][3], 'uom_id': 2, 'uom_po_id': 2 })
```
- **write:** con este método se pueden editar objetos ya creados (que son representaciones de filas de una tabla). En nuestro caso servirá, principalmente, para modificar precios de productos que se han modificado en las balanzas. Los parámetros son:
  - cursor de la base de datos, al igual que en todos los casos anteriores.
  - Identificador del usuario que realiza la acción.
  - Identificador del objeto a editar.
  - Diccionario con el formato "clave: valor" con los campos del objeto a modificar.
  - Devuelve "True" siempre.
  - Ejemplo de uso:

```
self.pool.get('product.template').write(cr, uid, id_producto {'categ_id': id_categoria, 'list_price': productos[i][2].replace(u',', u'.'), })
```



- **unlink:** Este método borra un objeto de OpenERP, los parámetros que hay que pasarle son:
  - Cursor a la base de datos.
  - Identificador del usuario que está realizando la acción.
  - Identificador o lista de identificadores de los objetos que se van a borrar.
  - Devuelve siempre "True".
  - Ejemplo de uso:

```
self.pool.get('balanzas.ventas_diarias').unlink(c  
r, uid, ventas_ids)
```

En algunos casos estos métodos pueden lanzar alguna excepción, es por ello que hay que contemplar todos estos casos para poder manejar la excepción sin que ocasione problemas de funcionamiento en el programa. Éstos casos son los siguientes:

- **AccessError:** se produce cuando el usuario no tiene permiso de escritura o modificación para el objeto indicado.
- **ValidateError:** se lanza al intentar introducir un valor inválido para un campo.
- **UserError:** se lanza a consecuencia de crear algún tipo de bucle infinito debido a un error de programación, como por ejemplo indicar que el padre de una categoría



es esa misma categoría.

Además de los métodos anteriores, la biblioteca `psycpg2` proporciona otros dos métodos para poder operar con la base de datos:

- **execute:** es un método que ejecuta una sentencia SQL directamente sobre la base de datos.
- **Fetchall:** obtiene los resultados de la ejecución de la sentencia SQL con la orden anterior (`execute`) en forma de tupla.

Con todos estos métodos disponibles lo que nos queda es procesar todos los datos obtenidos por el controlador y crear, modificar o eliminar los objetos que sean necesarios según el caso. Para facilitar la lectura del código, el nombre de los métodos será el mismo tanto en el modelo como en el controlador. Pasamos a enumerar y describir estos métodos:

- **obtener\_vendedores:** este método obtiene dos listas de vendedores desde dos fuentes diferentes. En primer lugar de la base de datos donde tenemos guardados los vendedores que hay dados de alta en el sistema, en segundo lugar consultará la lista de vendedores que hay en las balanzas.

```
nombres = web.obtener_vendedores(ip_puerto)
cr.execute("SELECT name, numero FROM balanzas_vendedor")
nombres_vendedores = cr.fetchall()
```

Para insertar los nuevos vendedores, en caso de introducirse en las básculas, lo haremos mediante una simple comparación de las listas con el operador de Python "not in":



```
for i in range(1, len(nombres), 2):  
  
    if nombres[i] not in nombres_vendedores:  
  
        self.pool.get('balanzas.vendedor').create(cr, uid,  
        {'balanza_id':ids[0], 'name':nombres[i], 'numero':nombres[i -1]})
```

En caso de que un vendedor sea borrado de la balanza éste no será borrado en el programa, ya que lo necesitaremos para conservar el histórico de ventas de este vendedor. OpenERP, al soportar auditoría de acciones de forma totalmente integrada, ya está preparado para este cometido con su configuración del parámetro "cascade", que impedirá el borrado en cascada de todos los datos referentes a cualquier objeto. Esto es necesario, además de para el histórico, por razones puramente legales en caso de integrar la contabilidad en el programa. Además de esta forma podremos disponer de estadísticas de años atrás (en realidad de toda la vida de la empresa desde que empezó a utilizar OpenERP).

También realizaremos un "mapeo" de los vendedores con res.partner, que es un objeto de negocio que representa a los clientes y proveedores en OpenERP, para posteriormente poder realizar facturas y albaranes diarios con sus ventas tratándolo a efectos prácticos como si de un cliente más se tratase:

```
self.pool.get('res.partner').create(cr, uid, {'name':nombres[i]})  
cr.execute("SELECT id FROM res_partner WHERE name = '" + nombres[i] + "'")  
id = cr.fetchall()  
self.pool.get('res.partner.address').create(cr, uid, {'name':nombres[i],  
        'partner_id':id[0][0]})
```



- **obtener\_productos:** este método obtiene todos los productos con su nombre y las categorías que hay en la balanza. Aparte de los productos que hay en la balanza hay que contemplar algunos casos especiales, que son las ventas que se realizan sin utilizar la memoria de producto de la que dispone la balanza. Estas ventas se realizan indicando directamente el precio del producto, ya sea a peso o por unidades.

```
cr.execute("SELECT id FROM product_template WHERE name = 'otros'")
if len(cr.fetchall()) is 0:
    self.pool.get('product.product').create(cr, uid, {
        'name':'otros',
        'categ_id':id_categoria,
        'list_price':1.00,
        'balanza_id':id_bascula,
        'default_code':'OTR',
        'uom_id':1,
        'uom_po_id':1
    })
```

Identifiquemos estos casos:

- **ventas directas sin peso:** esto es cuando se vende un producto empaquetado con precio predefinido, es decir, por unidades. Este tipo de ventas se refleja en los tickets de la balanza con un +E en la línea de venta del artículo.
- **ventas directas con peso:** esto es cuando se vende un producto sin empaquetar con precio predefinido, es decir, por kilogramos. Este tipo de ventas se refleja en los tickets de la balanza con un +P en la línea de venta del artículo.



#### 4.- La programación del módulo

- **Ventas de abono sin peso:** cuando hay abonos de un producto vendido por venta directa sin peso, la cantidad devuelta puede indicarse directamente a la balanza. Esto se indicará con los caracteres -E en la línea del producto en el tique.
- **Ventas de abono con peso:** cuando hay abonos de un producto vendido por venta directa con peso, la cantidad devuelta puede indicarse directamente a la balanza. Esto se indicará con los caracteres -P en la línea del producto en el tique.
- **Ventas sin nombre:** ventas de productos que no tienen nombre en la báscula. El vendedor introduce el precio por kilo y la báscula indica los kilos y el precio total. También puede indicarse por unidades. Este caso lo trataremos como un producto más y será gestionado mediante el controlador *web.py*.

```
#Creamos un producto para representar los abonos
cr.execute("SELECT id FROM product_template WHERE name = 'abonos'")
if len(cr.fetchall()) is 0:

    self.pool.get('product.product').create(cr, uid, {
                                                'name':'abonos',
                                                'categ_id':id_categoria,
                                                'list_price':-1.00,
                                                'balanza_id':id_bascula,
                                                'default_code':'ABO',
                                                'uom_id':1,
                                                'uom_po_id':1
                                                })
```



```
V02N00112 17:42 03/09/10
OP   kg   EUR/kg   EUR
1 +E                4,80
2 +P    2    7,60   15,20
3 0,620   9,80   6,08
4 0,515   8,20   4,22
5 0,220  15,50   3,41
6 0,345   9,90   3,42
=====
```

Además de estos casos especiales tendremos que contemplar el caso del producto sin categoría que pueda estar insertado en la balanza, ya que las balanzas lo permiten pero OpenERP exige una categoría al menos para los productos. Para solucionar este problema el módulo creará una categoría llamada "ninguna" en OpenERP y ésta será la categoría en la que incluiremos los productos que en la báscula no tengan categoría definida.

```
#Creamos una familia para productos sin familia
cr.execute("SELECT id FROM product_category WHERE name =
'ninguna'")
ids = cr.fetchall()
if len(ids) is 0:

    self.pool.get('product.category').create(cr, uid,
{'name':'ninguna'})
```



Una vez contemplados y tratados adecuadamente todos y cada uno de los casos especiales que existen en el problema el programa ya puede tratar las familias existentes en las balanzas así como los productos. Así los obtendremos y serán añadidos a OpenERP teniendo siempre en cuenta que los precios de los productos pueden ser cambiados por el cliente en cualquier momento desde la interfaz de la báscula. Tal vez en un futuro se implemente el volcado de productos y precios desde OpenERP a las balanzas, aunque por el momento esto no es necesario.

- **obtener\_ventas\_diarias:** con este método obtendremos el “gran total diario” desde la báscula. Cada una de las ventas diarias estará relacionada con un vendedor mediante el siguiente código:

```
'vendedor_id':fields.many2one('balanzas.vendedor', 'Vendedor'),
```

Teniendo todo esto en cuenta podremos obtener todas las ventas llamando al método homónimo del controlador. Habrá que recorrer entonces el vector que nos devuelve este método, que será un vector de “ventas\_diarias”, y guardando cada una de las ventas en el vendedor correspondiente.

Debido al bajo volumen de datos que es capaz de gestionar cada una de estas balanzas se tomó la decisión de recrear al completo las ventas cada vez que éstas son obtenidas. No obstante, si es necesario, en posteriores versiones esto puede cambiar omitiendo ventas anteriores a las registradas en OpenERP y prestando atención sólo a las nuevas ventas que se hayan realizado en la balanza.

```
datos = web.obtener_gran_total_diario(ip_puerto)
vendedores = datos[0]
ventas_diarias = datos[1]

for i in range(0, len(ventas_diarias), 3):

    cr.execute("SELECT id FROM balanzas_ventas_diarias WHERE date = '" + ventas_diarias[i][0] + "'")
    ventas_ids = cr.fetchall()

    self.pool.get('balanzas.ventas_diarias').unlink(cr, uid, ventas_ids)

    for z in range(len(vendedores)):

        cr.execute("SELECT id FROM balanzas_vendedor WHERE name = '" + vendedores[z] + "'")
        vendedor_id = cr.fetchall()[0][0]

        self.pool.get('balanzas.ventas_diarias').create(cr, uid, {'date':ventas_diarias[i][0],
'vista_id':ids[0],
'balanza_id':balanza_id,
'vendedor_id':vendedor_id,
'total':ventas_diarias[i][z + 1].replace(u',', u'.').strip(),
'abonos':ventas_diarias[i + 1][z + 1].replace(u',', u'.').strip(),
'neto':ventas_diarias[i + 2][z + 1].replace(u',', u'.').strip()})
```

- **obtener\_tiquets:** La labor que este método realiza es algo más compleja que la de los anteriores métodos. El objetivo es crear un albarán diario con las ventas de cada vendedor. Para llevar a cabo esta tarea hemos creado un método privado en la misma clase donde reside este método que llamará al método *obtener\_tiquets* del controlador "web.py". Este método privado se encargará de separar los diferentes modelos de tiques dependiendo de la interfaz de la balanza. Dependiendo del modelo el total de los tiques puede obtenerse de una

forma u otra, este trabajo lo realizará *obtener\_total\_tiquets*. El método *obtener\_tiquets* del modelo simplemente creará los albaranes con los totales ya calculados. Para detallar un poco más, *\_\_obtener\_total\_tiquets* (el método privado) construye un vector de vendedores con un tamaño por vendedor dependiendo de las líneas de ventas del mismo. Agrupa las ventas del mismo producto en una sola línea del albarán y calcula el total vendido de cada producto teniendo además en cuenta los casos especiales de abonos (-E y -P) y ventas directas (+E, +P).

```
if str(vendedores[i][z][2]).startswith(u'-P') and str(aux[j][2]).startswith(u'-P'):  
  
    aux[j][4] = float(aux[j][4]) + float(vendedores[i][z][4])  
    afegit = True  
    break  
  
if str(vendedores[i][z][2]).startswith(u'+P') and str(aux[j][2]).startswith(u'+P'):  
  
    aux[j][4] = float(aux[j][4]) + float(vendedores[i][z][4])  
    afegit = True  
    break  
  
if cmp(vendedores[i][z][2], u'-E') is 0 and cmp(aux[j][2], u'-E') is 0:  
  
    aux[j][4] = float(aux[j][4]) + float(vendedores[i][z][4])  
    afegit = True  
    break  
  
if cmp(vendedores[i][z][2], u'+E') is 0 and cmp(aux[j][2], u'+E') is 0:  
  
    aux[j][4] = float(aux[j][4]) + float(vendedores[i][z][4])  
    afegit = True  
    break
```

Luego, el método privado *\_\_obtener\_total\_tiquets* pasa el resultado a *obtener\_tiquets*. *obtener\_tiquets* creará un albarán por vendedor teniendo en cuenta los siguientes puntos:

Los albaranes en OpenERP son llamados pedidos de ventas.

El albarán debe ser de tipo "out" para que sea un albarán de salida.

Es necesario poner el estado de la factura (invoice\_state) en "para ser facturado" (2binvoiced). Así, una vez el usuario revise que los albaranes estén correctos podrá crear una factura a partir de ellos.

```
#Creamos un 'albaran' (productos salientes en oerp), tiene que ser de tipo 'out'  
  
self.pool.get('stock.picking').create(cr, uid, {  
    'address_id':partner,  
    'type':'out',  
    #Para poder crear fact.  
    'invoice_state':'2binvoiced',  
    'vista_id':vista_id,  
    'balanza_id':balanza_id  
    })
```

Una vez hemos creado los albaranes, debemos crear las líneas de venta para cada vendedor. Tendremos aquí también en cuenta todos los casos especiales mencionados anteriormente.

```
#Vamos a recorrer las lineas de ventas para este vendedor. para ir añadiendolas  
#al albaran  
for z in range(len(vendedores[i])):  
    if str(vendedores[i][z][2]).startswith(u'-P'):  
        cr.execute("SELECT id FROM product_template WHERE name = 'abonos'")  
        id_otros = cr.fetchall()[0][0]  
        self.pool.get('stock.move').create(cr, uid, {  
            'product_id':id_otros,  
            'name':'abonos (-P)',  
            'picking_id':id_albaran,  
            'product_uom':1,  
            'location_id':10,  
            'location_dest_id':10,  
            'product_qty':vendedores[i][z][4],  
        })
```

Debemos tener especial cuidado en el caso de los productos sin nombre. Con el fin de poder diferenciarlos utilizaremos el precio de venta. Si se da el caso de que no exista una línea de venta con el precio de venta dado para este producto sin nombre crearemos la línea asignándole la categoría “ninguna”.

```
#Caso en el que no exista el producto sin nombre todavia
if len(id_producto) is 0:
    #Cojemos la categoria ninguna para meter estos productos
    cr.execute("SELECT id FROM product_category WHERE name = 'ninguna'")
    id_categ = cr.fetchall()[0][0]
    #Y creamos el producto con el precio correspondiente
    id_producto = self.pool.get('product.product').create(cr, uid, {
        'name':vendedores[i][z][1],
        'list_price':vendedores[i][z][3],
        'balanza_id':balanza_id,
        'default_code':vendedores[i][z][1],
        'uom_id':2,
        'uom_po_id':2,
        'categ_id':id_categ,
        'list_price':vendedores[i][z][3]
    })

#Ya lo tenemos todo para el caso de productos sin nombre
self.pool.get('stock.move').create(cr, uid, {
    'product_id':id_producto,
    'name':'producto sin nombre',
    'picking_id':id_albaran,
    'product_uom':2,
    'location_id':10,
    'location_dest_id':10,
    'product_qty':vendedores[i][z][2],
})
```

- **calcular\_caja**: este método se encarga de realizar el arqueo de caja al finalizar la jornada laboral. El usuario podrá consultar la cantidad en euros que debe haber en la caja. Para todo esto nos apoyaremos en el método *calcular\_total\_tiquets* que nos proporciona el total de líneas de venta que ha habido durante el día, simplemente recorre cada línea y

va sumando el total de cada tique. Con lo cual obtenemos la cantidad de dinero que debe haber en la caja al final del día. Hay que prestar atención a los abonos para restar el valor a `caja_final` de forma que se refleje la realidad, además de tener en cuenta el valor de `caja_inicial` que es el dinero con el que la caja empieza el día.

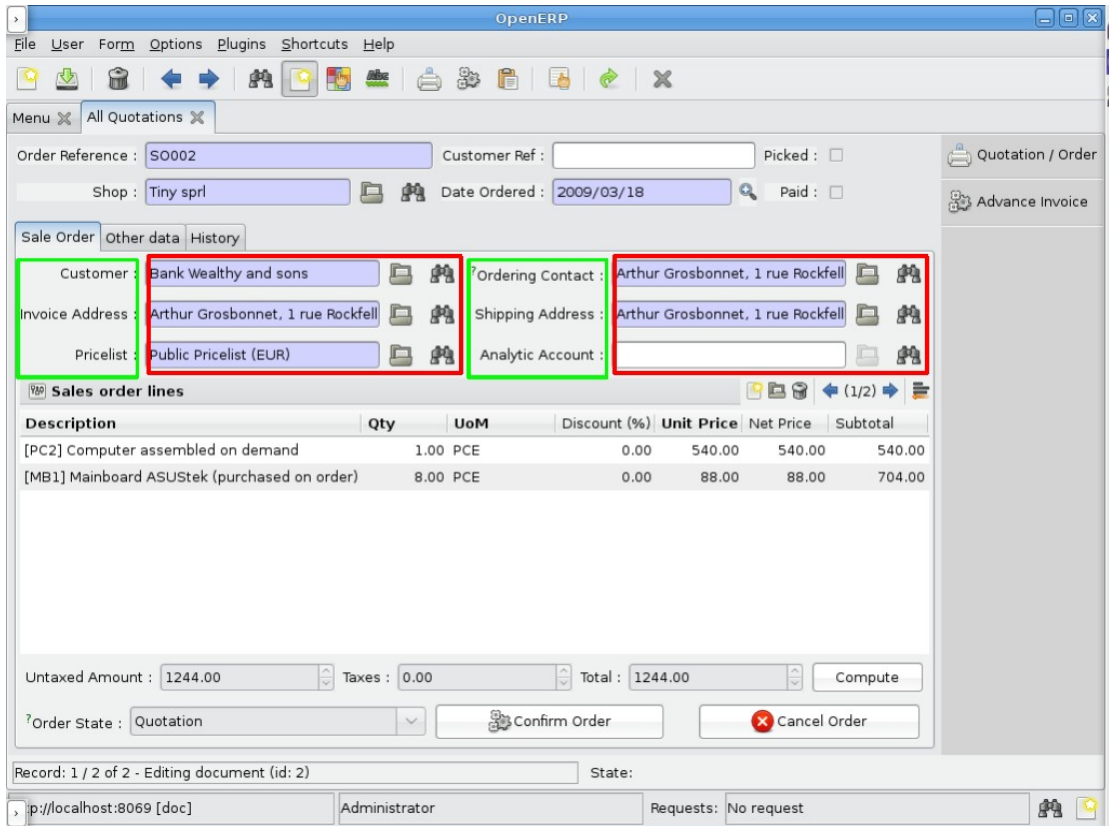
```
for i in range(len(vendedores)):
    for z in range(len(vendedores[i])):
        if cmp(vendedores[i][z][2], u'-E') is 0 or str(vendedores[i][z]
[2]).startswith(u'-P'):
            continue
        caja_final += float(vendedores[i][z][4])

caja_final += caja_inicial
caja_final = round(caja_final, 2)
```

## 2.- La vista

La forma en la que los objetos se muestran al usuario es a través de XML. Cada uno de los objetos de negocio tiene su propia vista escrita en XML y puede ser modificada en tiempo de ejecución mostrándose los cambios inmediatamente. Cada uno de los objetos tiene dos tipos de vista, la de formulario y la de árbol. La primera, la de formulario, es para que el usuario modifique o vea todos los datos del objeto, mientras que la segunda es para poder consultar la lista de objetos creados.

La forma de mostrar cada campo en pantalla es de derecha a izquierda, y cada campo va acompañado de su etiqueta por defecto. La distribución de campos se realiza de la siguiente manera:



Cada una de las líneas está dividida en cuatro columnas que pueden contener tanto un campo de edición como una etiqueta. Toda la vista es configurable, hay campos que ocupan las ocho columnas al completo, como las tablas de objetos, y podemos hacer que un campo ocupe tantas columnas como deseemos indicándolo en el archivo XML de la vista.



The screenshot shows the OpenERP interface for a Sales Order. The order reference is SO001, customer is Agrolait, and the date ordered is 2009/03/18. The sales order lines table is highlighted with a blue border.

Description	Qty	UoM	Discount (%)	Unit Price	Net Price	Subtotal
New server config + material	1.00	PCE	0.00	123.00	123.00	123.00
[PC1] Basic PC	3.00	PCE	0.00	450.00	450.00	1,350.00
[PC1] Basic PC	3.00	PCE	0.00	450.00	450.00	1,350.00
[MB1] Mainboard ASUStek A7N8X	5.00	PCE	0.00	88.00	88.00	440.00

Summary: Untaxed Amount: 3263.00, Taxes: 0.00, Total: 3263.00. Order State: Quotation. Buttons: Confirm Order, Cancel Order.

Los archivos que definen las vistas tienen el siguiente formato:

```
<?xml version="1.0"?>
<openerp>
  <data>
    [definición de las vistas]
  </data>
</openerp>
```

Tenemos tres tipos principales de definiciones de las vistas, estos tipos son los siguientes:



- Etiquetas con el atributo de vista 'ir.ui.view' que contienen la propia definición de la vista. Por ejemplo:

```
<record model="ir.ui.view" id="balanzas_vendedor_tree_view">
  <field name="name">balanzas.vendedor.tree</field>
  <field name="model">balanzas.vendedor</field>
  <field name="type">tree</field>
  <field name="arch" type="xml">
    <tree string="Vendedores">
      <field name="name"/>
    </tree>
  </field>
</record>
```

- Etiquetas con el atributo "ir.actions.act\_window", que conecta acciones con vistas.

```
<record model="ir.actions.act_window" id="action_balanzas_vista_tree_view">

  <field name="name">Vista Global</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">balanzas.vista</field>
  <field name="view_type">form</field>
  <field name="view_mode">form</field>

</record>
```

- Etiquetas que crean las entradas de menú para acceder a las vistas:

```
<menuitem id="menu_balanzas" name="Balanzas"/>
<menuitem id="menu_balanzas_configuracion" name="Configuración"
parent="balanzas.menu_balanzas" sequence="1"/>
```



Además disponemos de etiquetas para personalizar las vistas, agrupar elementos, separar elementos, etc.

- **Separator:** sirve para añadir una línea que separe los elementos. Con el atributo "string" le indicamos la etiqueta del separador mientras que con el atributo "colspan" indicamos el número de columnas que ocupará el separador.

```
<separator string="Datos balanza" colspan="4"/>
```

- **Notebook:** podemos utilizar esta etiqueta para crear pestañas en las vistas. Así es posible distribuir la información en pantalla de una forma más efectiva, además de disponer de más sitio para repartirla. Con el atributo tabpos podemos indicar si las pestañas estarán arriba, abajo, a la derecha o a la izquierda.

```
<notebook colspan="4">...</notebook>
```

- **Group:** agrupa columnas y separa el grupo en tantas columnas como se desee. Esta etiqueta dispone de los siguientes atributos:
  - **colspan:** es el número de columnas a utilizar.
  - **Rowspan:** es el número de filas a utilizar.
  - **Col:** es el número de columnas en las que se dispondrán los hijos.

- **String:** se dibujará un marco alrededor de los campos con este nombre.

```
<group colspan="4" rowspan="10">
  <separator string="Datos balanza" colspan="4"/>
  <field name="marca"/>
  <field name="modelo"/>
  <field name="version"/>
  <field name="caja_inicial"/>
  <newline/>

  <group colspan="2" >
    <separator string="Inicialización" colspan="1"/>
    <newline/>
    <button name="obtener_vendedores" string="Obtener vendedores"
      type="object" icon="gtk-execute"/>
    <button name="obtener_productos" string="Obtener productos"
      type="object" icon="gtk-execute"/>

  </group>
  <newline/>
  <separator colspan="6" string="Vendedores"/>
  <field name="vendedores_ids" nolabel="1" colspan="4"
    widget="one2many_list" />
  <newline/>
  <separator colspan="6" string="Productos"/>
  <field name="productos_ids" nolabel="1" colspan="4"
    widget="one2many_list"/>
</group>
```

- **Widget:** podemos cambiar la forma en la que el campo se muestra. OpenERP dispone de los siguientes widgets para poder mostrar la información de una u otra forma:

- one2many\_list
- many2one\_list
- many2many
- url
- email



- image
  - float\_time
  - reference
  - one2one\_list
- **default\_focus:** Con el valor puesto en "1" permite colocar el foco en este campo al abrir el formulario. Lógicamente tan sólo un campo puede tener este atributo a "1".

Atributos avanzados más relacionados en el funcionamiento que en la vista en sí son los siguientes:

- **on\_change:** llama a la función indicada al cambiar el contenido de un campo.
- **Attrs:** hace que los atributos de un campo dependan de otro campo. El formato es el siguiente: "{`atributo': [(`nombre\_del\_campo`,`operador`,`valor`), (`nombre\_del\_campo`,`operador`,`valor`)],`atributo2': [(`nombre\_del\_campo`,`operador`,`valor`),]}"

```
<field digitos="(14, 3)" name="volumen" attrs="{`readonly': [(`tipo`,`=`,`servicio`)]}" />
```

- **eval:** permite evaluar un atributo como si éste fuese código en Python. Podremos entonces definir valores que no queremos que sean tratados como cadenas de caracteres.



## 4.- La programación del módulo

```
<field name="valor" eval="2.3" />
```

- **Page:** dentro de una etiqueta "notebook" define una nueva pestaña. Con el atributo "string" definiremos el nombre de esta pestaña.

```
<page string="Balanza"> ... </page>:
```

- **Button:** añade un botón al formulario. Este botón puede realizar diversas acciones. El botón es un elemento de diseño muy importante para nuestro módulo, ya que mediante estos botones llamaremos a los métodos creados.

```
<button name="obtener_vendedores" string="Obtener vendedores" type="object" icon="gtk-execute"/>
```

- **Label:** añade una etiqueta al formulario.

```
<label string="Texto"/>
```

- **Newline:** fuerza la inserción de una nueva línea aunque la línea actual no esté completamente llena.

```
<newline/>
```

Por último, veamos los atributos que puede tener un campo:

- **select:** con el valor en "1" este campo se incluirá en la búsqueda básica, con el valor puesto en "2" se incluirá en la búsqueda avanzada.



- **Colspan:** con este atributo le indicaremos a un campo el número de columnas que ocupa, de forma que el campo se extenderá hasta abarcarlas todas.
- **Readonly:** si ponemos este atributo en "1", el campo no será editable. Es útil para utilizarlo en campos calculados dinámicamente y que no queremos que sean modificados por el usuario.
- **Required:** indicamos si queremos que el campo sea obligatorio. El objeto no se podrá guardar hasta que todos los campos con el atributo "required" activado (con el valor "1") estén rellenos.
- **Nolabel:** podemos eliminar la etiqueta de este campo con el atributo "nolabel" activado en "1".
- **Invisible:** colocando este valor a "True" ocultaremos tanto la etiqueta como el campo en sí.
- **Password:** si ponemos este campo en "True" los caracteres introducidos se verán como asteriscos. Se utiliza para introducir contraseñas.
- **String:** con este atributo podremos cambiar el nombre de la etiqueta del campo.
- **Domain:** podremos restringir el dominio de este campo, es decir, los valores que podemos introducir en el mismo con este atributo. Un ejemplo podría ser el valor "[('partner\_id','=',partner\_id)]".

Una vez hemos revisado los elementos básicos del diseño ya podemos crear las vistas. Básicamente una vista completa dispone de una entrada en el menú y de una acción que conectará la entrada

visible en el menú con su correspondiente formulario. Así, y siguiendo el estándar que utiliza OpenERP, colocaremos nuestras vistas en un archivo cuyo nombre sigue el formato "nombredelmodulo\_view.xml", en nuestro caso "balanzas\_view.xml. Pasaremos a repasar los puntos básicos de nuestras vistas y a comprobar los resultados obtenidos:

– **objeto *balanzas\_balanza***: en esta vista es donde realizamos la inicialización de los productos y los vendedores. También daremos información de la balanza. La información correspondiente a la balanza será introducida a mano y los vendedores y los productos serán obtenidos gracias a los métodos *obtener\_vendedores* y *obtener\_productos*. Estos métodos serán lanzados con dos elementos "button". Además añadiremos dos tablas para que el usuario pueda comprobar si los productos y los vendedores se han insertado correctamente en OpenERP. Éstas son las partes principales de la vista:

– la definición de la vista

```
<record model="ir.ui.view" id="balanzas_balanza_view">
  <field name="name">balanzas.balanza.form</field>
  <field name="model">balanzas.balanza</field>
  <field name="type">form</field>
  <field name="priority" eval="5"/>
  <field name="arch" type="xml">
</record>
```

– la definición de los elementos principales

```
<form string="Balanzas">
  <group colspan="4">
    <field name="name"/>
    <field name="puerto"/>
    <field name="padre"/>
  </group>
  <newline/>
  <group colspan="4" rowspan="10">
    <separator string="Datos balanza" colspan="4"/>
    <field name="marca"/>
    <field name="modelo"/>
```





## 4.- La programación del módulo

```
<field name="version"/>
<field name="caja_inicial"/>
<newline/>
<group colspan="2" >
  <separator string="Inicialización" colspan="1"/>
  <newline/>
  <button name="obtener_vendedores" string="Obtener vendedores"
type="object" icon="gtk-execute"/>
  <button name="obtener_productos" string="Obtener productos"
type="object" icon="gtk-execute"/>

</group>
<newline/>
<separator colspan="6" string="Vendedores"/>
```

– Y así quedaría la vista:

OpenERP

Archivo Usuario Formulario Opciones Conectores Atajos Ayuda

Nuevo Guardar Borrar Atrás Adelante Lista Formulario Calendario Gráfico Imprimir

Menú Balanza

Dirección IP/Nombre de dominio :  Puerto :

Principal

**Datos balanza**

Marca :  Modelo :

Versión :  Caja inicial :

**Inicialización**

**Vendedores**

**Vendedores**    (1/5)

Nombre

Efrain David

Miguel

**Productos**

**Productos**    (1/51)

Código	Nombre	Variantes	UdM por defecto	Stock real	Stock virtual	Precio lista	Precio cos
1	aliñadas extremeñas		Kg.	0,00	0,00	4,80	1,
?	marida de Aragón		Kg.	0,00	0,00	4,80	1,

Registro: 1 / 1 de 1 - Editando documento (id: 1) Estado:

socket://localhost:8070 [goandre2] javi Solicitudes: Sin solicitudes

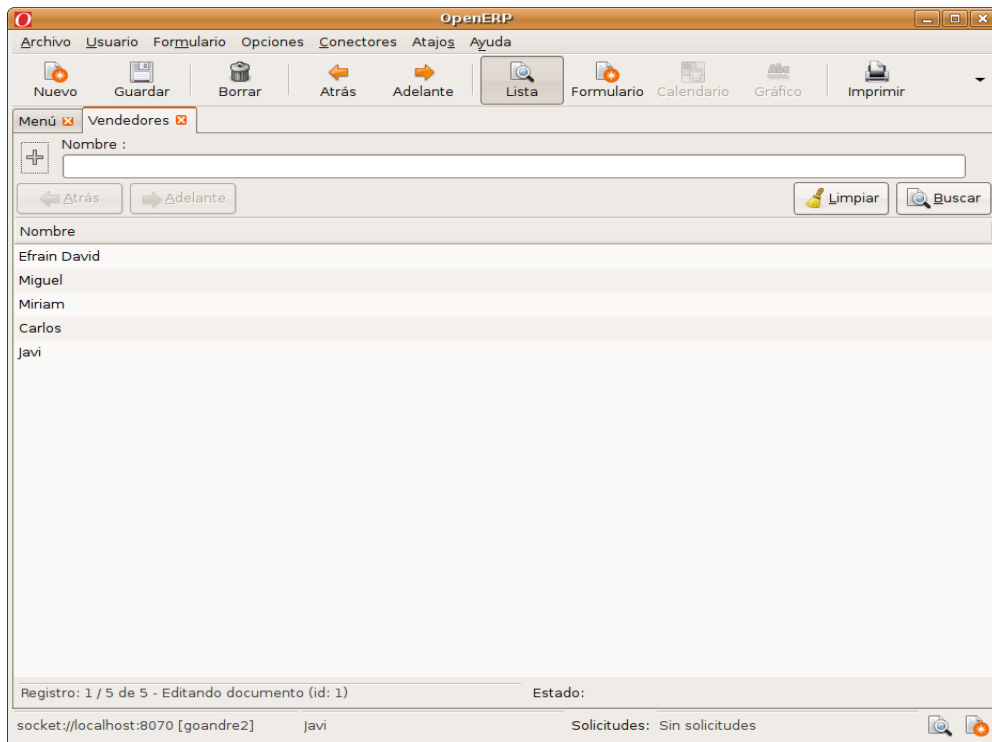
Ahora ya sólo nos queda crear una acción para poder conectar esta vista a la entrada del menú correspondiente por medio de su identificador:

```
<record model="ir.actions.act_window"
id="action_balanzas_balanza_tree_view">
  <field name="name">Balanza</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">balanzas.balanza</field>
  <field name="view_type">form</field>
  <field name="view_mode">tree,form</field>
</record>
```

- **balanzas.vendedor:** en esta vista simplemente mostraremos la lista de vendedores previamente insertados en la balanza en forma de árbol. Las partes principales que definen esta vista son:

```
<record model="ir.ui.view" id="balanzas_vendedor_tree_view">
  <field name="name">balanzas.vendedor.tree</field>
  <field name="model">balanzas.vendedor</field>
  <field name="type">tree</field>
  <field name="arch" type="xml">
    <tree string="Vendedores">
      <field name="name"/>
    </tree>
  </field>
</record>
```

La vista quedaría como sigue:



Y ésta es la acción que posteriormente conectaremos con la entrada del menú de OpenERP:

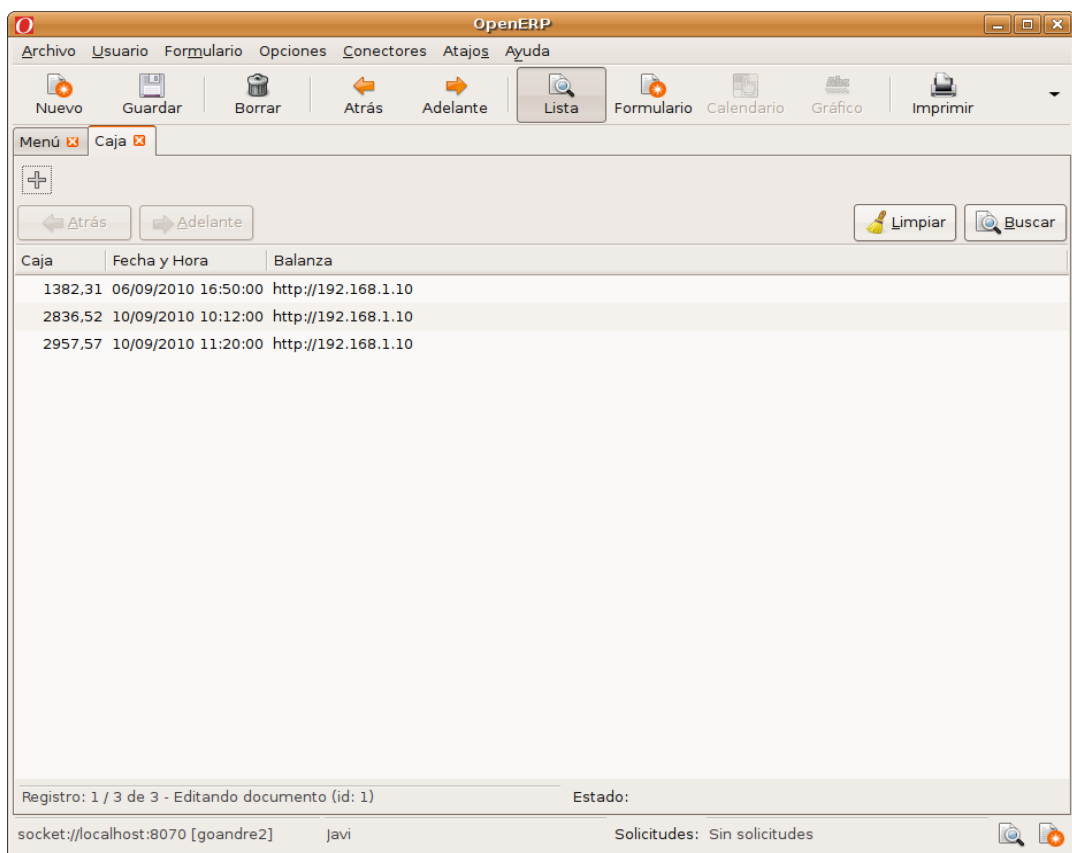
```
<record model="ir.actions.act_window"
id="action_balanzas_vendedor_tree_view"          <field
name="name">Vendedores</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">balanzas.vendedor</field>
  <field name="view_type">form</field>
  <field name="view_mode">tree,form</field>
</record>
```

- **balanzas.caja:** esta vista es muy parecida a balanzas.vendedor. En ella mostraremos al usuario una lista con los cierres de caja y los días correspondientes a tales cierres. Las principales partes de esta vista son:

```
<record model="ir.ui.view" id="balanzas_caja_tree_view">
  <field name="name">balanzas.caja.tree</field>
  <field name="model">balanzas.caja</field>
```

```
<field name="type">tree</field>
<field name="arch" type="xml">
  <tree string="Caja">
    <field name="caja"/>
    <field name="date"/>
    <field name="balanza_id"/>
  </tree>
</field>
</record>
```

La vista quedaría de la forma siguiente:



Y el conector que enlaza la vista con el menú es como sigue:

```
<record model="ir.actions.act_window" id="action_balanzas_balanza_tree_view">
  <field name="name">Balanza</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">balanzas.balanza</field>
  <!-- <field name="domain">[('field_name','condition',criteria)]</field-->
```



## 4.- La programación del módulo

```
<field name="view_type">form</field>  
<field name="view_mode">tree,form</field>
```

```
</record>
```

- **balanzas.ventas\_diarias:** Esta vista mostrará toda la información de las ventas que realiza cada vendedor diariamente. Al igual que nos muestra la interfaz web de la balanza que en este problema estamos tratando. Las partes más importantes de esta vista son:

```
<record model="ir.ui.view" id="balanzas_ventas_diarias_tree_view">  
  <field name="name">balanzas.ventas_diarias.tree</field>  
  <field name="model">balanzas.ventas_diarias</field>  
  <field name="type">tree</field>  
  <field name="arch" type="xml">  
  
    <tree string="Ventas diarias">  
      <field name="vendedor_id"/>  
      <field name="total"/>  
      <field name="abonos"/>  
      <field name="neto"/>  
      <field name="acumulado"/>  
      <field name="date"/>  
      <field name="balanza_id"/>  
    </tree>  
  
  </field>  
</record>
```

Y la vista queda como sigue:

The screenshot shows the OpenERP interface with a menu set to 'Ventas diarias'. The table displays the following data:

Vendedor	Total	Abonos	Neto	Acumulado	Fecha	Balanza
Efrain David	0,00	0,00	0,00	0,00	11/08/2010	http://192.168.1.10
Miguel	193,77	0,00	193,77	0,00	11/08/2010	http://192.168.1.10
Miriam	0,00	0,00	0,00	0,00	11/08/2010	http://192.168.1.10
Carlos	29,43	0,00	29,43	0,00	11/08/2010	http://192.168.1.10
Javi	0,00	0,00	0,00	0,00	11/08/2010	http://192.168.1.10
Efrain David	0,00	0,00	0,00	0,00	10/08/2010	http://192.168.1.10
Miguel	431,08	2,40	428,68	0,00	10/08/2010	http://192.168.1.10
Miriam	0,00	0,00	0,00	0,00	10/08/2010	http://192.168.1.10
Carlos	0,00	0,00	0,00	0,00	10/08/2010	http://192.168.1.10
Javi	0,00	0,00	0,00	0,00	10/08/2010	http://192.168.1.10
Efrain David	0,00	0,00	0,00	0,00	09/08/2010	http://192.168.1.10
Miguel	226,17	0,00	226,17	0,00	09/08/2010	http://192.168.1.10
Miriam	51,23	0,00	51,23	0,00	09/08/2010	http://192.168.1.10
Carlos	0,00	0,00	0,00	0,00	09/08/2010	http://192.168.1.10
Javi	0,00	0,00	0,00	0,00	09/08/2010	http://192.168.1.10
Efrain David	0,00	0,00	0,00	0,00	07/08/2010	http://192.168.1.10
Miguel	351,15	0,00	351,15	0,00	07/08/2010	http://192.168.1.10
Miriam	0,00	0,00	0,00	0,00	07/08/2010	http://192.168.1.10

Ahora veremos el código XML del conector:

```
<record model="ir.actions.act_window"
id="action_balanzas_ventas_diarias_tree_view">
    <field name="name">Ventas diarias</field>
    <field name="type">ir.actions.act_window</field>
    <field name="res_model">balanzas.ventas_diarias</field>
    <field name="view_type">form</field>
    <field name="view_mode">tree,form</field>
</record>
```

- **balanzas.vista:** esta vista no corresponde a ningún patrón para el correcto funcionamiento del programa, sino más bien a una petición expresa por parte del cliente. Se requiere que el gran total diario, el arqueo de caja y la creación de albaranes diarios de ventas por vendedor se realicen desde la misma página del programa.



#### 4.- La programación del módulo

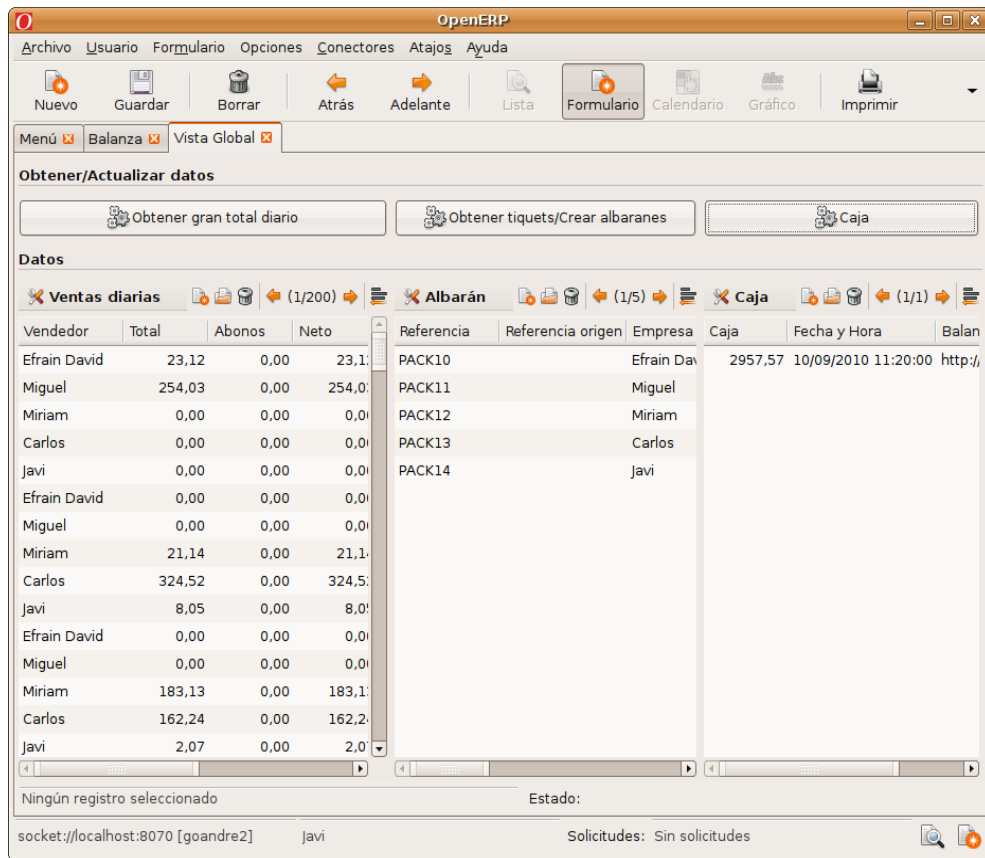
Actualmente OpenERP no proporciona una forma nativa para realizar este tipo de vistas, así que fue necesario implementar esta vista como si de un objeto de negocio de OpenERP se tratase. Creamos las relaciones correspondientes entre los elementos a mostrar y la vista y se utilizó la opción “\_auto = False” de OpenERP para que la información de este objeto no se guardara en la base de datos, ya que tan sólo utilizaremos este objeto con el fin de mostrar datos agrupados y, además, poder modificarlos. Los elementos principales que componen la vista son:

```
<form string="Ventas totales">

    separator colspan="6" string="Obtener/Actualizar datos"/>
    <button name="obtener_ventas_diarias" string="Obtener gran total
diario" type="object" icon="gtk-execute"/>
    <button name="obtener_tiquets" string="Obtener tiquets/Crear albaranes"
type="object" icon="gtk-execute"/>
    <button name="calcular_caja" string="Caja" type="object" icon="gtk-
execute"/>
    <separator colspan="6" string="Datos"/>
    <field name="ventas_diarias_ids" nolabel="1"/>
    <field name="albaranes_ids" nolabel="1"/>
    <field name="caja_ids" nolabel="1"/>

</form>
```

Y el resultado final de la vista es el que se muestra a continuación:



El conector es como se muestra a continuación:

```
<record model="ir.actions.act_window" id="action_balanzas_vista_tree_view">
  <field name="name">Vista Global</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">balanzas.vista</field>
  <field name="view_type">form</field>
  <field name="view_mode">form</field>
</record>
```

Llegados a este punto tan solo nos queda crear las diferentes entradas de menú que hemos conectado mediante las acciones a las vistas. En el siguiente ejemplo creamos el menú principal balanza y le asociamos la sección de configuración:

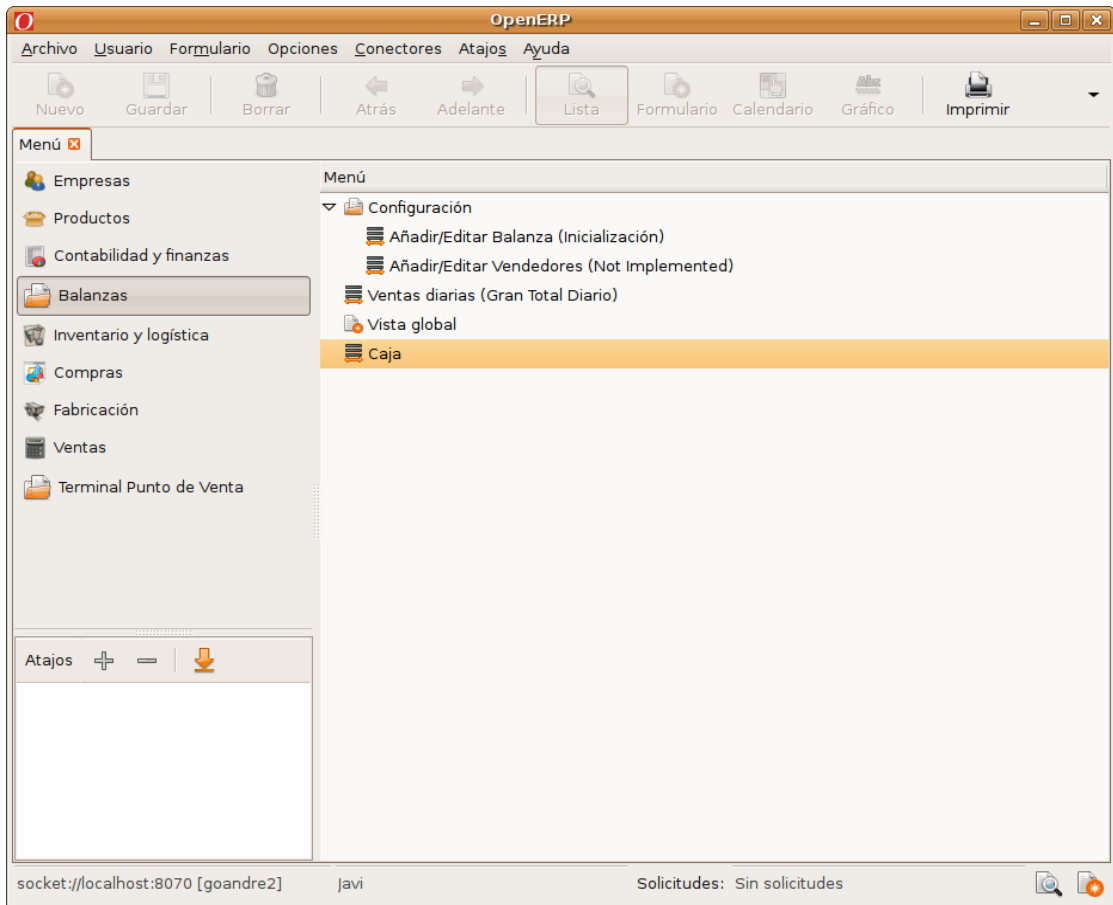
```
<menuitem id="menu_balanzas" name="Balanzas"/>
<menuitem id="menu_balanzas_configuracion" name="Configuración"
```





## 4.- La programación del módulo

```
parent="balanzas.menu_balanzas" sequence="1"/>
```



Ya tenemos la vista completamente preparada para que el usuario interactúe con ella, tan sólo queda crear el controlador que obtendrá los datos de las balanzas dependiendo de su marca, modelo y de la versión de la interfaz (web en este caso).



### 3.- El controlador

El controlador es una parte fundamental del módulo que estamos programando, ya que se encargará de obtener y formatear los datos, comprobando que todo esté correcto y arreglando cualquier problema que haya. Una vez realizado este trabajo estos datos se devolverán al programa principal del módulo para que los convierta definitivamente en objetos de negocio de OpenERP. Todo este trabajo debe devolver un formato determinado de datos de forma totalmente transparente al tipo de balanza que se encuentre en la tienda del cliente.

Para la realización de este trabajo se contaba con tres balanzas fabricadas y programadas por la empresa MAPAL. Las tres balanzas están interconectadas mediante un sistema unificado de gestión programado por MAPAL que permite gestionar las tres balanzas como si se tratasen de una sólo, intercambiando todos los datos de ventas, artículos, familias, etc. Esto quiere decir que desde el punto de vista de nuestra aplicación tan sólo hay una balanza, una caja y todas las ventas y vendedores están en esa misma balanza. De modo que no podemos extraer por el momento información de qué balanza ha realizado cada venta sino simplemente el total de las mismas. Por una parte el usuario no demanda más que los totales globales de la tienda y, por tanto, no es necesaria más información. Por el otro esperamos en posteriores versiones poder liberarnos de esta limitación trabajando conjuntamente con los desarrolladores de la interfaz de la balanza.

Al estar trabajando con una página web necesitaremos un analizador de código HTML (comúnmente llamado *parser*). Como no podía ser de otra forma la comunidad de Python ya ha desarrollado un excelente módulo



## 4.- La programación del módulo

que realiza esta función, este módulo es llamado "BeautifulSoup". Para instalar este módulo de Python utilizaremos el sistema de administración de paquetes que lleva incorporado Ubuntu:

```
$sudo aptitude install python-beautifulsoup
```

Gracias a "BeautifulSoup" podremos extraer los datos necesarios de cada página web con unas pocas líneas de código.

```
direccion = direccion + "/pag235.htm"
pagina = urllib2.urlopen(direccion).read()
pagina = BeautifulSoup(pagina, convertEntities='html')
tabla_vendedores = pagina("td")
vendedores = [entry.string for entry in tabla_vendedores if
hasattr(entry, 'string')]
```

Lo anterior es un ejemplo de como extraer los datos de los vendedores que hay dados de alta en la balanza.





Para poder rellenar cada una de las tablas necesitamos los siguientes métodos que estarán en el archivo de métodos separados que llamaremos "web.py".

El archivo "web.py" es un controlador unificado de balanzas, de forma que extraerá los datos de cualquier marca y modelo de balanza y los presentará al módulo de balanzas en el mismo formato sea cual sea el formato original. Así, "web.py" tiene una serie de métodos que extraerán de las balanzas la información indicada. Estos métodos son: obtener\_vendedores, obtener\_productos, obtener\_gran\_total\_diario y obtener\_tiquets. Cada uno de estos métodos recibe al ser llamado tres parámetros: la url, el modelo y la marca. Cada método realiza las acciones necesarias para hablar con la interfaz de esa marca y modelo en concreto y devolverá siempre los datos al método que lo ha llamado (que será siempre un método del archivo "balanzas.py") en un formato determinado.

Actualmente tan sólo está implementado el controlador de una balanza, ya que no tenemos acceso por el momento a más balanzas diferentes. Sin embargo las bases están sentadas para poder gestionar mediante operadores "case" o "if else" todos los tipos de balanzas que se deseen. El único requisito es devolver los datos en el formato que espera recibirlos el programa que está llamando al método.

A continuación podemos ver un ejemplo de uso: [producto, familia]

```
productos = [[u'plátanos', u'fruta'], [u'pera', u'fruta'], [u'pepino',  
u'verdura'], [u'tomate', u'verdura'], [u'creïlles', u'verdura'],  
[u'ceba', u'verdura']]
```

## Conclusiones

Una vez realizado el desarrollo del módulo de balanzas para OpenERP estamos en condiciones de opinar sobre las tecnologías que hemos utilizado para llevar a cabo este proyecto en el que, además de haber aprendido un lenguaje de programación y el funcionamiento de un ERP completo como es OpenERP, tenemos la satisfacción de verlo funcionar en producción con los resultados esperados.

En primer lugar empezaremos hablando de nuestras impresiones con el lenguaje de programación utilizado en el proyecto. En Python encontramos un lenguaje eficaz para la programación ágil de aplicaciones. Además de ser apto para programas de considerable tamaño e incluso de gran demanda. Resulta sorprendente su velocidad de ejecución a pesar de ser un lenguaje interpretado. Su curva de aprendizaje es realmente corta frente a otros lenguajes similares, un programador con una mínima experiencia en POO puede dominarlo en muy poco tiempo.

Por otro lado está la agrupación de tecnologías que sientan el entorno del programa OpenERP. Tenemos ante nosotros un ERP con unas enormes posibilidades. Encontramos una gran carencia en las opciones de programación de la interfaz pero a cambio está a nuestra disposición una integración perfecta con un lenguaje muy potente que nos permite desarrollar módulos en tiempo récord. Además cuenta con una comunidad muy activa que prácticamente ya ha desarrollado módulos para casi todas las necesidades comunes en las PYME españolas y gran cantidad de módulos muy específicos.

En cuanto a las balanzas. El módulo cumple su función. Es obvio que en posteriores versiones hay que ir avanzando en varios aspectos como puede ser el intercambio de datos bidireccional y la posibilidad de ofrecer una gestión multitienda real. Esperemos también que posteriores versiones de OpenERP nos ofrezcan desarrollar una interfaz mucho más ergonómica ('Selects' dependientes para la elección de marca, modelo y versión de la balanza por ejemplo). La interfaz web que nos ofrece la balanza, debido a las limitaciones inherentes de los recursos que tiene la máquina y a su sistema de programación y gestión de memoria, es útil pero insuficiente en algunos aspectos. Además de añadir dificultad a la obtención de datos, hemos estado intentando que la balanza sea capaz de ofrecer los datos en XML, pero la falta de memoria de ésta hace imposible por el momento esta opción.

Este trabajo nos ha permitido descubrir un ERP que podemos ofrecer a las PYMES a un coste razonable permitiéndonos centrarnos en la instalación y formación de usuarios de la aplicación. Dejando de lado, la mayoría de las veces, los altos costes en desarrollo que suelen acarrear otros ERP más complejos y que, en realidad, no aportan una utilidad mayor que la de éste.



## Bibliografía





## GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2008 Free Software Foundation, Inc.

<<http://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License



principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section



does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats



include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has



no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other



respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

#### **4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing



distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.



- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.





- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.



The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".



## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole



aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your



license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## **10. FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free



Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.



The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.