UNIVERSIDAD POLITECNICA DE VALENCIA

PROYECTO FINAL DE CARRERA

# Modelo educacional dirigido al aprendizaje de las herramientas de administración del software OpenbravoERP


**Presentado por:**

Jose Vicente Lladró Giner

**Dirigido por:**

Andrés Boza


Valencia, 2010

Dedico este proyecto a mi padre, que me ha enseñado a superar cualquier bache que haya en mi camino, a toda mi familia porque siempre estará ahí para todo lo que necesite y a mis amigos, en especial a Javi y Thanys por motivarme en los momentos que más lo necesitaba.

## Tabla de contenido


**1.- Introducción** **7**
*1.1.- Presentación* *7*
*1.2.- Objetivos* *7*
*1.3.- Estructura del proyecto* *8*

**2.- Los sistemas ERP** **9**
*2.1.- El ERP, la solución a una necesidad* *9*
2.1.1.- ¿Qué es un ERP? 9
2.1.2.- Historia 10
2.1.3.- Características 11
2.1.4.- Clasificación 12
*2.2.- ERPs de código abierto* *16*
2.2.1.- Compiere 16
2.2.2.- Fisterra 16
2.2.3.- FacturaLUX 17
2.2.4.- OpenERP 17
2.2.5.- OpenBluelab 18
2.2.6.- Project Open 18
2.2.7.- OpenXpertya 18
2.2.8.- OpenbravoERP 19
*2.3.- Implantación de un ERP* *20*
2.3.1.- Ventajas 20
2.3.2.- Inconvenientes 21
*2.4.- OpenbravoERP* *22*
2.4.1.- La evolución a la Web 2.0 22
2.4.2.- Cambio del modelo de negocio 23
2.4.3.- Arquitectura en tres capas 24
2.4.4.- ¿Porqué OpenbravoERP? 26
*2.5.- Conclusión* *26*

**3.- Instalación de OpenbravoERP** **27**
*3.1.- Pre-requisitos de la instalación* *27*
3.1.1.- Tipos de instalación 27
3.1.2.- Topologías permitidas 28
3.1.3.- Instalación elegida 28
*3.2.- Instalación del Stack o Pila Tecnológica* *28*
3.2.1.- Java JDK 28
3.2.2.- Eclipse 30
3.2.3.- PostgreSQL 30
3.2.4.- Apache Ant 31
3.2.5.- Apache Tomcat 33
3.2.6.- Instalación de OpenbravoERP 34

**4.- Estudio de los modelos de implementación** **41**
*4.1.- Modelo de Desarrollo Dirigido* *41*
4.1.1.- Implementación del MDD 41
4.1.2.- Diccionario de Aplicaciones 42
4.1.3.- Modelo-Vista-Controlador 43
*4.2.- Modularidad* *44*
*4.3.- Métodos de desarrollo* *44*

**5.- Revisión del esquema de la Base de Datos** **47**

*5.1.- Estándar de codificación* 47
*5.2.- Arquitectura REST* 48
*5.3.- Capa de Acceso a Datos* 49
*5.4.- Conexión a la Base de Datos* 50
*5.5.- Diagramas Entidad-Relación* 51
5.5.1.- Introducción 51
5.5.2.- Elaboración de los diagramas E-R 52
5.5.3.- Modelo de base de datos 54
5.5.4.- Modelo de Entidad 54
*5.6.- Conclusiones* 55

**6.- Análisis del módulo para la ampliación de funcionalidades** **56**
*6.1.- Diccionario de la Aplicación* 57
6.1.1.- Módulo 57
6.1.2.- Tablas y columnas 61
6.1.3.- Conjunto de datos 64
6.1.4.- Ventanas, pestañas y campos 67
6.1.5.- Referencia 72
6.1.6.- Informes y procesos 76
6.1.7.- Formulario 82
6.1.8.- Mensaje 85
6.1.9.- Texto interfaces 87
6.1.10.- Mapeo de Implementación del Diccionario 88
6.1.11.- Actualizar infraestructura de histórico de auditoría 91
6.1.12.- Sincronizar términos 92
*6.2.- Configuración* 93
6.2.1.- Elemento 93
6.2.2.- Tipo agrupación campos 95
6.2.3.- Inputs auxiliares 96
6.2.4.- Callout 97
6.2.5.- Reglas de validación 99
6.2.6.- Mes 100
6.2.7.- Puntos de Extensión 101
*6.3.- Mantenimiento* 103
6.3.1.- Prueba 103
6.3.2.- Actualiza Identificadores tablas 104
6.3.3.- Recompilar objetos DB 105
6.3.4.- Consulta SQL 105
*6.4.- Test Automático* 106
6.4.1.- Introducción 106
6.4.2.- Creación 107

**7.- Desarrollo de prácticas docentes para incluir nuevas funcionalidades a OpenbravoERP** **109**
*7.1.- Práctica 1: Implantación de OpenbravoERP* 109
7.1.1.- Objetivo 109
7.1.2.- Resumen 109
7.1.3.- Introducción 109
7.1.4.- Desarrollo 110
7.1.5.- Revisión global 127
*7.2.- Práctica 2: Ampliando la funcionalidad de OpenbravoERP* 127
7.2.1.- Objetivo 127
7.2.2.- Resumen 127
7.2.3.- Introducción 127
7.2.4.- Desarrollo 128
7.2.5.- Revisión global 140

**8.- Conclusiones** **141**
*8.1.- Acercamiento a los objetivos planteados* *141*
*8.2.- Conclusión a nivel de funcionalidad* *141*
*8.3.- Conclusión del trabajo propio* *141*
*8.4.- Líneas de trabajo futuro* *142*

**Bibliografía** **143**

# 1.- Introducción

En este capítulo se presenta una introducción al proyecto final de carrera que consiste en el análisis de la herramienta OpenbravoERP. Se comenzará presentando el trabajo que se va a realizar. Seguidamente, se realizará una breve descripción de los objetivos planteados en al comienzo del proyecto y, finalmente, abarcaremos cuál será su estructura a lo largo de esta memoria.

## 1.1.- Presentación

Este proyecto final de carrera se enfoca en el análisis y estudio de la aplicación Web OpenbravoERP, que es un software para la Planificación de Recursos Empresariales (*Enterprise Resource Planing*). Openbravo ERP Community Edition es una aplicación de código abierto[1] enfocada a la gestión de pequeñas y medianas empresas.

Más concretamente estudiaremos la arquitectura de OpenbravoERP para conocer cómo estructura sus módulos o cómo esta organizada la base de datos. También indagaremos en el sistema que permite gestionar y ampliar su funcionalidad de cara a amoldar el Sistema ERP dependiendo de las necesidades de cada empresa en un entorno real.

Este proyecto hará uso de los conocimientos adquiridos a lo largo de la carrera de Ingenieria informática de asignaturas como: Arquitectura de Bases de Datos, Desarrollo Web, Creación de Documentos de Hipertexto o Informática del Sitema Productivo, Logístico y Comercial. Abarca dos especialidades de la carrera: Ingeniería del Diseño de Software y Sistemas de Información.

## 1.2.- Objetivos

El principal objetivo del presente proyecto es diseñar un modelo educacional para el aprendizaje de los recursos que proporciona OpenbravoERP para ampliar su funcionalidad. El enfoque tomado es totalmente técnico para que los desarrolladores puedan entender las tecnologías utilizadas en todo momento.

Del obejtivo principal se derivan los siguientes:

1. Conocer el funcionamiento interno de la aplicación Web OpenbravoERP. El estudio se realizará para poder realizar las modificaciones pertinentes a lo largo del proyecto.

2. Analizar su arquitectura, la base de datos y las herramientas que le dan soporte.

3. Diseñar una serie de actividades educacionales que guíen al usuario en el desarrollo de modificaciones en el ERP.

---

[1] Open Source o código abierto es el término por el que se conoce al software distribuido y desarrollado libremente

Los beneficios del proyecto van a ir dirigidos sobre todo hacia los usuarios del OpenbravoERP que quieran realizar algún cambio sobre la herramienta frente a unas necesidades concretas. También tendremos a la comunidad de usuarios y de desarrolladores que se beneficiaran de una forma indirecta al poder disfrutar de las nuevas funcionalidades desarrolladas. Sin embargo, también destacaremos el beneficio personal ya que la experiencia obtenida a partir del estudio de esta herramienta va a favorecer la mejora del conocimiento interno de este ERP.

## 1.3.- Estructura del proyecto

Esta memoria se estructura en 4 bloques:

1. **Base teórica**: está compuesto únicamente por el capítulo 2 de la memoria. Realiza una pequeña introducción de lo que es un ERP, su funcionalidad, la situación actual de su mercado y profundiza en las características de del ERP Openbravo.

2. **Arquitectura de OpenbravoERP**: está compuesto por los capítulos 3, 4 y 5. En este bloque se realizará un listado de los prerequisitos para dar soporte a la instalación del ERP. Se analizarán también la arquitectura con la que se sostenta el sistema y que nos ayudará a entender mejor su funcionamiento.

3. **Módulo de desarrollo**: es el bloque central de la memoria. Se compone del capítulo 6 y en él se describen todas las posibilidades que ofrece OpenbravoERP para ampliar su funcionalidad.

4. **Prácticas didácticas**: este último bloque, compuesto por el capítulo 7, se verán las dos prácticas didácticas propuestas para que los usuarios puedan aprender de forma interactiva todo lo aprendido a lo largo de la memoria. Las prácticas son “la implantación de OpenbravoERP” y “el desarrollo de modificaciones sobre la funcionalidad básica de OpenbravoERP”.

# 2.- Los sistemas ERP

En este apartado vamos a detallar el contexto y la motivación del proyecto. Estudiaremos la forma de abarcar esas necesidades mediante los Sitemas ERP. Averiguaremos qué son, para qué sirven y analizaremos su historia para conocer tanto su evolución como su situación actual.

Una vez conozcamos el entorno de los ERPs estaremos en situación de analizar Openbravo ERP, el sistema software con el que vamos a trabajar durante todo este proyecto.

## 2.1.- El ERP, la solución a una necesidad

A lo largo de todo este apartado vamos a ver que conforma un Sistema ERP y los requisitos que debe abarcar su funcionalidad para que se pueda considerar un ERP fiable.

Indagaremos también en la historia que envuelve a estos sistemas, su pasado, presente y futuro.

Y por último estudiaremos que tipos de ERPs se encuentran hoy en el mercado.

### 2.1.1.- ¿Qué es un ERP?

Un ERP consiste en una serie de aplicaciones software (un sistema) cuya función es la gestión de una organización. Pero vayamos a una definición más detallada.

“Los sistemas de planificación de recursos de la empresa (*Enterprise Resource Planing* o ERP) son sistemas de gestión de información que integran y automatizan muchas de las prácticas de negocio asociadas con los aspectos operativos o productivos de una empresa”. (Senn, 1990).

Esta definición da a conocer que una empresa genera diariamente datos; nuevos clientes, ventas, trabajadores, etc. Todos estos **datos** por si solos no significan nada más que números, pero si los procesamos de una cierta manera podemos llegar a obtener **información** muy valiosa para la empresa. Además, este sistema automatiza muchos de los procesos diarios, ahorrando coste y tiempo.

Sigamos con la definición, “el propósito fundamental de un ERP es otorgar apoyo a los clientes del negocio, tiempos rápidos de respuesta a sus problemas así como un eficiente manejo de información que permita la toma oportuna de decisiones y disminución de los costos totales de operación”.

Lo que viene a decir es que gran parte de las decisiones vitales de una empresa tanto a corto, mediano y largo plazo pueden verse condicionadas a la información procesada que se obtiene del ERP. Tanto la dirección de los distintos departamentos como la gerencia observan el estado actual de la empresa de una forma eficiente (rápida y veraz) y deciden cual es la estrategia más oportuna en cada momento.

"ERP representa un amplio espectro de funciones que intenta abarcar todas las entidades

de una empresa. Requiere de la profundidad organizacional y funcional de una gran variedad de empresas de manera que se pueda examinar y modificar un concepto de empresa único" (Gartner Group, 2009).

"La intensidad en la competencia global ha creado un ambiente volátil en los negocios, lo cual implica que las empresas se preocupen por:

- Tiempos de respuesta más rápidos en el desarrollo de nuevos productos y órdenes de entrega al cliente
- Satisfacción del cliente
- Diseño de productos y servicios personalizados
- Reducción en los costes
- Productos consistentes y órdenes y procesos de pago simplificados a clientes multinacionales". (Figueroa, 2009)

En conclusión, el propósito fundamental de un ERP es otorgar apoyo a los clientes del negocio, tiempos rápidos de respuesta a sus problemas así como un eficiente manejo de información que permita la toma oportuna de decisiones y disminución de los costos totales de operación.

### 2.1.2.- Historia

La historia de los ERP se remonta a la década de 1960, cuando los sistemas se enfocaban principalmente al control de inventario. Entonces, gracias al progreso tecnológico empezaron a surgir necesidades para la gestión empresarial. La mayoría de estos sistemas fueron diseñados para manejar el inventario con base a los conceptos tradicionales de inventario. Ya en 1970 se hizo un cambio de enfoque hacia el MRP (Material Requirement Planning). Este sistema ayudaba en la traducción del programa maestro de producción en los requisitos para las unidades individuales, tales como: componentes y materias primas, conjuntos de planificación y adquisición de otros materiales. Este sistema se dedicaba principalmente a la planificación de las necesidades de materia prima.

Luego, en 1980 se introdujo el concepto de MRP II (Manufacturing Resource Planning) que implicaba la optimización del proceso de producción. Aunque el MRP II, al principio era una extensión de MRP para incluir las ventas y la gestión de las actividades de distribución, durante los años posteriores, el MRP II se amplió para incluir áreas como Finanzas, Recursos Humanos, Ingeniería, Administración de Proyectos, etc. Esto dio a luz al ERP, que abarca la coordinación entre funciones y su integración para la mejora del proceso de producción.

“A finales de los 90 el incremento de la competitividad global combinado con la evolución del mercado y de la tecnología, causaron que muchas empresas reinventaran sus productos y servicios, incluido su estructura organizacional y otros controles operacionales. Las empresas que operan globalmente pronto se dieron cuenta de que cuanto más flexible sea el desarrollo de recursos y mejor sea el enfoque para la extracción del valor de su información mejor atendían las necesidades de sus clientes” (Reary, 2000).

Hoy en día, los sistemas ERP son la base de las empresas de todo el mundo. Se utiliza

como un instrumento de gestión y ofrece a las organizaciones una gran ventaja competitiva. “La necesidad de un eje principal de información para una empresa no ha desaparecido”. (Gartner Group[2], 2009).

Hoy en día Gartner va introduciendo el concepto de ERP II. Como dice Zrimsek, uno de sus analistas, “en 1990 cuando se acuñó el término, ERP era la empresa centrada en sí misma pero no lo en lo que ocurría a su alrededor”. Para Zrimsek, “Hoy, nos estamos moviendo hacia el comercio de colaboración o *c-commerce*. Para ello tienes que compartir la información fuera de la empresa”.

ERP II no sólo es la columna vertebral de la empresa, también es el enlace de información de una empresa en la cadena de suministro. Eso es porque el negocio del mañana va a jugar un papel múltiple en múltiples cadenas de suministro, a partir de fuentes tradicionales a los mercados electrónicos.

### 2.1.3.- Características

Para que un sistema sea considerado ERP tiene que tener las siguientes características:

1. **Debe ser un sistemas integral**: que permita controlar todos los procesos de una empresa de una forma coordinada y eficiente. Los datos que se introduzcan se introducen solo una vez y deben ser consistentes, completos y comunes.

2. **Debe ser modular**: cada empresa tiene unas necesidades diferentes. Por ello, es lógico que ciertos procesos estén presentes unas y en otras no. Los módulos agrupan estos procesos y se suelen instalar en el ERP en caso de que la empresa lo necesite. Se debe de poder usar un módulo libremente sin que afecte a los restantes.

3. **Debe ser adaptables**: los ERP están creados para adaptarse a cualquier tipo de empresa. Esto se logra por medio de la configuración o *parametrización* de cada proceso. Con esto, un mismo ERP puede llegar a ser funcional en un gran abanico de tipos de empresa.

Otras características propias de los sistemas ERP son:

- Procesar todas las transacciones que se producen en todos los departamentos de la empresa.

- Tener un papel clave en la mediación de resultados de la empresa al disponer de toda la información de todas las transacciones de la empresa.

- Realizar un seguimiento, medir e informar de la evolución de los acontecimientos sucedidos en la empresa u organización.

- Dar soporte a las funciones básicas del negocio o actividad.

- El sistema de responder en caso de que se produzcan cambios significativos en los procesos y en las necesidades de información de la empresa.

---

[2] Gartner Group es una consultoría con más de 30 años en el sector de las Tecnologías de la Información

- Debe permitir recoger la información de diferentes ubicaciones, procesarla y ofrecerla a los distintos departamentos y usuarios.

- Debe ofrecer una alta adaptabilidad a la situación particular de cada empresa. En algunos casos, incluso se ofrece al usuario final la utilización del código fuente, pudiendo con ello realizar modificaciones y adaptaciones a medida de cada empresa.

- Deben tener la capacidad y facilidad para ser utilizados por diferentes usuarios de diferentes áreas funcionales.

- Debe soportar el sistema de información dando todo el apoyo necesario para que éste funcione y sea eficaz.

- El sistema ERP debe basarse en una única base de datos que permita integridad, consistencia e integración de los mismos, permitiendo disponer de los diferentes módulos interconectados y actualizados.

**Módulos genéricos**

Los módulos de un sistema ERP varían dependiendo de las características de la empresa, pues son muy diferentes los requerimientos en organizaciones en las que, por ejemplo, su principal negocio es la producción, la distribución o bien los servicios. Algunos de los módulos más comunes son:

- Gestión Financiera

- Gestión de Ventas

- Gestión de Compras

- Gestión de la Distribución y Logística

- Gestión y planificación de la Producción

- Gestión de Proyectos

- Gestión de Recursos Humanos

### 2.1.4.- Clasificación

Existe una posible clasificación de los ERPs dependiendo del tipo de licencia que lo soporta. Tenemos los sistemas: **propietario**, **de código abierto** y **SaaS**.

#### *2.1.4.1.- Propietario*

Los sistemas ERP propietarios son los que requieren el pago de una licencia. Además, esta licencia se paga dependiendo del número de puestos operativos en los que se vaya a instalar el software. El precio total puede llegar a encarecerse dependiendo de los módulos que se vayan a utilizar.

Podemos distinguir distintos tipos de sistemas propietarios dependiendo del sector al que

van dirigidos. Por ejemplo, productos como: Sage (Sage, 1981) o SAP (SAP, 1972) van dirigidos a grandes empresas o multinacionales, ofreciendo un producto sólido y con mayor soporte. O por el contrario, nos encontramos ERPs como: Solmicro (Solmicro, 1997) o Deister (Deister, 1988) que van enfocados a pequeñas y medianas empresas de un sector más concreto.

**Ventajas**

- **Control de calidad**. Por norma general, estos productos tienen un control riguroso de calidad que llevan a cabo muchas pruebas sobre el software que producen.

- **Comunidad de usuarios**. Cuando el software propietario es una marca conocida es fácil encontrar a alguien que lo sepa usar o alguna comunidad de usuarios que le de soporte. Además, es posible encontrar publicaciones, ampliamente difundidas, que documentan y facilitan su uso.

- **Software específico**. Existe software diseñado para sectores muy específicos que no existe en ningún otro lado más que en la compañía que lo produce.

**Inconvenientes**

- **Código fuente privativo**. Es uno de los mayores inconvenientes ya que hace a la empresa cliente completamente dependiente de la empresa desarrolladora. No se pueden realizar posibles cambios en la funcionalidad sin el apoyo de la empresa propietaria y, en caso de que ésta desaparezca, el cliente queda totalmente desprotegido.

- **Soporte técnico ineficiente**. En la mayoría de los casos el soporte técnico es insuficiente o tarda demasiado tiempo en ofrecer una respuesta satisfactoria.

- **Derecho exclusivo de innovación**. En caso de necesidad de innovación en el ERP propietario, sólo su empresa desarrolladora decide si evolucionar su producto o no.

- **Coste de las licencias**. Para implantar el ERP en los puestos operativos de la empresa cliente es necesario desembolsar una cantidad por cada licencia. Cuantos más puestos operativos sean necesarios utilizar, mayor es el coste de implantación. Esto hace que la accesibilidad al ERP se vea reducida.

#### *2.1.4.2.- Código abierto*

“Los programas privativos te dan libertad. Tienes la libertad de escoger entre los errores viejos y los nuevos.” (Richard Stallman, 1998).

Los desarrollos de código abierto u *opensource* son una clara alternativa a los privativos. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código. Sin embargo esto no significa que el producto sea gratuito ya que, por norma general, estos desarrollos necesitan algún tipo de financiación y suelen obtenerlo mediante el soporte opcional a sus herramientas.

Los proyectos de código abierto se basan en la entrega y garantía de las siguientes libertades al usuario final:

- Libertad para usar el programa para cualquier actividad.

- Libertad para el acceso y modificación del código fuente.

- Libertad para la libre distribución de la aplicación, modificado o no.

Si buscamos un ERP de código abierto podemos optar por las siguientes soluciones: Openbravo ERP (Openbravo, 2001), OpenXpertya (OpenXpertya, 2007), Open ERP (Open ERP, 2005) o Compiere (Compiere, 1999).

**Ventajas**

- **Tener una licencia**. Siempre será mejor usar un producto de código abierto a usar uno propietario pirateado.

- **Modificación del código fuente**. Al tener todo el código disponible, cada empresa puede adaptar su ERP según lo vaya necesitando. Además, en caso de que el soporte a la herramienta desaparezca, una empresa con un departamento de desarrollo puede realizarlo sin dependencia alguna de los creadores originales.

- **Comunidad de desarrollo**. Esta es sin duda la mejor baza estos ERPs. Esto es debido a que este tipo de desarrollos vienen respaldados por una comunidad de usuarios que ofrecen soporte e implementación que, por norma general, son completamente gratuitos. Además, cuanto mayor sea la comunidad, más rápidamente evoluciona el ERP y se convierte en un producto más competitivo.

**Inconvenientes**

- **Falta de responsabilidad**. Este tipo de software no suele tener garantía de ningún tipo. Es importante tener buenas referencias antes de implantarlo para evitar futuros problemas.

- **Cambio de licencia**. Es posible que por falta de capital el proyecto se vea condenado al cierre o a un cambio de licencia que implique un desembolsar dinero por cada licencia. En caso de ser así, podríamos quedarnos con una versión obsoleta.

- **Costes ocultos**. Resulta muy difícil, por la propia complejidad de estos proyectos, entender su arquitectura si no se recibe formación. Así pues, resulta necesario invertir en la formación. Además, la gratuidad de la licencia induce a un error cuando pensamos que los desarrollos extra para la personalización tendrán que ser gratis también y esto no siempre es así.

#### *2.1.4.3.- SaaS*

Gracias a la evolución de internet ha surgido un nuevo modelo de negocio conocido como SaaS o software como servicio (*Software as a Service*). Esto consiste en la implantación, desarrollo, mantenimiento y ayuda diaria por parte de la empresa que ofrece el servicio. El cliente tiene el sistema hospedado en la compañía de IT y accede a el por medio de internet. Esta modalidad es compatible tanto con los sistemas propietarios como los de código abierto.

Las características de SaaS son:

- Acceso y administración a través de una red.

- Modelo de la aplicación uno-a-muchos (una instancia, múltiples usuarios), tanto arquitectura como precios, colaboración y administración.

- Las actividades son administradas en lugares centrales y no en una oficina del cliente.

**Ventajas**

- **Costes de hosting**. El cliente no se encarga de almacenar físicamente su ERP, por lo que se ahorra costes de soporte electrónico, mantenimiento y seguridad.

- **Externalización del servicio**. La responsabilidad de la operación recae en la empresa IT. La garantía de la disponibilidad y correcta funcionalidad es parte del servicio de la empresa proveedora del software.

**Inconvenientes**

- **Información remota**. El problema más importante de SaaS es que el cliente no tiene acceso directo a su información ya que todo está almacenado en un lugar remoto, con la pérdida de privacidad, control y seguridad que ello conlleva, ya que la compañía IT podría consultarlos.

- **Código inaccesible**. El usuario puede no tiene acceso al programa, por lo que no podría realizar modificaciones.

- **Migración**. Al ser el servicio y el programa dependientes de la misma impresa esto puede imposibilitar al usuario a migrar a otro servicio utilizando el mismo programa.

#### *2.1.4.4.- Comparativa*

A continuación vemos una tabla [ver tabla 2.1] que compara, para estos tres tipos de ERP, los principales aspectos que debemos tener en cuenta para seleccionar una solución para nuestra empresa:

| | Software Propietario | Software Libre | SaaS |
|---|---|---|---|
| **Coste de adquisición** | Malo | Bueno | Medio |
| **Rapidez de despliegue** | Medio | Medio | Bueno |
| **Coste de mantenimiento** | Medio/Malo | Medio | Bueno |
| **Capacidad Personalización** | Medio | Bueno | Malo |

Tabla 2.1: Comparativa entre los tipos de ERP

Como podemos observar, el software propietario queda casi descartado debido a los costes y complicada maniobrabilidad. El mercado tiende hacia un producto adaptable y

flexible hacia el cliente, complementándolo si es posible, con la rápida implantación y accesibilidad.

Es por ello que en este proyecto, vamos a decantarnos por el estudio e implantación de un ERP de código abierto que se acerque hacia el modelo SaaS en nuestra empresa.

## 2.2.- ERPs de código abierto

Con el paso del tiempo se han desarrollado ERPs de código abierto que son casi tan poderosos como los ERP propietarios. A continuación describimos algunos de ellos:

### 2.2.1.- Compiere

Es una solución que integra ERP + CRM (gestión de las relaciones con los clientes) bajo interfaces Windows y Web (Compiere, 1999). Es una solución internacional basada en los flujos de trabajo que realiza el personal de la empresa. Es una solución 100% Java sobre base de datos Oracle, con servidor de aplicaciones JBOSS [ver imagen 2.1].

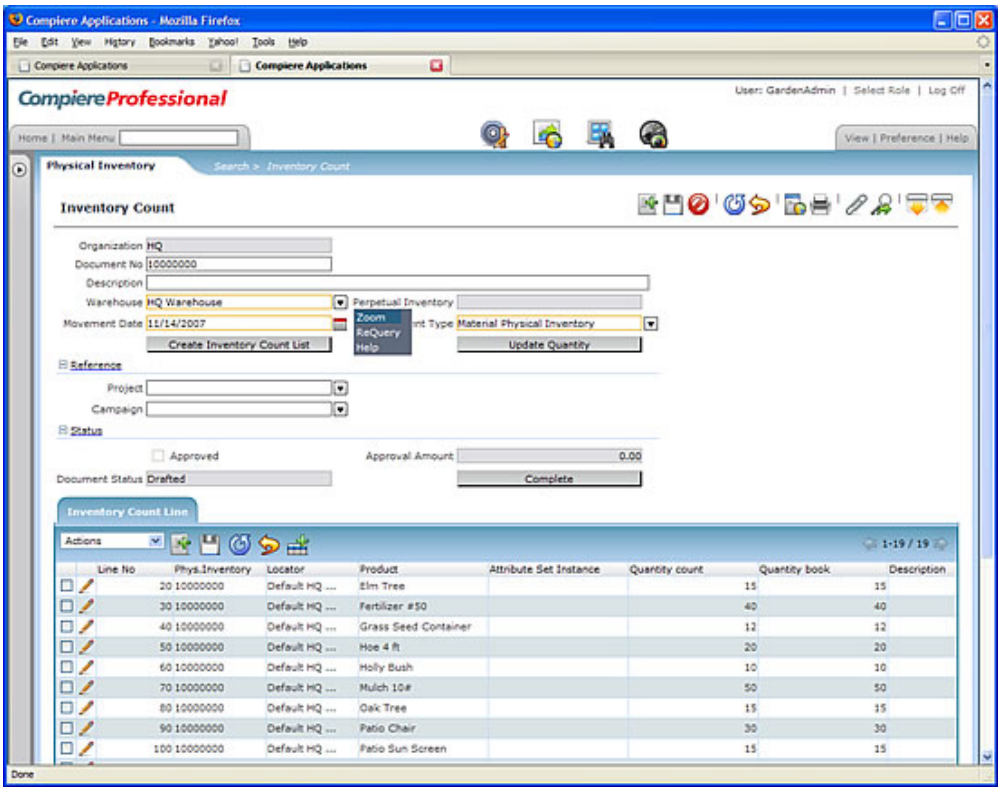



Imagen 2.1: Compiere, listado de inventario

### 2.2.2.- Fisterra

Es un proyecto que pretende crear un ERP genérico hecho con software libre (Fisterra, 2003). En la actualidad soporta: administración de clientes y pedidos, facturación, gestión de stock y de pagos, punto de venta, funcionamiento distribuido y replicación offline de los datos. La implementación usa Gnome SDK y PostgreSQL (libgda) [ver imagen 2.2].

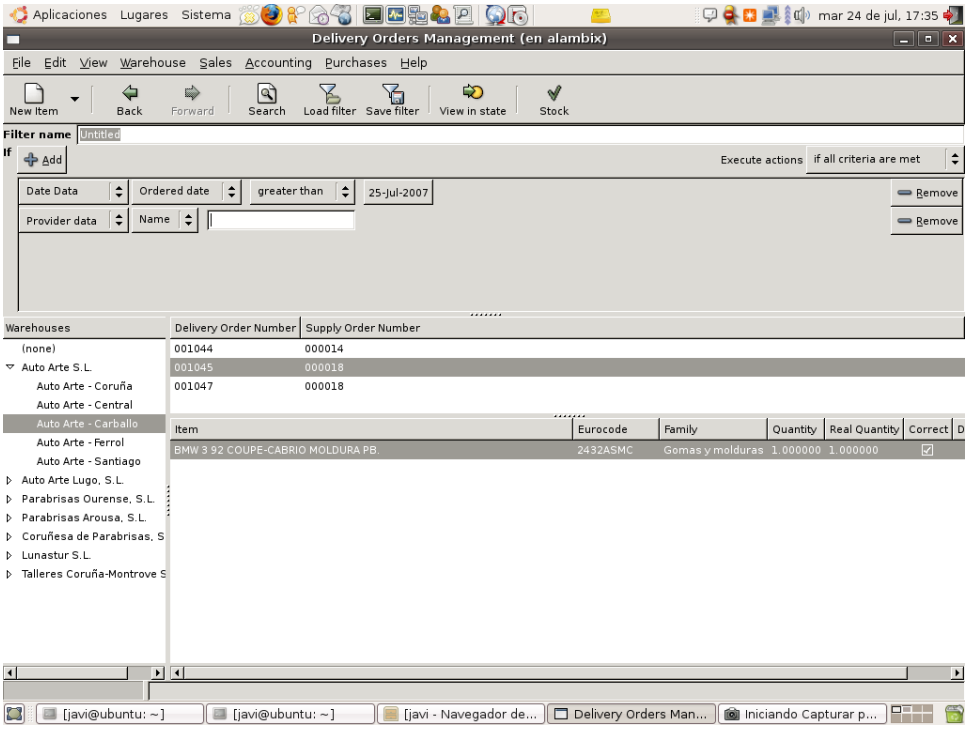



Imagen 2.2: Fisterra, administración de reparto de pedidos

### 2.2.3.- FacturaLUX

FacturaLUX es un software ERP orientado a la administración, gestión comercial, finanzas y en general a cualquier tipo de aplicación donde se manejen grandes bases de datos y procesos administrativos. Su aplicación abarca desde la gestión financiera y comercial en empresas hasta la adaptación a procesos complejos de producción (FacturaLUX, 2008).

### 2.2.4.- OpenERP

Anteriormente llamado TinyERP, es un software de ERP bajo licencia GPL inicialmente desarrollado en Bélgica [ver imagen 2.3]. OpenERP es un ERP principalmente enfocado a la PYME (para 5-50 usuarios), aunque dispone de módulos como gestión de proyectos o estadísticas, más habituales de empresas de mayor tamaño. Implementa funcionalidades como Ventas, Compras, Stock, Contabilidad, Tesorería y algunos más (Open ERP, 2005). OpenERP se encuentra en un estado funcional sobre Linux y Windows (utiliza fundamentalemente la librería de ventanas GTK+), aunque algunos módulos aún están en desarrollo. Actualmente está avanzando hacia su integración con EzPublish. Internamente usa un modelo de Flujos de Trabajo (WorkFlow), con arquitectura en tres capas (interfaz, lógica de negocio y persistencia). Está desarrollado en Python, PyGTK y sobre PostgreSQL, lo que podría ser interpretado como su punto débil debido a la velocidad obtenida.


Imagen 2.3: OpenERP, tabla de proyecto

### 2.2.5.- OpenBluelab

Potente aplicación, que proporciona a una empresa o negocio todas las herramientas necesarias para planificar sus recursos (OpenBlueLab, 2007). Entre las funciones con las que cuenta, ofrece un completo administrador de contenidos y muchas cosas más. Por lo demás es gratuito, y está disponible de momento solamente en inglés. Desarrollada en Java, PHP y Ajax.

### 2.2.6.- Project Open

Project Open es un completo y complejo ERP orientado a proyectos, de propósito general, muy configurable (Project Open, 2009). No está pensado para empresas manufactureras o que fabriquen o vendan productos físicos, sino para empresas de servicios (consultoría, publicidad, IT, sistemas de software, traducciones, etc.).

### 2.2.7.- OpenXpertya

OpenXpertya es una solución integral para la empresa que engloba ERP y CRM, con integración de servicios en línea de B2B o B2C (en función del tipo de cliente final) e incluso B2E (servicios internos) y con soporte de exportación e datos (enlaces) al estandard EDI (intercambio electrónico de información entre empresa: facturas, albaranes, pedidos: EDIFACT, estandard mundial de la ONU) y con posibilidad de trabajar con cubos multidimensionales OLAP (análisis exhaustivo de resultados). Todo ello adaptado muy de cerca a nuestra legislación, tanto fiscal, como mercantil, civil, contable, etc. (OpenXpertya, 2007)

El propósito de openXpertya es cubrir ampliamente todas aquellas necesidades de gestión que una empresa de tamaño medio o grande podría tener. Es la planificación global de todos los recursos.

La solución openXpertya tiene capacidad multientidad, multiempresa, multicentro,

multialmacen, multicaja, etc [ver imagen 2.4].

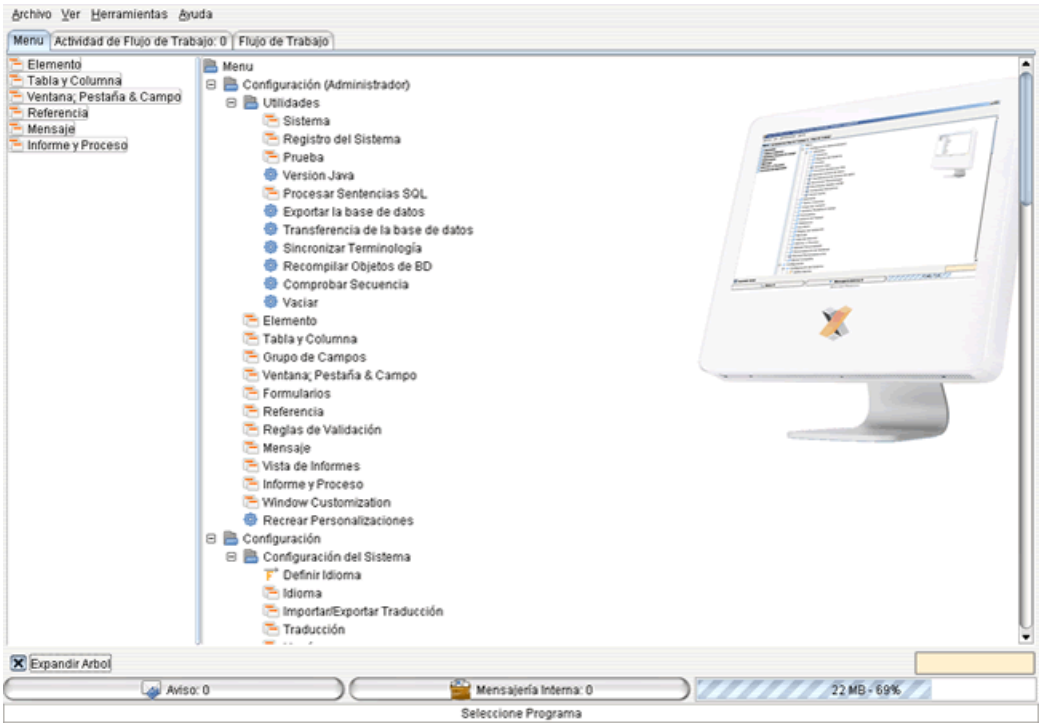



Imagen 2.4: OpenExperya, menú principal

### 2.2.8.- OpenbravoERP

OpenbravoERP es una aplicación de Código abierto de gestión empresarial del tipo ERP destinada a empresas de pequeño y mediano tamaño (Openbravo, 2001). La estructura de datos de la aplicación está basada originalmente en Compiere. Openbravo es una aplicación con arquitectura cliente/servidor web lo que permite que sea utilizada desde cualquier navegador web [ver imagen 2.5].

Escrita en Java se ejecuta sobre Apache y Tomcat y con soporte para bases de datos PostgreSQL y Oracle. Actualmente se encuentra en expansión por lo que se puede encontrar en una gran multitud de lenguas.

OpenbravoERP está licenciado bajo Openbravo Public License Version 1.1 (”OBPL”) que es una adaptación de la licencia libre Mozilla_Public_License.

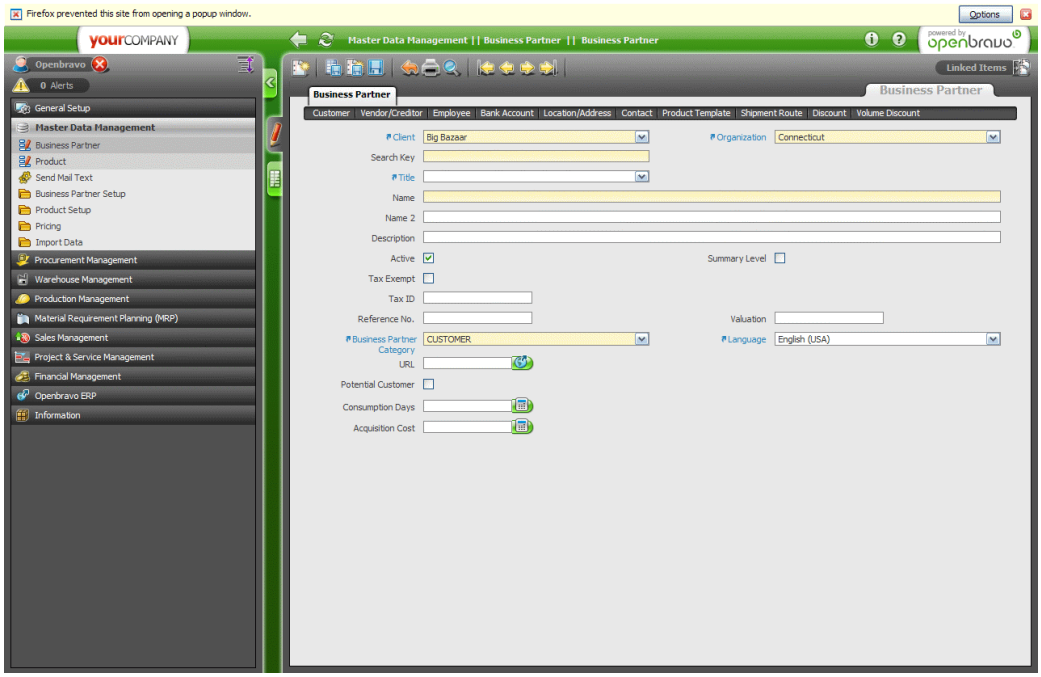



Imagen 2.5: OpenbravoERP, gestión de clientes

## 2.3.- Implantación de un ERP

Una implantación de ERP bien hecha, que cubra correctamente los procesos funcionales de la empresa, que sea adaptable al cambio y centralice la información asegurará un retorno de inversión, mejorará la organización y supondrá un beneficio para la empresa que lo implante. Sin embargo, la implantación de esta herramienta no implica que los problemas en la organización vayan a desaparecer, en absoluto.

Es necsario estudiar las ventajas e inconvenientes de los distintos softwares del mercado que mejor se adapten a la empresa (cada uno cubre un tipo de empresa en función del tamaño, número de usuarios, sector...). La elección del software depende también de la tecnología que se quiera utilizar, por lo tanto, lo primero es analizar los procesos. Luego decidir la herramienta que cubra esos procesos de forma standar en su mayor parte.

Cabe concienciar a los usuarios del cambio que viene y elegir un buen equipo que sepa guiar a la empresa en la implantación. Hay que involucrar a los empleados y aydarles en el uso correcto de la herramienta y su familiarización lo más rápido posible.

El cambio en la metodología de la empresa puede ser drástico, es por ello que cabe realizar un estudio de sus ventajas e inconvenientes.

### 2.3.1.- Ventajas

De una forma concreta se puntualizan los beneficios de los sistemas ERP en nueve puntos (Rashid et al, 2002):

- **Acceso a información fiable**. Este beneficio se logra por:
    - El uso de una base de datos común.
    - Consistencia y exactitud de los datos.

    - Mejoras en lso informes del sistema.

- Evita redundancia de datos y operaciones. Como los distintos módulos del sistema ERP acceden en tiempo real a la misma base de datos central, se evitan dos cosas:

    - Los registros duplicados o múltiples de los mismos datos en el sistema.

    - La duplicación de las operaciones por falta de actualización del registro sobre ellas.

- **Reducción del tiempo de ciclo y de entrega.** Este beneficio se logra, por una parte, al minimizar el proceso de recuperación, y por otra, al realizar informes sobre los retrasos de producción o entrega.

- **Reducción de costes**. Esta reducción se debe tanto a la economía de tiempo, como a las mejoras en el control y en el análisis de las decisiones empresariales.

- **Fácil adaptabilidad**. Los sistemas ERP se pueden modificar a través de la redefinición de sus distintos procesos de negocio, esto hace fácil que se adapte y reestructure para satisfacer los nuevos requerimientos.

- **Mejoras en "escalabilidad"**. Debido a un diseño modular y estructurado los sistemas ERP permiten realizar adiciones de funciones para aumentar o escalar la solución inicial.

- **Mejoras en el mantenimiento**. La existencia de un contrato a largo plazo de mantenimiento con el proveedor, como parte de la adquisición de sistema ERP, hace que mejore el proceso de mantener el sistema de información al día de los avances tecnológicos y de gestión.

- **Alcance fuera de la organización**. Los módulos de extensión de los sistemas ERP como son los CRM (Customer Relationship Management - Gestión de la relación con el cliente), y los SCM (Supply Chain Management - Gestión de la cadena de abastecimiento) hacen que la organización se integre con clientes y proveedores, fuera de los límites tradicionales de la empresa.

- **Comercio electrónico y e-business**. Por una parte esto es posible debido a que la infraestructura tecnológica de los sistemas ERP soportan procesos en Internet, lo que es básico para el comercio electrónico, y por otra parte, a que la adopción de los sistemas ERP desarrolla una cultura de colaboración entre negocios.

### 2.3.2.- Inconvenientes

Las principales limitaciones y obstáculos que pueden suponer la existencia de un ERP en una empresa con los siguientes:

- **Cambios organizacionales**. La implantación de un sistema ERP implica cambios en la infraestructura de Tecnologías de Información de la organización, en los procesos de negocio, en la estructura y en la cultura de la empresa.

- **Superación del análisis coste/beneficio**. Los costes de un sistema ERP son altos y se realizan por adelantado. En cambio, los beneficios casi invariablemente no pueden ser cuantificados al comienzo de un proyecto, y estos solo serán visibles

cuando el sistema comience a operar, y quizás, un tiempo después de ello.

- **Inflexibilidad del sistema ERP**. Tanto la tendencia a ser sistemas complejos, difíciles de dominar totalmente, como la existencia de pocas personas con experiencia en su instalación y mantenimiento, contribuyen a que un sistema ERP pueda transformarse en inflexible.

- **Alcanzar beneficios estratégicos**. Si una organización adopta procesos de negocio que nacen de los modelos genéricos que proporciona el proveedor del sistema ERP puede dejar de utilizar aquellos procesos de negocios únicos que han sido fuente de sus ventajas sobre la competencia. Asimismo, para algunas organizaciones la centralización de la coordinación y la toma de decisiones promovida por los sistemas ERP puede no ser la mejor forma de operar. Algunas empresas claramente no necesitan el nivel de integración que proporcionan los sistemas ERP (Davenport, 1998).

- **El éxito depende en las habilidades y la experiencia** de la fuerza de trabajo, incluyendo la educación y como hacer que el sistema trabaje correctamente. Muchas compañías reducen costos reduciendo entrenamientos. Los propietarios de pequeñas empresas están menos capacitados, lo que significa que el manejo del sistema ERP es operado por personal que no está capacitado para el manejo del mismo.

- **Costes indirectos.** Los vendedores del ERP pueden cargar sumas de dinero para la renovación de sus licencias anuales, que no está relacionado con el tamaño del ERP de la compañía o sus ganancias. Además, Una vez que el sistema esté establecido, los costos de los cambios son muy altos (reduciendo la flexibilidad y las estrategias de control).

- **Sistema rígido**. Los ERP son vistos como un sistema difíciles de adaptar al flujo específico de los trabajadores y el proceso de negocios de algunas compañías.

- En cuanto a la **disponibilidad de algunos datos**, se hace lento el proceso por tener que recalcularlos en el tiempo que son requeridos, para lo cual se hacen consultas en el historial, que no está almacenado de manera directa.

### 2.4.- OpenbravoERP

Para saber porqué hemos elegido este ERP para ser el que implantemos en la empresa, hemos de entender primero la evolución que sigue la tecnología durante estos últimos años y hacia que camino se dirige.

Nos acercaremos también al modelo de negocio que envuelve a este producto y así averiguar el porqué su licencia es gratuita o poequé su código fuente es libre.

Y por último, evaluaremos todas sus ventajas es inconvenientes.

#### 2.4.1.- La evolución a la Web 2.0

La Web 2.0 es la representación de la evolución de las aplicaciones de escritorio hacia aplicaciones web enfocadas al usuario final. Se trata de entornos que impliquen **colaboración entre sus usuarios**. Éste termino (originado en 2003) está comúnmente asociado con un fenómeno social, basado en la interacción que se logra a partir de

diferentes aplicaciones web, que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración.

“En general, cuando mencionamos el término Web 2.0 nos referimos a una serie de aplicaciones y páginas de Internet que utilizan la **inteligencia colectiva** para proporcionar servicios interactivos en red dando al usuario el control de sus datos.” (Tim O'Reilly, fundador de O'Reilly Media)

Hoy en día, es más relevante la accesibilidad de una aplicación que su potencia. Debemos tener en cuenta que tanto la aparición de nuevas tecnologías de servidor como una mejora drástica en el ancho de banda de las conexiones a intertet han favorecido la aparición de plataformas web que necesitan gran cantidad de procesado. Es por ello que los desarrollos han podido “olvidarse” de aspectos secundarios y orientarse más hacia la usabilidad y la accesibilidad.

“No te preocupes tanto de los programas y algoritmos como de la forma en que almacenas y recuperas una enorme cantidad de datos, con latencias mínimas y alta disponibilidad”. (Galli, 2009).

Ricardo Galli es el co-fundador de Menéame. Esta Web es un claro ejemplo de la web 2.0. Es un sitio basado en la participación comunitaria en el que sus usuarios envian historias (enlaces a noticias de cualquier página web) y los demás las votan, premiando las más votadas de forma que se visualizen en la página principal del sitio.

En conclusión, una aplicación web 2.0:

- No debe actuar como una "aplicación cerrada": la información debe poderse introducir y extraer fácilmente.
- Los usuarios deberían controlar su propia información.
- Basada exclusivamente en la Web: los sitios Web 2.0 con más éxito pueden ser utilizados enteramente desde un navegador.
- La existencia de links es requisito imprescindible.

Esto va a permitir a un software funcionar armónicamente con **otras aplicaciones desarrolladas en el mismo entorno 2.0**, integrarse a nivel de servicios y de procesos.

### 2.4.2.- Cambio del modelo de negocio

El modelo de negocio que han venido utilizando las empresas proveedoras de ERPs ha consistido básicamente en la venta de un software y posterior mantenimiento. Aunque este modelo ha servido durante más de diez años y su valor para el negocio de los ERPs es indiscutible, ya no engloba esa visión poderosa en los entornos empresariales hoy en día.

La introducción de una red de nuevas tecnologías y la evolución de las necesidades del mercado están haciendo cambiar el modelo de negocio de los ERPs tal y como se conoce. Estas tecnologías se orientan generalmente hacía las arquitecturas cliente-servidor, es decir, servicios web que se pueden acceder desde cualquier PC con conexión a internet. Visto de éste modo, ya no tiene mucho sentido vender licencias por cada cliente/instancia que se conecte a la aplicación.

La solución que se viene planteando desde hace años no es la venta del propio software, sino la venta de un servicio que lo respalde. Esto se conoce como SaaS (*Software as a Service*), o también denominado "bajo demanda" (*On-demand*). SaaS es un modelo de distribución de software donde las aplicaciones están hospedadas por el ISV (*Indpendent Software Vendor*) y el cliente accede a ellas a través de una conexión de red, de manera que libera a las empresas usuarias del mantenimiendo de las aplicaciones, de la operación y del soporte técnico.

A su vez el vendedor mantiene la propiedad de la aplicación haciéndose cargo de las instalaciones, mantenimiento, actualizaciones, procesamiento, seguridad y otros requerimientos necesarios para asegurar la disponibilidad del servicio. La aplicación está basada en un conjunto común de código y de definiciones de datos para todos sus clientes, ésto se denomina arquitectura "multitenancy" (multi-inquilino).

Un ejemplo cercano a esta tendencia es OpenbravoERP. Su licencia de adquisición es completamente gratuita. El único coste que puede conllevar su implantación es precisamente lo que ofrece SaaS: mantenimiento de servidores, asesoramiento, seguridad, actualizaciones, etc.

La única barrera que separa a OpenbravoERP de un sistema completamente SaaS es que no es del todo compatible con la arquitectura multi-inquilino. Es decir, las diferentes empresas que van a utilizar una misma instancia están obligadas a ser muy parecidas en sus prácticas de negocio. Esto es debido a que el módulo *Application Dictionary* de OpenbravoERP aplica modificaciones en la instancia del ERP y esto afecta a todas las empresas "alojadas" en él. Sin embargo, ofrece una buena segmentación de las empresas de cara a los diferentes administradores, roles y usuarios del ERP.

### 2.4.3.- Arquitectura en tres capas

Como ya hemos mencionado anteriormente, OpenbravoERP es de código abierto y cualquiera lo puede modificar según sus necesidades. Para alguien que desconozca el código fuente de la aplicación y se proponga modificarlo va a necesitar horas de estudio para saber cómo está implementado, qué código ejecuta cierta funcionalidad o en que fichero está la clase que está buscando. Esta tarea puede ser muy tediosa si el código fuente está mal estructurado o no respeta una serie de estándares que definen una nomenclatura común a utilizar.

OpenbravoERP está programado respetando la arquitectura en tres capas. Esto es un modelo de programación el cual separa el código fuente en tres partes independientes: la capa de presentación, de negocio y de datos [ver imagen 2.6]:

- **Capa de presentación**: es la parte de la aplicación con la que interactuará el usuario: la interfaz gráfica. Ésta ha de ser lo más "amigable" posbile (fácil de manejar) para ofrecer un sistema cómodo al usuario. Al ser OpenbravoERP un desarrollo Web, las tecnologías que se utilizan para esta capa son HTML y JavaScript, que son precisamente los lenguajes que van a "entender" los navegadores Web. En cierto modo, la arquitectura cliente-servidor que utiliza OpenbravoERP está muy ligada a la arquitectura en tres capas. La capa de presentación se comunica únicamente con la capa de negocio.

- **Capa de negocio**: es donde reside toda la lógica de negocio de la aplicación. La capa de presentación envía las órdenes del usuario a esta capa, y ésta se encarga de realizar todas las operaciones necesarias para llevarlas a cabo. Por cada operación se debe devolver un *feedback* o retorno para que sea visualizado por la

capa de presentación. En OpenbravoERP la capa de negocio va implementada con el lenguaje de servidor Java (J2EE). La capa de negocio es la única que se comunica con las capa de datos.

- **Capa de datos**: es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio. En OpenbravoERP se pueden utilizar los Gestores de Bases de Datos PosgreSQL o Oracle. La capa viene implementada también en Java, SQL y además se utiliza el lenguaje XML para configurar el acceso a las tablas de la base de datos.

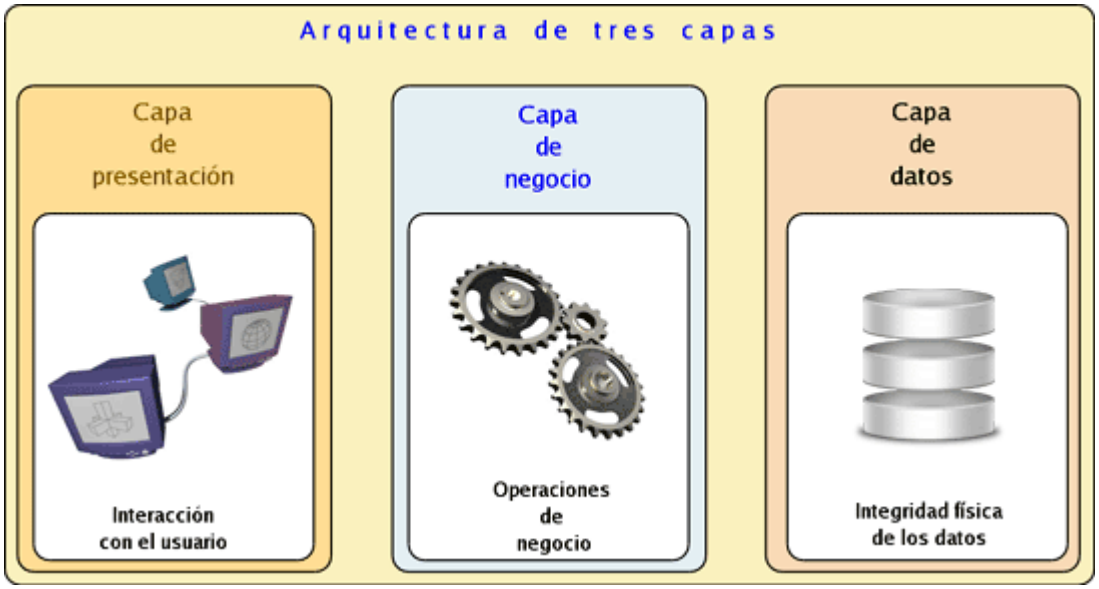



Imagen 2.6: Arquitectura de tres capas

En OpenbravoERP, la capa de presentación será siempre el ordenador del cliente con su navegador Web. Las capas de negocio y de datos van alojadas en un servidor web el cual responderá las peticiones HTTP de los clientes. Sin embargo, es posible alojar la capa de datos en un servidor diferente al de la capa de negocio para mejorar los tiempos de respuesta y la escalabilidad[3].

La ventaja principal de la arquitectura de tres capas es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sea necesario realizar algún cambio, solo hará falta retocar la capa correspondiente sin necesidad de revisar/modificar el resto.

Además, el desarrollo de una aplicación que soporte esta arquitectura va a estrucutrarse mucho mejor ya que al equipo de programadores le va a resultar más sencillo dividir las tareas. Al cumplir ciertos estándares, un programador que quiera implementar nuevas funcionalidades y que desconozca cómo está organizado el código fuente, tardará mucho menos tiempo en realizar sus tareas.

[3] La escalabilidad es la propiedad por la cual una aplicación le resulta fácil o no de responder a una demanda ascendente de sus usuarios

### 2.4.4.- ¿Porqué OpenbravoERP?

Razones para usar OpenbravoERP frente a otras soluciones en el mercado:

- Coste: Licencia mpl 1.1 (mozilla public license) Open souce. Coste de implantación un 30% menos (eliminas licencias).

- Acesibilidad: acceso vía Web (desde cualquier lugar con acceso a internet).

- Personalización: es posible perosnalizar fácilmente el ERP mediante los módulos. Esun sistema preparado para poder ampliar su funcionalidad.

- ERP completo: gestion de abastecimiento, almacenes, proyectos, fabricacion, ventas y finanzas, asi como niveles basicos de CRM y business intelligence

- Multiusuario: El software permite multi-moneda, multi-compañia y multi-contabilidad.

- Asistencia: tiene una comunidad de servicios alrededor del producto (consultoria estrategica, consultoria de implantacion, mantenimiento, …). Una empresa puede bajarse el programa y configurarlo por su cuenta, o bien contratar los servicios de una red de *partners* con experiencia y soporte de la empresa.

- "La compañía es española, radicada en Pamplona, tiene experiencia sólida y exitosa en el desarrollo de proyectos ERP en una variedad de empresas locales desde 2001 y, a principios de este año, captó cinco millones de euros en inversión de Sodena el fin de plantearse el desarrollo del programa siguiendo un esquema abierto". (Dans, 2006).

- Gran comunidad de usuarios: la comunidad de usuarios y *partners* puede dar lugar a un software de gran calidad y adaptabilidad.

## 2.5.- Conclusión

En este capítulo se ha visto el concepto del ERP, su historia y sus características principales. Para complementarlo, se han analizado las ventajas y desventajas que puede tener el llevar a cabo la implantación de un sistema ERP en una empresa.

Para poder elegir el sistema ERP adecuado para una empresa se han analizado una serie de ERP de código abierto que existen en la actualidad en el mercado. Y finalmente, en el úlitmo apartado, se ha justificado la selección del OpenbravoERP como el sistema a implantar.

# 3.- Instalación de OpenbravoERP

En este apartado vamos a instalar OpenbravoERP partiendo de un Sistema Operativo Windows vacío.

Primero de todo analizaremos las posibilidades que nos ofrece el sistema para instalar su ERP. Dependiendo de los requisitos de nuestra empresa elegiremos que tipo de instalación y topología es la más adecuada.

Seguidamente instalaremos la pila de tecnologías y aplicaciones que son necesarias para el correcto funcionamiento del sistema ERP.

## 3.1.- Pre-requisitos de la instalación

Debido a que el sistema que vamos a instalar permite una serie de posibilidades para facilitar esta tarea, debemos ser pacientes y estudiar cada una de ellas para saber cual nos adecua mejor.

Vamos a ver detenidamente los tipos de instalación y las topologías que permite.

### 3.1.1.- Tipos de instalación

La última distribución de OpenbravoERP (la versión 2.50), es posible instalarla de 4 formas distintas:

- **Aplicación Comunitaria**: consiste en una instalación completa para diferentes entornos virtuales (VirtualBox o VMWare) y Sistemas Operativos (Windows o Ubuntu). Este tipo de sistemas se suelen utilizar cuando se requiere una implantación rápida, por ejemplo para testear el sistema o realizar pruebas. En pocos minutos es posible instalar el ERP, pero siempre con el inconveniente de que se está limitado al software de los paquetes que proporciona Openbravo.
- **Instalación Ubuntu**: Openbravo ERP está incluido en los repositorios Ubuntu. De esta forma se puede descargar e instalar el software en el SO Ubuntu Server en pocos minutos (esta distribución está preparada para Ubuntu Server, un SO diseñado para hacer de servidor Web).
- **Instancia Amazon EC2**: Amazon EC2 es un simple y poderoso servicio para administrar infraestructuras de servidores (disco duro, firewall, procesador, memoria). Se puede instalar una distribución GNU/Linux y desde ahí crear una imagen AMI (Amazon Machine Image), la cual administra los recursos del servidor. Openbravo ofrece la posibilidad de instalar su ERP mediante una instancia Amazon EC2 pre-configurada en la cual ya viene instalado todo el sistema.
- **Instalación personalizada**: esta instalación proporciona un control total sobre el sistema. Permite elegir qué versión utilizar de cada herramienta y una personalización completa. El único inconveniente es que requiere configurar todos los niveles del sistema y, por lo tanto, un mayor nivel de conocimiento sobre las tecnologías utilizadas.

Una de las ventajas de realizar una instalación personalizada es que permite colocar las diferentes capas del ERP (persistencia y lógica de negocio) en máquinas diferentes

dependiendo del volumen de datos y el número de accesos concurrentes. La tercera capa, la de presentación, va a ser siempre ejecutada por la máquina cliente desde su navegador (Internet Explorer, Mozilla Firefox, etc.). Estas configuraciones de capas en distintas máquinas son denominadas *topologías*.

### 3.1.2.- Topologías permitidas

Las diferentes topologías que se pueden utilizar con OpenbravoERP son:

- **Un solo servidor**: las dos capas del sistema se colocan en la misma máquina. Esta es la instalación más común.
- **Dos servidores**: un servidor se encarga de la capa de persistencia (base de datos) y otro de la de lógica de negocio (aplicación y servicio Web). Esta topología se utiliza cuando el volumen de datos manejado por la empresa es muy grande y la cantidad de consultas enviadas a base de datos excede de lo normal.
- **Multi servidores**: múltiples servidores ejecutando una base de datos RAC (clusters de base de datos) y múltiples servidores ejecutando la aplicación junto a un balanceador de carga. Esta solución es muy cara, pero con ella se puede escalar OpenbravoERP tanto como lo requiera la situación.

### 3.1.3.- Instalación elegida

Para este proyecto hemos optado por realizar la instalación personalizada en un único servidor Windows. Puesto que tenemos una empresa pequeña no es necesario recurrir a más de una máquina. Además, realizar este tipo de instalación nos va a permitir acceder al código fuente mediante el entorno de desarrollo Eclipse y poder realizar las modificaciones necesarias directamente sobre el código.

## 3.2.- Instalación del *Stack* o Pila Tecnológica

La *Stack* o Pila Tecnológica es el conjunto de tecnologías implicadas en el correcto funcionamiento de OpenbravoERP. Todas ellas están disponibles para los distintos Sistemas Operativos del mercado, de esta forma Openbravo se asegura que su ERP pueda utilizarse en cualquier entorno.

En los entornos virtuales que hemos comentado antes la *stack* ya viene completamente configurada y en funcionamiento. Esto facilita enormemente la instalación de OpenbravoERP de cara a los usuarios con poco nivel de conocimiento en estas áreas.

A continuación vamos a detallar la instalación de la *stack* y del ERP. Seguiremos este orden:

1. Java JDK
2. Eclipse
3. PostgreSQL
4. Apache Ant
5. Apache Tomcat
6. OpenbravoERP

### 3.2.1.- Java JDK

El JDK o Java SDK (siglas en inglés son *Software Development Kit*) consiste en una serie de aplicaciones (o también llamado *framework*) para desarrollar aplicaciones para Java.

El JDK se compone principalmente por:

- **Javac**: el compilador de código fuente para Java.
- **Java**: el intérprete necesario para ejecutar estas aplicaciones en cualquier SO.
- **Appletviewer**: un visualizador de Applets Java.
- **Javadoc**: genera la documentación de un programa automáticamente a partir de sus clases.

**Proceso de instalación**

1. Para instalar el JDK en nuestro SO tendremos que descargarnos el instalador desde esta página Web:

   http://www.oracle.com/technetwork/java/javase/downloads/index.html

2. Una vez descargado ejecutaremos el programa *jdk-6u21-windows-i586.exe*. Por defecto, el JDK se instala en la carpeta "C:/Program Files/Java/jdk1.6.0_21". Esta ruta nos hace falta para configurar la variable de entorno JAVA_HOME, utilizada en muchos programas.
3. Para configurar esta variable haremos clic derecho en "Mi PC" y accederemos a las propiedades. En la parte inferior está el botón "Variables de entorno…". Al hacer clic nos abrirá una nueva ventana.
4. En la parte inferior, "Variables del sistema", haremos clic en "Nueva.." e indicaremos los siguientes valores [ver imagen 3.1].
5. En "Nombre de la Variable" poner: "JAVA_HOME".
6. En "Valor de la Variable" poner: "C:\Program Files\Java\jdk1.6.0_21".

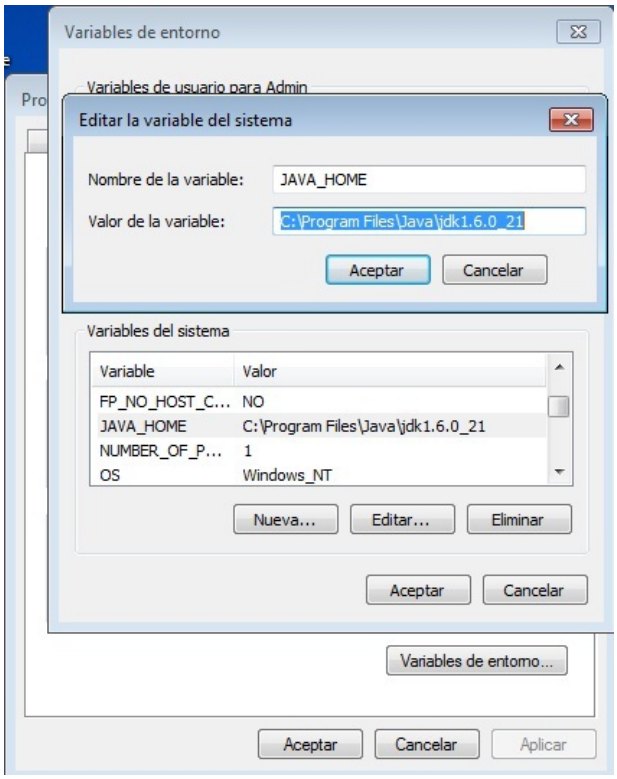



Imagen 3.1: Configuración de la variable de entorno JAVA_HOME

### 3.2.2.- Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto. Este software está principalmente diseñado para programar aplicaciones Java, aunque dispone de una serie de plugins o extensiones que le permiten desarrollar para otros lenguajes como PHP, ASP, etc.

Existen muchos entornos de desarrollo para Java en el mercado, pero Eclipse es uno de los más completos y accesibles (su licencia es gratuita). Además, su comunidad de usuarios aporta cada día nuevas funcionalidades al entorno.

**Proceso de instalación**

1. Para instalar Eclipse en nuestro equipo debemos de ir a la página Web de sus desarrolladores y descargar la versión que más se ajuste a nuestro SO: http://www.eclipse.org/downloads/
2. Una vez descargado procedemos a su instalación haciendo doble clic en el fichero: *eclipse-jee-helios-win32.exe*.
3. El proceso nos guiará por una serie de pasos entre los que configuraremos la carpeta de trabajo o *workspace*. Ésta será la carpeta donde más adelante pondremos el código fuente de OpenbravoERP.

### 3.2.3.- PostgreSQL

PostgreSQL es un Sistema Gestor de Base de Datos de licencia gratuita. OpenbravoERP soporta tanto bases de datos PostgreSQL como Oracle. Nosotros optaremos por PostgreSQL porque es la solución que más se acopla con nuestra pequeña empresa. Tengamos en cuenta que Oracle está preparado para gestionar grandes cantidades de información y además sus licencias son caras.

**Proceso de instalación**

1. Para instalar PostgreSQL primero de todo accederemos a su sitio Web para descargar la última versión para Windows: http://www.postgresql.org/download/
2. Una vez descargado ejecutaremos el fichero: *postgresql-8.4.4-1-windows.exe*.
3. Aparecerán una serie de pantallas en las que indicaremos “Siguiente” hasta llegar a la pantalla de configuración de clave de administrador [ver imagen 3.2]. Aquí introduciremos la clave para el usuario *postgres* con el que administraremos las bases de datos. En nuestro caso pondremos la clave: “1111”.
4. En la siguiente ventana nos preguntará el puerto de escucha del servidor. Indicaremos el valor por defecto: 5432.
5. Haremos clic en el botón finalizar para completar la instalación.


Imagen 3.2: Contraseña para el usuario *postgres*

### 3.2.4.- Apache Ant

Ant es una herramienta utilizada para la programación de tareas de compilación. Es similar a la herramienta Make de C, pero para Java.

Todas las tareas de compilación de OpenbravoERP vienen configuradas en los ficheros *build.xml* para poder ser ejecutadas median Ant.

**Proceso de instalación**

1. Accederemos al sitio oficial de Apache Ant para descargar la última versión estable: http://ant.apache.org/bindownload.cgi.
2. Una vez descargado ejecutaremos el fichero *apache-ant-1.8.1-bin.exe*.
3. Seguiremos los pasos e instalaremos el programa en la carpeta C:\ant.
4. Una vez instalado abriremos la ventana de variables de entorno como detallamos anteriormente en la instalación de Java.
5. Haremos clic la variable Path y luego el botón “Editar” [ver imagen 3.3].
6. Esta variable ya tiene un valor predefinido X. Nuestro objetivo es concatenar un valor Y al ya existente. Debemos de añadir el siguiente valor: “;C:\ant\bin;C:\Program Files\Java\jdk1.6.0_21\bin”. Con esto facilitamos desde línea de comandos ejecutar Java y Ant desde cualquier carpeta.

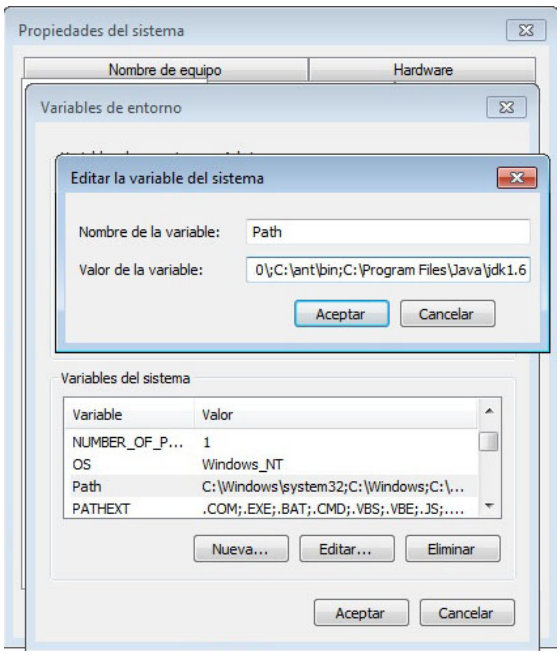



Imagen 3.3: Configuración del Path

7. Desde la misma ventana de Variables de entorno, hacer clic en el botón “Nueva..” y añadir: Nombre de la variable: “ANT_HOME” y el valor: “C:\ant” como indica la imagen 3.4.
8. Añadir otra variable con nombre: “ANT_OPTS” y valor: “-Xmx1024M”. Con esto permitiremos a Ant usar más memoria para sus procesos de compilación.

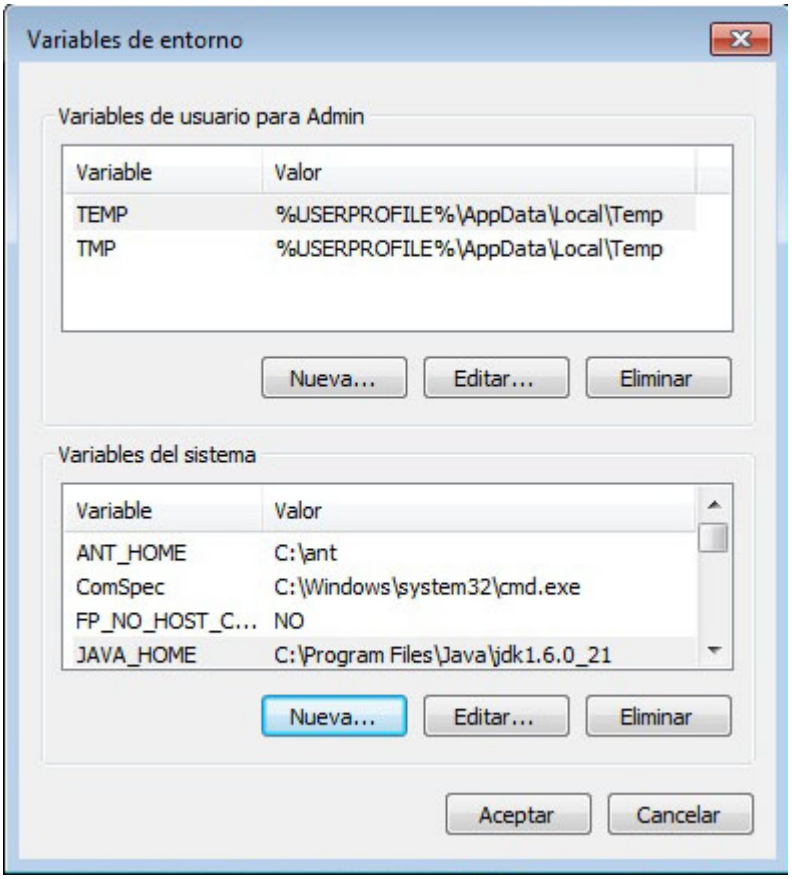



Imagen 3.4: configuración de la variable ANT_HOME

9. Por último, para comprobar que la configuración es correcta, vamos a abrir una línea de comandos y ejecutar el comando “ant”. Si el resultado es como en la imagen 3.5, Ant está correctamente instalado.

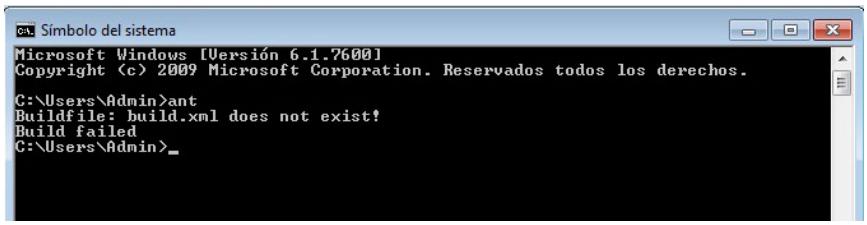



Imagen 3.5: Comprobando la instalación Ant

### 3.2.5.- Apache Tomcat

Tomcat es un contendor de aplicaciones Web Java, o también llamados *servlets*. Cuando un cliente con su navegador Web accede a un servidor vía HTTP, éste responde ejecutando la aplicación seleccionada por el cliente y devolviéndole código HTML para que pueda visualizarla el navegador. El objetivo de Tomcat es ejecutar esas aplicaciones Java en el servidor para devolver páginas Web a los clientes.

Además, Tomcat puede también funcionar como Servidor Web. Siendo innecesario instalar por ejemplo un Apache o un Internet Information Server ya que tiene esta funcionalidad.

**Proceso de instalación**

1. Ir a la página oficial de Apache Tomcat y descargaremos la versión 6.0.29 para Windows, en su versión “deployer” (no es un ejecutable): http://tomcat.apache.org/download-60.cgi.
2. Descomprimir su contenido en la carpeta “C:\apache-tomcat-6.0.29”.
3. Añadir la variable de entorno, nombre: “CATALINA_HOME” y valor: “C:\apache-tomcat-6.0.29”.
4. Añadir la variable de entorno, nombre: “CATALINA_OPTS” y valor: “-Djava.awt.headless=true -Xms384M -Xmx512M -XX:MaxPermSize=256M”. Esta modificación se ejecuta como parámetros de Tomcat. Su función es modificar los parámetros por defecto de la memoria que usa Tomcat puesto que son insuficientes para la ejecución de OpenbravoERP.
5. Para activar el usuario administrador de Tomcat tenemos que ir al fichero C:\apache-tomcat-6.0.29\conf\tomcat-users.xml y cambiar las siguientes líneas de código:

```
<!--
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
<user username="role1" password="tomcat" roles="role1"/>
-->
```

Por éstas:

```
< role rolename="manager"/>
<role rolename="tomcat"/>
<role rolename="admin"/>
```

```
<role rolename="role1"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="admin" password="admin" roles="admin,manager"/>
<user username="role1" password="tomcat" roles="role1"/>
```

Con esto estamos indicando que el usuario “admin” con la contraseña “admin” será el administrador de Tomcat. Guardamos el fichero

6. Abrir una línea de comandos y acceder a la carpeta “C:\apache-tomcat-6.0.29\bin” y ejecutar el comando *startup*.

Si hemos seguido todos los pasos correctamente, al abrir un navegador Web e introducir la URL http://localhost:8080 nos aparecerá la página Web de administración de Tomcat [ver imagen 3.6].


Imagen 3.6: Web de administración de Tomcat

### 3.2.6.- Instalación de OpenbravoERP

Llegados a este punto, ya tenemos configurada por completo la pila tecnológica necesaria para poner en funcionamiento el ERP. Para instalarlo vamos a seguir los siguientes pasos:

1. Descarga OpenbravoERP desde la página oficial: http://www.openbravo.com/downloads/.
2. Descomprime el fichero *OpenbravoERP-2.50MP20.zip* en la carpeta “C:\workspace” (recordemos que esta carpeta es la que almacena todos los proyectos de Eclipse). Este fichero contiene todo código fuente de OpenbravoERP preparado para compilar.
3. Abre una línea de comandos y colócate en la carpeta “C:\workspace\OpenbravoERP-2.50MP20\” e introduce el comando “ant setup”. Esto compilará el instalador de OpenbravoERP y generará un programa ejecutable [ver imagen 3.7].

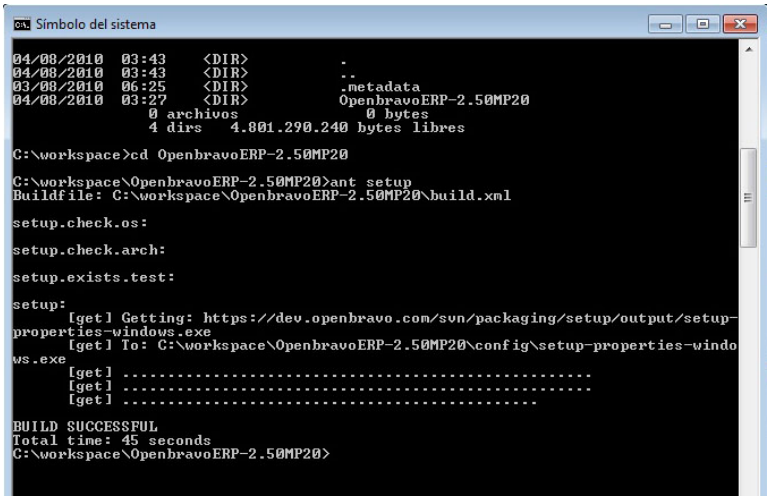



Imagen 3.7: generación del instalador

4. Abre una ventana en la carpeta “C:\workspace\OpenbravoERP-2.50MP20\config\” y ejecuta el programa *setup-properties-windows.exe*. Aparecerá una ventana como en la imagen 3.8.

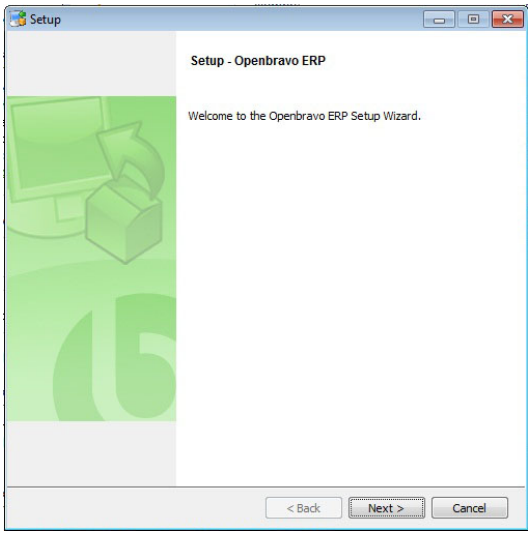



Imagen 3.8: programa instalador

5. Seguiremos los pasos que nos indica el instalador [ver imágenes 3.9, 3.10 y 3.11].

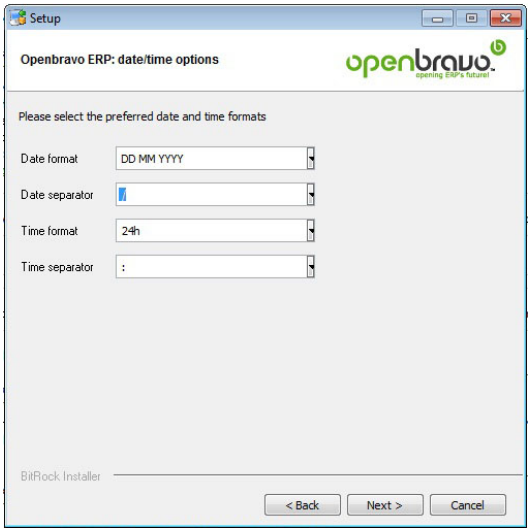



Imagen 3.9: configuración del formato de fecha

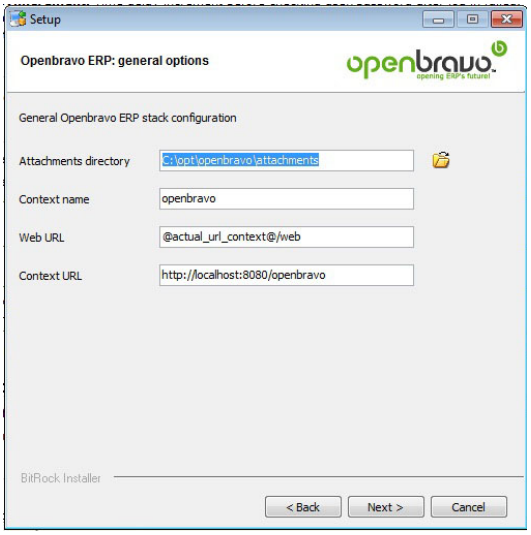



Imagen 3.10: configuración de la *stack*

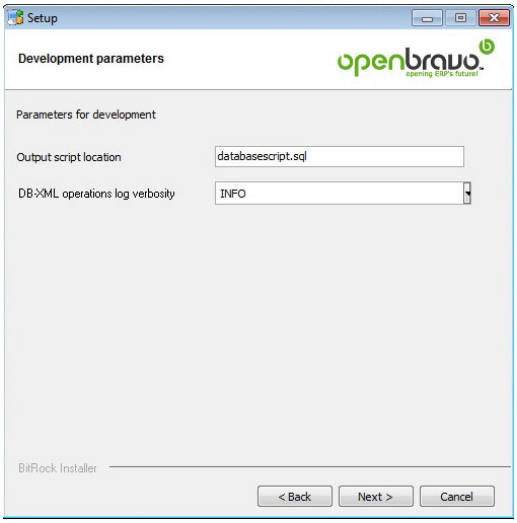


Imagen 3.11: configuración de los scripts de base de datos

6. Llegará un momento en el que el instalador nos pregunte qué tipo de Gestor de Base de Datos vamos a utilizar; Oracle o PostgreSQL. Seleccionaremos PostgreSQL y acto seguido nos preguntará los datos de conexión. Todos los datos que proporciona el instalador por defecto son correctos exceptuando la clave de conexión. Esta clave la configuramos en la instalación de PostgreSQL; su valor fue "1111". La configuración de conexión será así [ver imagen 3.12]:

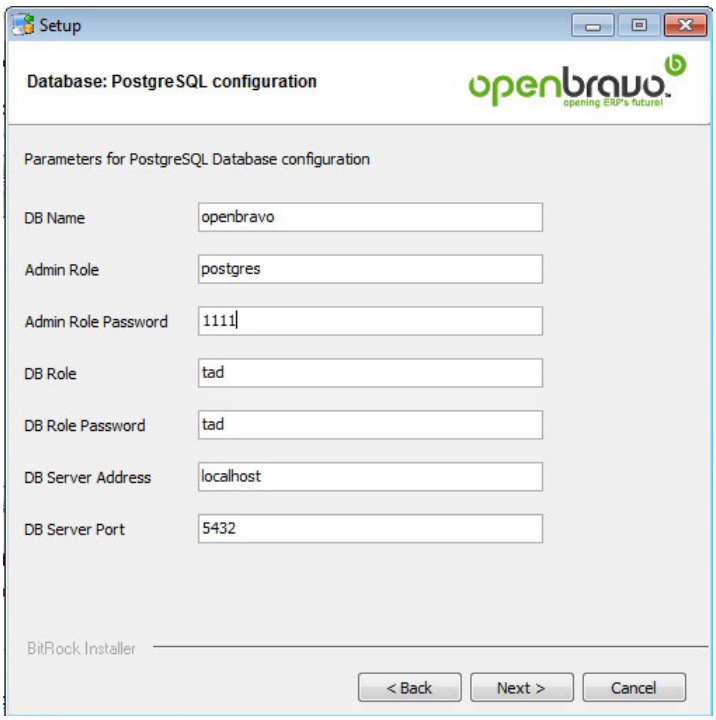



Imagen 3.12: configuración de los datos de conexión a PostgreSQL

7. En la siguiente ventana nos preguntará los datos de acceso a Tomcat. Introducimos el usuario "admin" y clave "admin" (como configuramos anteriormente en Tomcat) [ver imagen 3.13].


Imagen 3.13: datos de acceso a Tomcat

8. Haciendo clic en "Next" llegaremos al final del asistente de instalación de OpenbravoERP. Con esto, los ficheros *Openbravo.properties* y *log4j.lfc* de la carpeta "/config" quedan configurados.
9. Abre el fichero Openbravo.properties con un editor de textos y busca la línea:

```
#Deploy mode: valid values [class, war, none]
deploy.mode=class
```

Y sustitúyela por:

```
#Deploy mode: valid values [class, war, none]
deploy.mode=war
```

Con esto conseguiremos que al compilar el código fuente se genere un archivo **.war**. Este fichero es un archivo comprimido que viene preparado para colocarlo en Tomcat.

10. Abre una línea de comandos y colócate en la carpeta "C:\workspace\OpenbravoERP-2.50MP20\" e introduce el comando "ant install.source" [ver imagen 3.14]. A partir de ahora Ant va a empezar a compilar el código fuente del ERP, ejecutar los scripts de creación de base de datos y generar el fichero **openbravo.war**. Este proceso puede durar alrededor de 20 minutos, dependiendo de las prestaciones del PC. Al finalizar el proceso, si todo ha ido correctamente, recibiremos por línea de comandos la frase "BUILD SUCCESSFUL" [ver imagen 3.15].

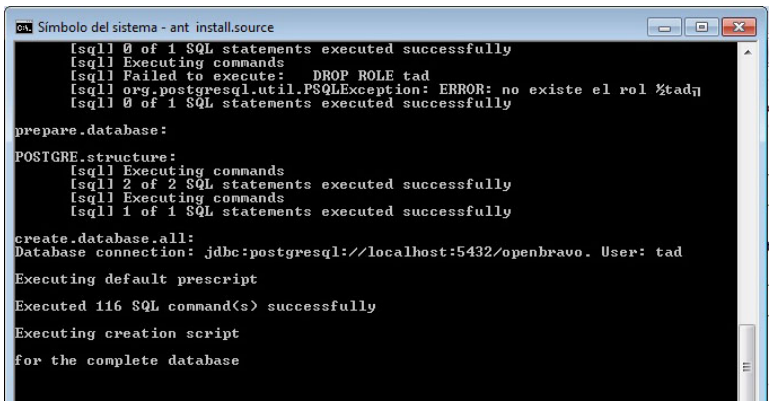



Imagen 3.14: compilando OpenbravoERP

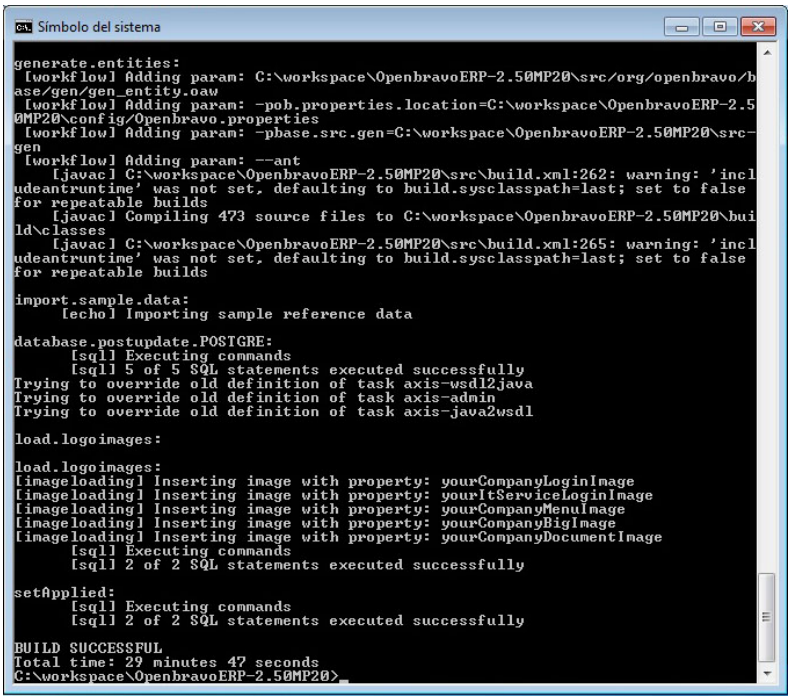



Imagen 3.15: compilación correcta

11. Al abrir la carpeta "C:\workspace\OpenbravoERP-2.50MP20\lib" encontraremos el fichero openbravo.war. Lo copiamos en la carpeta "C:\apache-tomcat-6.0.29\webapps\" para que Tomcat pueda acceder a él.
12. En la línea de comandos, vamos a la carpeta "C:\apache-tomcat-6.0.29\bin\" y ejecutamos "startup" para arrancar Tomcat. Durante este proceso se hace un *deploy* (descompresión) del archivo openbravo.war para generar la estructura de ficheros necesaria para que el servicio OpenbravoERP pueda funcionar. Cuando se muestre los segundos que ha tardado en arrancar significa que ya esta listo [ver imagen 3.16].

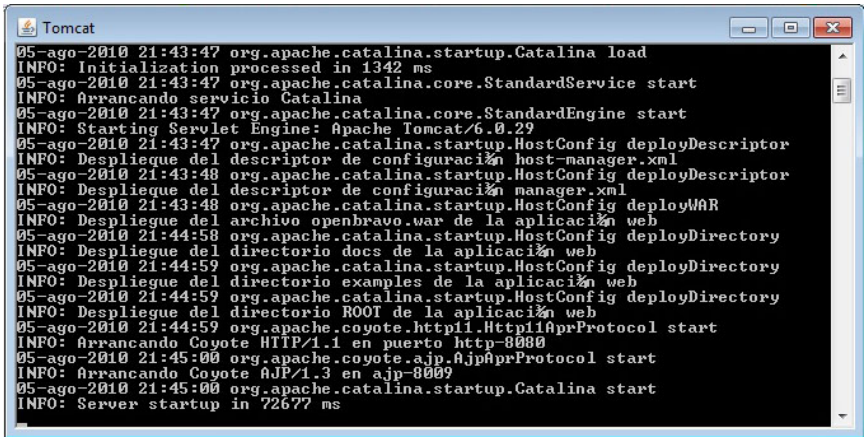



Imagen 3.16: Arranque de Tomcat

13. Por último, para comprobar si Tomcat ha arrancado el proceso correctamente, abriremos un navegador Web e introduciremos la URL "http://localhost:8080/openbravo". Si vemos la pantalla de login de OpenbravoERP es que todo ha salido correctamente.

# 4.- Estudio de los modelos de implementación

OpenbravoERP es un sistema de gestión empresarial que está preparado para adaptarse a todo tipo de sectores. Como comentamos anteriormente, una de sus mayores bazas con respecto a su competencia es que el código fuente es completamente libre y modificable. Con esto se consigue que cada empresa pueda modificar su funcionalidad dependiendo de sus necesidades.

Cuando una aplicación libera su código, cualquier desarrollador puede indagar en él y modificarlo a su antojo. Pero esta tarea no es siempre tan sencilla, ya que cuanto más grande es el sistema, más complejo es realizar modificaciones y que mantenga su coherencia.

El código fuente de OpenbravoERP sigue unas pautas que facilitan enormemente la tarea de realizar modificaciones en él. A lo largo de este capítulo vamos a estudiar su arquitectura en dos apartados: el Modelo de Desarrollo Dirigido y la Modularidad. Esto nos ayudará a abordar la tarea de realizar modificaciones sobre este ERP.

## 4.1.- Modelo de Desarrollo Dirigido

OpenbravoERP incorpora en su estructura lo que se denomina el Modelo de Desarrollo Dirigido o MDD (*Model Driven Development*). Esto quiere decir que usa un metodología para definir los componentes básicos de la aplicación, también llamados *metadata*, como pueden ser; ventanas, procesos o campos de texto. Basándose en este modelo, OpenbravoERP consigue generar automáticamente porciones de código fuente en Java, aumentando la velocidad de desarrollo y reduciendo la posibilidad de que existan errores de programación.

El MDD proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos. Esta metodología hace que la funcionalidad del sistema esté definida como un modelo independiente de la plataforma (no importa el Sistema Operativo o la arquitectura del hardware). Para implementar un MDD es necesario aplicar múltiples normas, incluyendo el Lenguaje de Modelado Unificado (UML), XML Metadata Intercharge (XMI), etc.

Uno de los principales objetivos del MDD es separar el diseño de la arquitectura y de las tecnologías de construcción, facilitando que el diseño y la arquitectura puedan ser alterados independientemente. Así, se asegura que el sistema sobreviva a los cambios que se produzcan en las tecnologías de fabricación y en las arquitecturas software.

### 4.1.1.- Implementación del MDD

Para la implementación del Modelo de Desarrollo Dirigido es necesario distinguir entre la información que se va a manejar y los procesos implicados en la generación automática de código [ver imagen 4.1].

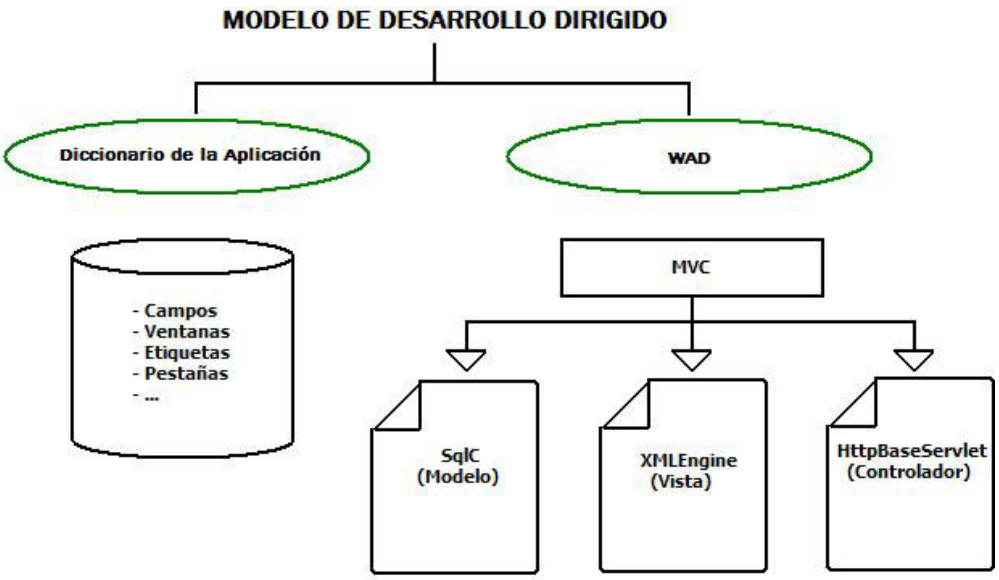



Imagen 4.1: Modelo de Desarrollo Dirigido

Por un lado nos encontramos con el Diccionario de Aplicaciones, que va a servir de almacén de los *metadatos*. En él podemos encontrar un listado de todos los componentes con los que se forma OpenbravoERP: ventanas, campos, etiquetas, etc.

Por otro lado está el WAD (*Wizard for Application Development* o "asistente para el desarrollo de la aplicación"), que es el proceso que genera el código fuente a partir del Diccionario de Aplicaciones. Es decir, todos los ficheros que conforman el ERP son generados a través de WAD. Puesto que esto es una tarea delicada, Openbravo ha desarrollado un Framework (conjunto de aplicaciones) llamado Openbravo MVC, que como su nombre indica, sigue los patrones de la arquitectura MVC (Model-View-Control o Modelo-Vista-Controlador).

El MVC es una clara referencia a la distinción entre las 3 capas principales de una aplicación [ver apartado 2.4.3]:

- Modelo: capa de persistencia o de datos.
- Vista: capa de presentación (interfaz de usuario).
- Controlador: capa de lógica de negocio (contiene la funcionalidad de la aplicación).

### 4.1.2.- Diccionario de Aplicaciones

El Diccionario de Aplicaciones es el módulo de OpenbravoERP cuya función es exclusivamente administrar todos los *metadatos* del sistema de una forma centralizada.

El administrador del sistema tiene acceso a este módulo y puede realizar cambios en él. Puede desarrollar nuevas funcionalidades simplemente definiendo nuevas ventanas, elementos de datos e informes sin necesidad de escribir una línea de código. Estos cambios son almacenados en la base de datos y son utilizados posteriormente por el WAD para generar el nuevo código fuente. De esta forma, los cambios se ven reflejados en la aplicación. La mayoría de código de OpenbravoERP es generado a partir de la información descrita en el Diccionario de Aplicaciones.

### 4.1.3.- Modelo-Vista-Controlador

La función de Modelo-Vista-Controlador es generar código fuente automáticamente sin necesidad de tener conocimientos de programación o de base de datos. Todo este código necesita información sobre la funcionalidad que debe implementar y para ello se debe de complementar con el Diccionario de la Aplicaciones, que es quien almacena todos estos *metadatos*.

Para abarcar la compleja tarea que supone la automatización, Openbravo ha desarrollado tres herramientas que se acoplan a cada una de las capas: el Openbravo MVC.

Cada una de las herramientas del Openbravo MVC sustrae cierta información preparada por el Diccionario de Aplicaciones. Con ellas, es posible realizar casi cualquier cambio en OpenbravoERP. Éstas son: XmlEngine, SqlC y HttpBaseServlet.

**Modelo - SqlC**

SqlC (SQL Compiler) es una utilidad usada para evitar las tareas repetitivas de escribir clases Java para interactuar con la base de datos (capa de persistencia). La entrada es un fichero XML que contiene instrucciones SQL estándar y los parámetros usados en las instrucciones. SQLC lee este fichero y genera una clase Java que contiene todo el código necesario para conectarse a la base de datos, ejecutar las instrucciones, leer los registros y crear tipos estándar Java con lo que devuelve la información.

**Vista - XmlEngine**

XmlEngine es una sencilla aplicación que se encarga de generar la capa de presentación. Se usa para crear documentos XML/HTML desde una plantilla del mismo formato. Además suele ir acompañada de un fichero de conjuración XML de contenido dinámico para ser insertado en la plantilla.

Cuando una página es solicitada, se crea un documento a partir de la plantilla y los parámetros deseados. La herramienta XmlEngine es ka responsable de generar los formularios para editar un registro, listar una selección de registros, crear informes o imprimir formularios específicos de la aplicación.

**Controlador - HttpBaseServlet**

*HttpBaseServlet* y *HttpBaseSecureServlet* son las clases Java genéricas con las que están implementados todos los Servlets del MVC. La funcionalidad del desarrollo viene descrita en esta capa (lógica de negocio).

Un *Servlet* es la clase Java preparada para servir peticiones HTTP desde del contenedor de Servlets: Tomcat. En cierto modo, si un objeto Java quiere comunicarse con un navegador Web debe implementar esta clase.

Openbravo ha creado estos dos Servlets genéricos para implementar funcionalidades comunes como la autenticación, autorización, conexión al a base de datos y el manejo de errores. Estas suelen ser tareas básicas y necesarias, por lo que cualquier desarrollo nuevo debe de implementar. La manera más sencilla y rápida de cubrir esas necesidades es utilizar las clases *HttpBaseServlet* y *HttpBaseSecureServlet* como base de todo desarrollo en OpenbravoERP.

La diferencia entre estos dos Servlets es que *HttpBaseSecureServlet* realiza el control estándar de lectura de datos, interactuando con la base de datos con las clases generadas por SqlC.

## 4.2.- Modularidad

La Modularidad es una nueva capacidad introducida desde OpenbravoERP 2.50 que permite **definir y empaquetar** funcionalidad adicional. Los módulos suelen venir configurados como extensiones, independientes del núcleo del ERP.

La modularidad cambia la forma en que OpenbravoERP se adapta a nuestras necesidades. En vez de modificar el código para adaptarse a las necesidades del cliente, es posible extender su funcionalidad (desde un módulo independiente) y configurarla.

Este sistema, junto con la liberación del código fuente del ERP, ha permitido que una gran comunidad de usuarios desarrollen sus propios módulos y los puedan compartir con el resto de usuarios. Gracias a esto hoy en día podemos instalarnos el ERP básico y añadirle una gran variedad de funcionalidades gracias a los módulos disponibles vía Web.

Las ventajas que aporta la modularidad son:

- **Permite el desarrollo puro distribuido**: las nuevas funcionalidades se pueden desarrollar a través de módulos de una manera distribuida pura. Un equipo de desarrollo puede trabajar aislado de otros equipos ya que el ciclo de vida de su módulo es completamente independiente del desarrollo de otros módulos.
- **Mejora el mantenimiento del código**: el desarrollo a través de módulos significa empaquetamientos independientes. Con una definición apropiada de las dependencias y comprobando que la API sea estable el proceso de actualizar una instancia de un módulo puede hacerse con un simple clic.
- **Favorece el poder compartir y reutilizar las nuevas funcionalidades**: cada equipo de desarrollo puede empaquetar su código en módulos y publicarlo en Openbravo Forge (donde está el Repositorio Central). Después de eso, el módulo quedará público para que cualquier usuario pueda descargarlo e instalarlo.

La modularidad se complementa con el MDD de forma directa. Un conjunto de modificaciones sobre el Diccionario de Aplicaciones representan un módulo.

## 4.3.- Métodos de desarrollo

Con lo que hemos explicado en los apartados anteriores ya podemos abordar la tarea del desarrollo sobre OpenbravoERP.

Primero de todo debemos ser conscientes de los diferentes métodos de desarrollo ya que podríamos obtener de tres maneras distintas el mismo resultado. Los métodos son [ver imagen 4.2]:

- **Diccionario de Aplicaciones**: siguiendo la aproximación MDD, lo más común es editar el módulo del Diccionario de Aplicaciones a través de un navegador Web. Esta vía permite un rápido desarrollo de las nuevas funcionalidades a cambio de cierta limitación.

- **Modificación Directa de los metadatos**: el desarrollador puede conectarse directamente a la base de datos mediante un cliente SQL (pgAdmin III, sqlDeveloper) para administrar los metadatos. Para realizar modificaciones por esta vía hay que tener un alto conocimiento sobre cómo están gestionados los metadatos en las diferentes tablas de la base de datos. Un pequeño error puede acarrear en una inconsistencia grave del sistema.
- **Implementación de código**: puesto que el código fuente es abierto, un desarrollador puede implementar todas las modificaciones que quiera directamente sobre el código fuente mediante un entorno de desarrollo IDE (por ejemplo Eclipse). Esta vía permite una flexibilidad absoluta pero implica un alto conocimiento de las API (conjunto de librerías Java) de Openbravo.

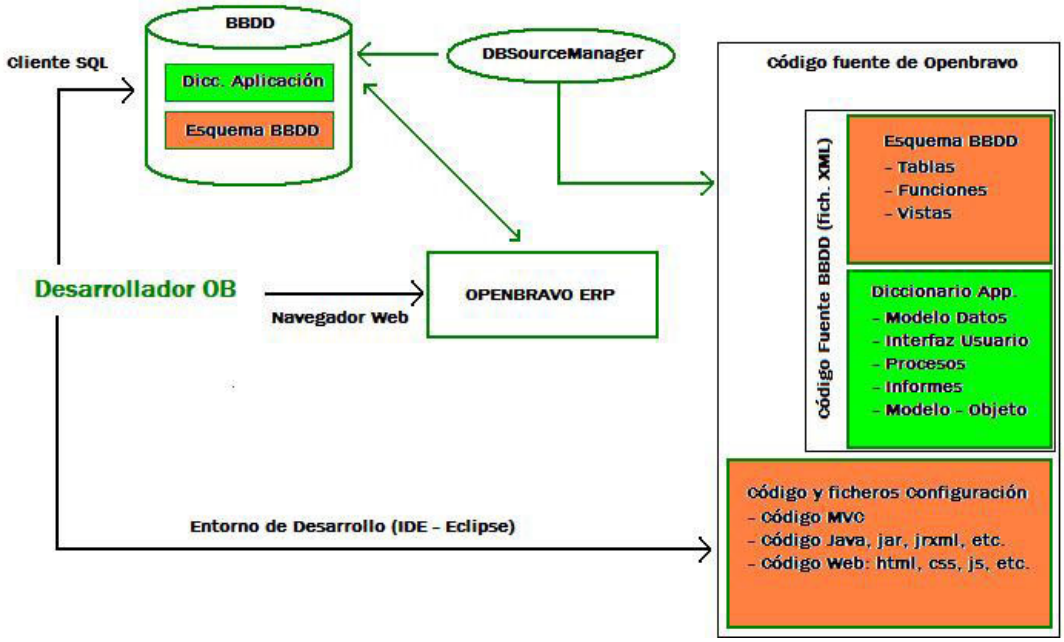



Imagen 4.2: Métodos de desarrollo

Cuando decidamos desarrollar para OpenbravoERP debemos hacer un análisis de los nuevos requisitos y de cómo las herramientas de Openbravo pueden ayudarnos a implementarlos. Hay que tener en cuenta que debemos utilizar el Diccionario de Aplicaciones en la medida de lo posible por varias razones:

- **Rapidez**: se consiguen mejores tiempos de desarrollo.
- **Sencillez**: no exige demasiados conocimientos. Sería más complicado tener que conocer la API de OpenbravoERP.
- **Coherencia**: nos aseguramos de que no va a faltar ninguna conexión entre los componentes del sistema.
- **Consistencia**: es improbable que existan errores de programación, ya que el código se genera automáticamente desde el WAD.
- **Modularidad**: todo lo que modifiquemos se va a empaquetar en un módulo nuevo. Podemos compartirlo. Nos aseguramos de que si actualizamos el núcleo del ERP no se vea afectado nuestro desarrollo.

Aún así, no siempre vamos a poder realizar todas las funcionalidades mediante el Diccionario de Aplicaciones. Será entonces cuando acudamos a la implementación sobre el código fuente.

Para ello contamos con la ventaja de que todos los artefactos software se almacenan en ficheros de texto, dentro del proyecto de desarrollo. Esto incluye la definición del esquema de base de datos y su contenido.

Cuando queramos modificar los metadatos directamente utilizaremos una herramienta llamada DBSourceManager para administrar el código de la base de datos. Ésta puede leer los objetos del esquema de base de datos y del Diccionario de Aplicaciones para exportarlo a ficheros XML. También puede crear o actualizar la base de datos del ERP a partir de estos ficheros.

Una vez hayamos terminado el desarrollo, deberemos de recompilar el ERP mediante Ant [ver apartado 3.2.6]. Este proceso invocará al WAD para que se generaren las clases Java y ficheros XML/HTML a partir del Diccionario de Aplicaciones. Se compilarán todos los ficheros fuente para generar todos los binarios que Tomcat utilizará para visualizar el ERP [ver imagen 4.3]. Si todo ha ido bien, veremos los cambios realizados.

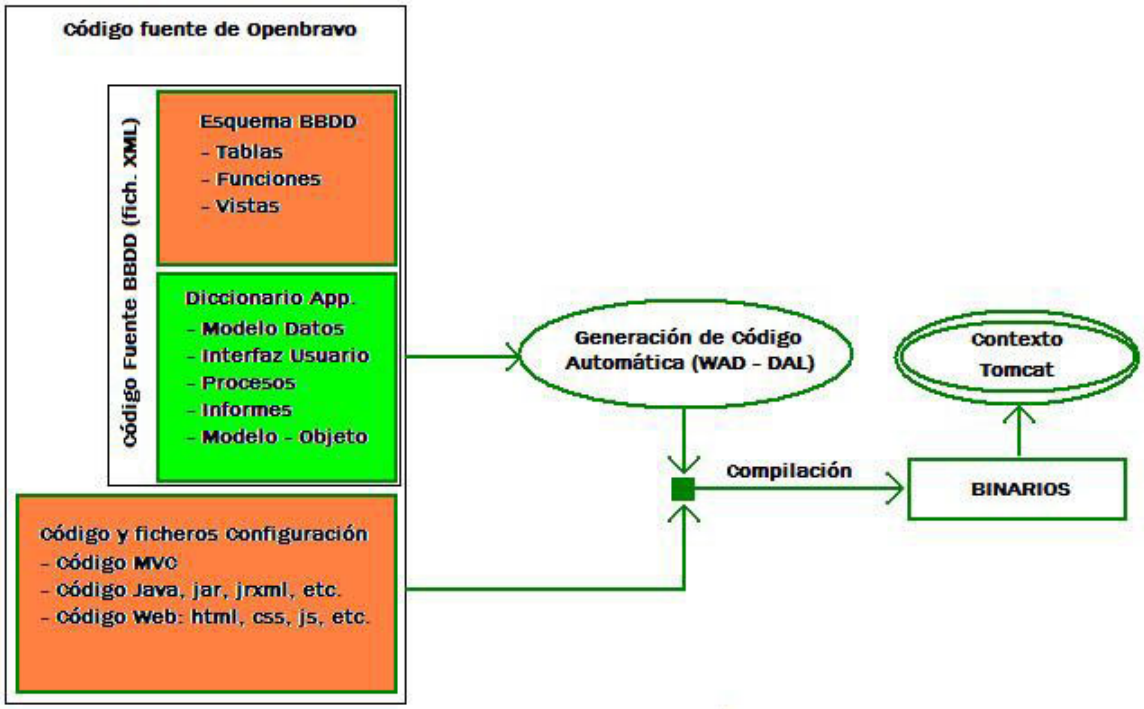



Imagen 4.3: proceso de compilación de OpenbravoERP

# 5.- Revisión del esquema de la Base de Datos

La base de datos de OpenbravoERP consta de 473 tablas, 384 funciones, 75 vistas y 234 disparadores o *triggers*. Ante semejante cantidad de información se hace necesaria la clasificación de todos sus componentes. Es por ello que a la hora de desarrollar se utilizan una serie de normativas o nomenclatura para que todos los desarrolladores sigan. De esta forma se consigue tener un código limpio y fácil de entender.

A lo largo de este capítulo vamos a estudiar cómo está esquematizada la base de datos de OpenbravoERP. Para poder realizar modificaciones en ella necesitaremos conocer antes el estándar de codificación que se debe seguir.

Seguidamente, veremos cómo configurar el acceso a la base de datos desde la aplicación. Esto nos servirá para poder seleccionar la base de datos que queramos en el momento que queramos.

Por último, analizaremos la metodología a utilizar en el caso de querer realizar una modificación en la base de datos.

## 5.1.- Estándar de codificación

El estándar de codificación son una serie de normas que deben seguir todos los desarrolladores que quieran modificar el código fuente o el esquema de la base de datos.

Una de estas normativas es la creación de una serie de campos obligatorios que deben ser comunes para todas las tablas de la base de datos. Estos campos deben de tener un valor obligarlo (son declarados en la base de datos como *not null*).

Las columnas obligatorias para todas las tablas de OpenbravoERP:

- **Clave primaria**: es el campo alfanumérico que identifica al registro. Todas las tablas tienen una única clave primaria. Esta columna debe ser un valor que se calcule automáticamente (tanto PostgreSQL como Oracle se pueden configurar para que generen ellos este valor aleatoriamente). La clave primaria debe nombrarse con la siguiente nomenclatura: <nombre de la tabla> seguido del sufijo "_ID".
- **AD_Client_ID y AD_Org_ID**: Son los campos que identifican a qué cliente y organización pertenece este registro. Esto es debido a que OpenbravoERP es una aplicación multicliente y multiorganización y toda la información debe pertenecer a un cliente y a una organización. Estas columnas son claves ajenas a las tablas AD_Client y AD_Org respectivamente.
- **IsActive**: Indica si el registro está activo o no.
- **Created**: Contiene la fecha en la que se creó el registro.
- **CreatedBy**: Indica el usuario que creó el registro. Este campo es una clave ajena a la tabla de usuarios AD_User.
- **Updated**: Contiene la última fecha en la que se modificó el registro.
- **UpdatedBy**: Indica el último usuario que modificó el registro. Este campo es una clave ajena a la tabla de usuarios AD_User.

En OpenbravoERP, todas las tablas deben estar asociadas a un módulo concreto. Dependiendo de su funcionalidad, ésta es clasificada en un módulo con su nombre mediante una nomenclatura concreta. El nombre de la tabla debe seguir el siguiente patrón: <prefijo del módulo> seguido del nombre de la tabla.

En la siguiente tabla podemos ver los prefijos asociados a cada módulo:

| **Prefijo del módulo** | **Descripción** |
|---|---|
| A | Gestión de activos |
| AD | Diccionario de Aplicaciones |
| C | Módulo central o *core* |
| FACT | Contabilidad |
| FIN | Financias |
| GL | Contabilidad general |
| M | Gestión de Materiales |
| MA | Manufacturación |
| MRP | Recursos de materiales |
| R | Gestión de peticiones |
| S | Gestión de servicios |
| I | Tablas auxiliares para los informes |

Tabla 5.1: Prefijos de los módulos

Otra regla es que cuando un campo de la tabla hace referencia a otra tabla, su nombre debe empezar por el prefijo “EM_” seguido del prefijo del módulo a donde pertenece el campo. De esta forma es rápido distinguir cuales son las claves ajenas de una tabla.

## 5.2.- Arquitectura REST

La Transferencia de Estado Representacional (*Representational State Transfer*) o REST es una técnica de arquitectura de software para sistemas distribuidos en Web. Su principal función es ofrecer una interfaz Web simple que utilice XML y http para que un servidor pueda interactuar con otros servidores.

Openbravo REST es un *Framework* que ofrece seguridad y datos de acceso a los servicios REST para el ERP. Estos servicios Web de acceso a datos (DAL) proveen un modo de recuperar, actualizar, crear y borrar objetos a partir de peticiones http estándar para aplicaciones externas al ERP.

Esto puede ser necesario, por ejemplo, si queremos montar una tienda virtual cuyos datos (artículos, stock, pedidos) los gestiona OpenbravoERP [ver imagen 5.1]. Lo más lo lógico en este caso es que la tienda virtual esté montada en un servidor Web y OpenbravoERP en otro (arquitectura más segura). Gracias a la capa de acceso a datos (DAL) podríamos acceder a los mediante los servicios REST a la información de OpenbravoERP. Estos servicios funcionan, obviamente, previo registro y acceso de un usuario privilegiado.

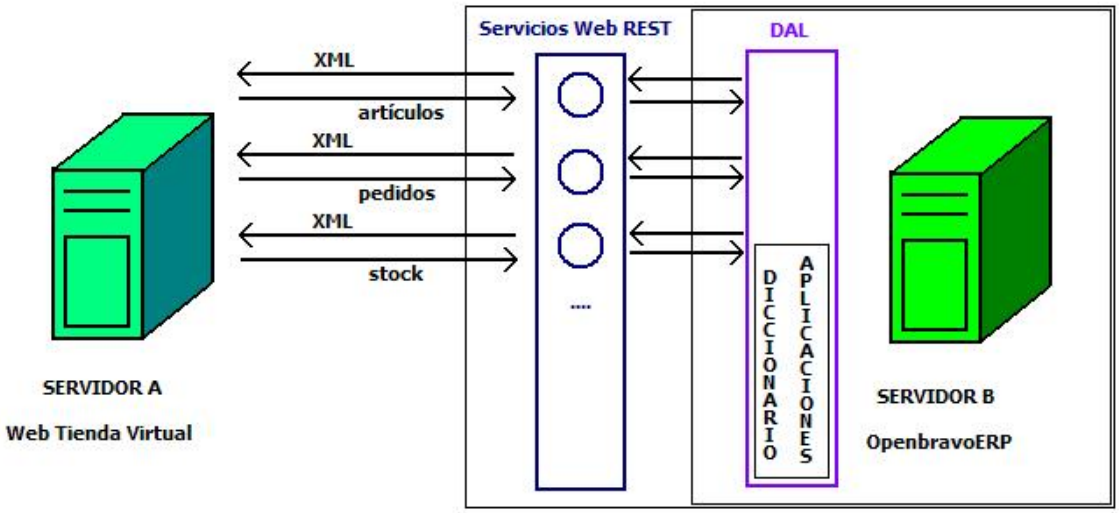



Imagen 5.1: Arquitectura de los servicios Web REST

El funcionamiento de los servicios REST se basa en la configuración de uno o varios esquemas XML que se adecuan a las operaciones que se desean realizar. Estos esquemas se generan a partir del modelo del Diccionario de Aplicaciones. Una vez configurados los servicios, es posible invocarlos mediante una petición HTTP. Finalmente obtendríamos un listado de objetos formateado en XML con los datos solicitados. Por ejemplo, OpenbravoERP tiene configurado un servicio Web que recupera todo el esquema de base de datos en formato XML. Con esta URL podríamos invocarlo:

http://localhost:8080/openbravo/ws/dal/schema

Para acceder al servicio Web tendremos que identificarnos con el usuario “Openbravo” y clave “openbravo”.

## 5.3.- Capa de Acceso a Datos

A partir de OpenbravoERP 2.50 existe un nuevo desarrollo llamado Capa de Acceso a Datos (*Data Access Layer* o DAL). El objetivo de esta herramienta es fortalecer la capa intermedia de la aplicación que se encarga de acceder a la Base de Datos. Sus funcionalidades son:

- Proporcionar ayuda para la realización de consultas seguras y recuperar objetos con la información solicitada (de esta forma no se requieren conocimientos de SQL para obtener información de la base de datos).
- Proporcionar una API (conjunto de clases y funciones) para realizar inserciones y actualizaciones en la base de datos.
- Manejo de transacciones.
- Realizar comprobaciones de validación y seguridad.
- Mejorar la productividad del programador proporcionando una mejor comunicación con la API y el entorno de desarrollo (IDE).
- Mapea automáticamente las nuevas entradas en el Diccionario de Aplicaciones en las tablas y campos correspondientes de la base de datos.
- Generar las clases Java en base a modelo del Diccionario de Aplicaciones de forma automática.

En el DAL se pueden distinguir dos grupos de procesos: los que se realizan en tiempo de desarrollo[4] y los de tiempo de ejecución[5]. En el tiempo de desarrollo el DAL se encarga de generar las clases Java necesarias de la capa de negocio. Y en tiempo de ejecución se encarga de mapear (crear asociaciones entre el código Java y la base de datos) las clases con la información de la base de datos, además de añadir un nivel más de seguridad y validación.

El DAL es una herramienta esencial cuando se pretende comunicar otra aplicación con OpenbravoERP. Esta capa funciona de pasarela y si se quiere cierta información de la base de datos de OpenbravoERP, hay que implementar su protocolo.

## 5.4.- Conexión a la Base de Datos

El archivo *Openbravo.properties*, que se puede localizar en la carpeta /config de la carpeta de trabajo de Openbravo, sirve para configurar los datos de conexión de la aplicación a la base de datos. Esta información se puede cambiar abriendo directamente el fichero con un editor de texto y modificando la información correspondiente.

A continuación vamos detallar qué posibilidades nos ofrece este fichero. El primer apartado contiene las siguientes líneas:

```
####################
# Date/time format #
####################

dateFormat.js=%d-%m-%Y
dateFormat.sql=DD-MM-YYYY
dateFormat.java=dd-MM-yyyy
dateTimeFormat.java=dd-MM-yyyy HH:mm:ss
dateTimeFormat.sql=DD-MM-YYYY HH24:MI:SS
```

Las tres primeras líneas son comentarios[6] para declarar el comienzo del apartado. Las otras cinco son pares nombre-valor que configuran el formato de la fecha en los diferentes lenguajes de programación. Esto es así ya que, por ejemplo, en los países angloparlantes la fecha se suele decir en este orden: mes-día-año y en España se dice como: día-mes-año.

```
#############
# Database #
#############

# Oracle example:
#
# bbdd.rdbms=ORACLE
# bbdd.driver=oracle.jdbc.driver.OracleDriver
# bbdd.url=jdbc:oracle:thin:@localhost:1521:xe
# bbdd.sid=xe
# bbdd.systemUser=SYSTEM
```

[4] En programación, se refiere al momento del tiempo en el que se programa la aplicación.
[5] En programación, se refiere al momento del tiempo en el que se ejecuta la aplicación.
[6] En los ficheros de configuración, las líneas que empiezan por el símbolo <#> son consideradas como comentarios y no se tienen en cuenta.

```
# bbdd.systemPassword=SYSTEM
# bbdd.user=TAD
# bbdd.password=TAD
# bbdd.sessionConfig=ALTER SESSION SET NLS_DATE_FORMAT='DD-MM-YYYY' NLS_NUMERIC_CHARACTERS='.,'

# Oracle instances in linux, can delay on getting DB connection (issue #12683).
# In these cases this property can be set to solve the problem
# java.security.egd=file:///dev/urandom
bbdd.rdbms=POSTGRE
bbdd.driver=org.postgresql.Driver
bbdd.url=jdbc:postgresql://localhost:5432
bbdd.sid=openbravo
bbdd.systemUser=postgres
bbdd.systemPassword=1111
bbdd.user=tad
bbdd.password=tad
bbdd.sessionConfig=select update_dateFormat('DD-MM-YYYY')
```

Las líneas que acabamos de leer se corresponden a los datos de conexión a la base de datos. Proporcionan la localización, en nombre de la base de datos, el usuario y la contraseña entre otros valores. Como se puede observar, hay dos grupos de pares nombre-valor: uno con el símbolo <#> delante y otro sin él. Esto es debido a que OpenbravoERP puede establecer conexiones a las bases de datos Oracle y PosgreSQL. El primer bloque sirve para realizar una conexión a Oracle y el segundo a PostgreSQL, que como es nuestro caso y para que tenga efecto, está configurado sin <#>.

Los parámetros para la conexión a la base de datos son:

- **rdbms**: nombre del Sistema Gestor de Base de Datos (SGBD); ORACLE o POSTGRE.
- **driver**: driver JDBC utilizado para comunicarse ; oracle.jdbc.driver.OracleDriver o org.postgresql.Driver.
- **url**: la URL JDBC de la base de datos. Esta URL sirve para agrupar cierta información de la base de datos, como por ejemplo la dirección IP o el puerto de conexión.
- **sid**: especifica el nombre de la base de datos a conectarse.
- **systemUser**: nombre del súper usuario (usuario con más privilegios) del SGBD.
- **systemPassword**: contraseña del súper usuario.
- **user**: nombre del usuario de la base de datos de OpenbravoERP.
- **password**: contraseña del anterior usuario.
- **sessionConfig**: formato de la sesión de la base de datos.

## 5.5.- Diagramas Entidad-Relación

### 5.5.1.- Introducción

A la hora de desarrollar un módulo nuevo o de modificar uno existente, debemos tener en cuenta qué tablas son las que están implicadas en la funcionalidad deseada. Para ello, nos será de gran utilidad desarrollar un diagrama Entidad-Relación de los módulos implicados en el nuevo desarrollo. El diagrama nos permitirá analizar y diseñar la implementación de forma más cómoda. De esta forma evitaremos, por ejemplo, duplicar campos existentes.

Como comentamos al principio del capítulo, OpenbravoERP consta de 473 tablas. No es el objetivo de este proyecto analizar las relaciones de cada una de ellas. Puesto que nuestro

objetivo es más didáctico, nos centraremos en el aprendizaje para la elaboración de un módulo concreto.

En el repositorio de Openbravo, en el apartado de Documentación, podemos encontrar un documento en el cual se detallan todos los diagramas Entidad-Relación del esquema de su base de datos. Este documento se puede utilizar a modo referencia cuando buscamos información concreta, pero debemos tener cuidado ya que en él se detalla la base de datos de la versión 2.40 de OpenbravoERP. A partir de la versión 2.50 se incluyen nuevas funcionalidades importantes como la modularidad, así que la base de datos sufre cambios. A fecha de hoy, no existe el mismo documento para la versión 2.50.

### 5.5.2.- Elaboración de los diagramas E-R

Los diagramas de Entidad-Relación son un diseño conceptual de la base de datos. Estos modelos expresan entidades relevantes para un sistema de información, en este caso, OpenbravoERP.

Gracias a un Diagrama de Entidad-Relación es mucho mas sencillo observar las dependencias entre entidades. Si queremos trabajar en un nuevo desarrollo sobre OpenbravoERP, y éste depende de ciertas entidades ya existentes, nos será de gran utilidad para tener una visión global de todas ellas.

Para facilitarnos la tarea de construir el diagrama utilizaremos una de las muchas aplicaciones que existen en el mercado. Podemos encontrar:

- **ERWin**: es una herramienta de pago bastante completa para el modelado de datos [ver imagen 5.2].

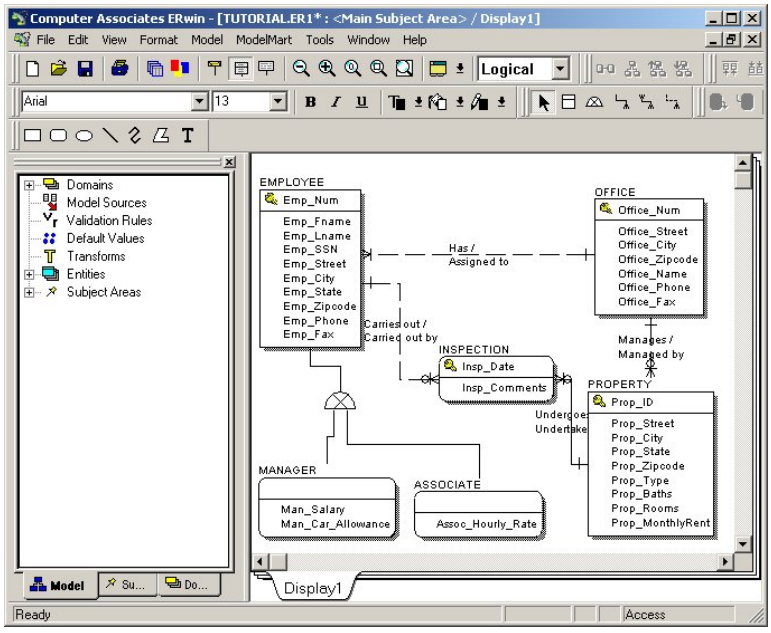



Imagen 5.2: Modelado con ERWin

- **Microsoft Visio**: Es la herramienta oficial de Microsoft para realizar todo tipo de diagramas [ver imagen 5.3].

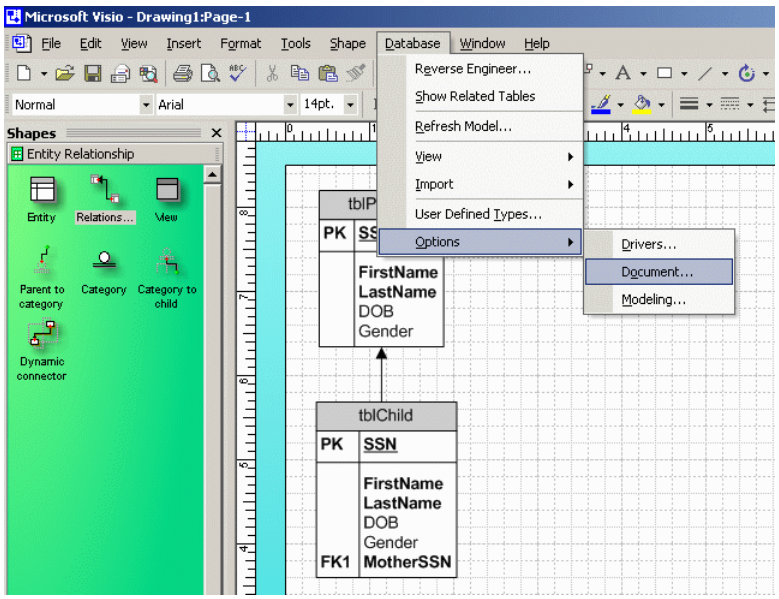



Imagen 5.3: Modelado con Microsoft Visio

- **Dia**: es una herramienta totalmente gratuita que permite realizar la mayoría de diagramas. Cuenta con una importante comunidad de usuarios que mejoran la aplicación mediante plugins [ver imagen 5.4].

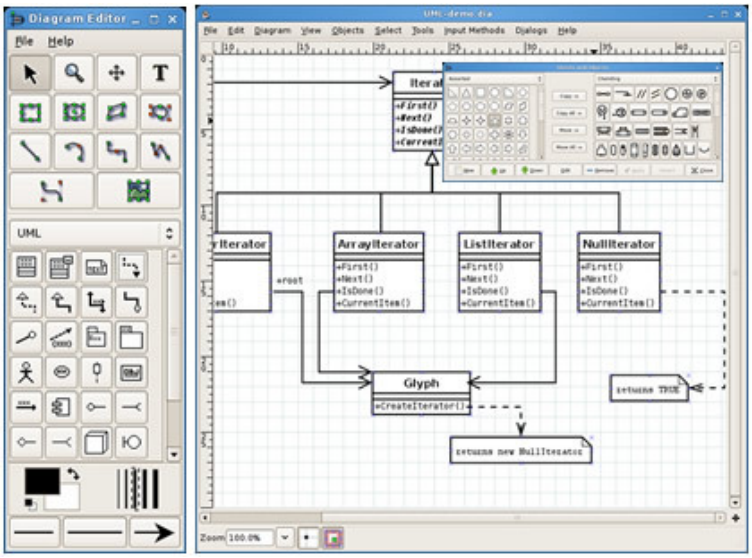



Imagen 5.4: modelado con Dia

Dependiendo de los recursos y de la experiencia de cada usuario se pueden realizar los diagramas Entidad-Relación en cualquiera de las opciones existentes en el mercado. Nosotros optaremos por Dia, ya que es una herramienta totalmente gratuita y que se acopla a nuestras necesidades.

Para realizar los diagramas debemos de fijarnos en dos recursos que nos proporciona Openbravo en su Wiki: el modelo de base de datos y el modelo de entidades.

### 5.5.3.- Modelo de base de datos

En este documento se almacena un listado de paquetes [ver imagen 5.5]. Cada paquete representa una funcionalidad en el ERP y está compuesto por una serie de tablas. Las tablas están relacionadas entre sí, ya que su objetivo es conseguir un propósito común: la funcionalidad final del paquete.

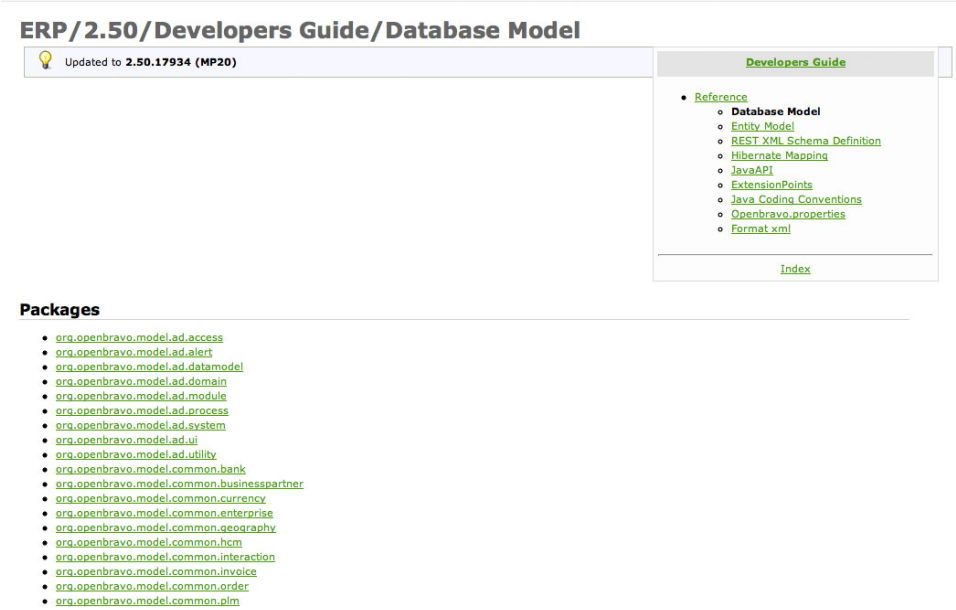



Imagen 5.5: Listado del modelo de base de datos

Partiendo de este documento ya es posible generar un diagrama Entidad-Relación básico. Basta con encontrar el paquete que cumple con la funcionalidad que buscamos y ver qué tablas contiene [ver imagen 5.6].

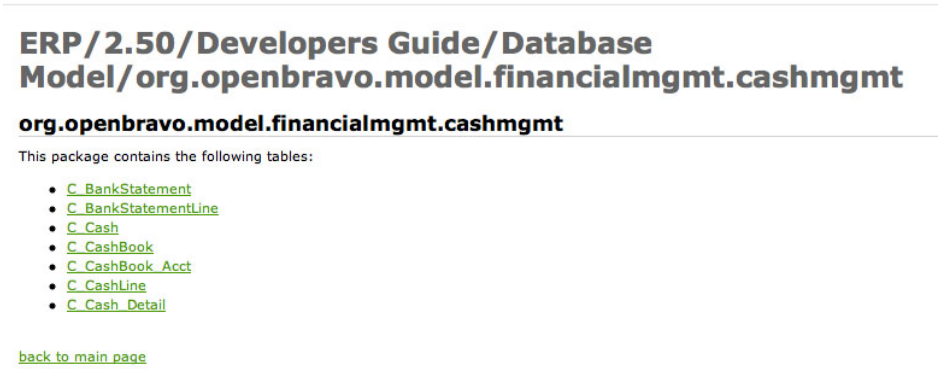



Imagen 5.6: Tablas pertenecientes a un paquete

Si hacemos clic en cualquiera de las tablas se nos abrirá una página Web con todos los campos de la tabla y su descripción.

### 5.5.4.- Modelo de Entidad

Este documento es una referencia para representar el modelo de entidad. Este modelo se usa por la Capa de Acceso a Datos (DAL) y los servicios Web REST.

En esta página observaremos una tabla con todas las entidades de OpenbravoERP 2.50 [ver imagen 5.7]. La tabla muestra esta información:

- Nombre de la entidad: es un nombre único en la base de datos y se corresponde al almacenado en el Diccionario de Aplicaciones.
- Nombre de la tabla: nombre de la tabla modelada por la entidad.
- Nombre de la clase: nombre de la clase Java que implementa esta entidad.

| Entity Name | Table Name | Java Class |
|---|---|---|
| ADAcctProcess | AD_AcctProcess | org.openbravo.model.ad.process.AcctProcess |
| ADAlert | AD_Alert | org.openbravo.model.ad.alert.Alert |
| ADAlertRecipient | AD_AlertRecipient | org.openbravo.model.ad.alert.AlertRecipient |
| ADAlertRule | AD_AlertRule | org.openbravo.model.ad.alert.AlertRule |
| ADAlertRuleTrl | AD_AlertRule_Trl | org.openbravo.model.ad.alert.AlertRuleTrl |
| ADAttachment | C_File | org.openbravo.model.ad.utility.Attachment |
| ADAuxiliaryInput | AD_AuxiliarInput | org.openbravo.model.ad.ui.AuxiliaryInput |
| ADCallout | AD_Callout | org.openbravo.model.ad.domain.Callout |
| ADChangeLog | AD_ChangeLog | org.openbravo.model.ad.system.ADChangeLog |
| ADClient | AD_Client | org.openbravo.model.ad.system.Client |
| ADClientModule | AD_ClientModule | org.openbravo.model.ad.module.ADClientModule |
| ADColumn | AD_Column | org.openbravo.model.ad.datamodel.Column |
| ADColumnAccess | AD_Column_Access | org.openbravo.model.ad.access.ColumnAccess |
| ADEPInstancePara | AD_EP_Instance_Para | org.openbravo.model.ad.process.ADEPInstancePara |
| ADEPProcedures | AD_EP_Procedures | org.openbravo.model.ad.process.Procedures |
| ADElement | AD_Element | org.openbravo.model.ad.ui.Element |
| ADElementTrl | AD_Element_Trl | org.openbravo.model.ad.ui.ElementTrl |
| ADErrorLog | AD_error_log | org.openbravo.model.ad.system.ErrorLog |
| ADExtensionPoints | AD_Extension_Points | org.openbravo.model.ad.process.ExtensionPoints |
| ADField | AD_Field | org.openbravo.model.ad.ui.Field |
| ADFieldGroup | AD_FieldGroup | org.openbravo.model.ad.ui.FieldGroup |
| ADFieldGroupTrl | AD_FieldGroup_Trl | org.openbravo.model.ad.ui.FieldGroupTrl |
| ADFieldTrl | AD_Field_Trl | org.openbravo.model.ad.ui.FieldTrl |
| ADFileType | AD_DataType | org.openbravo.model.ad.utility.FileType |
| ADForm | AD_Form | org.openbravo.model.ad.ui.Form |
| ADFormAccess | AD_Form_Access | org.openbravo.model.ad.access.FormAccess |
| ADFormTrl | AD_Form_Trl | org.openbravo.model.ad.ui.FormTrl |
| ADHeartbeatLog | AD_Heartbeat_Log | org.openbravo.model.ad.system.HeartbeatLog |

Imagen 5.7: Tabla del Modelo de Entidad

Esta tabla nos ayudará a relacionar cada entidad con su tabla en base de datos. Además, podremos localizar la clase Java que gestiona su información.

## 5.6.- Conclusiones

Este capítulo nos ha ayudado a comprender la envergadura de la base de datos de OpenbravoERP. Hemos aprendido que es necesaria cierta organización a la hora de diseñar la base de datos y que para ello emplearemos un estándar de codificación.

Además, hemos analizado la arquitectura empleada para acceder a la información. Para que en caso de querer ampliar la funcionalidad del ERP, tengamos en cuenta que disponemos de la Capa de Acceso a Datos y de los servicios Web REST, que incluyen una capa de validación y seguridad.

Y por último, hemos visto la documentación que proporciona OpenbravoERP para analizar su base de datos y poder elaborar diagramas Entidad-Relación.

Todo ello nos permitirá mas adelante comenzar a implementar nuevas funcionalidades sabiendo exactamente como interaccionar con la base de datos.

# 6.- Análisis del módulo para la ampliación de funcionalidades

El Diccionario de Aplicaciones es un módulo de configuración predeterminado del sistema donde se configura su funcionalidad. Aquí se definen los *metadatos* en donde se dan de alta los informes, procesos y formularios que se usan en la aplicación. También se definen todas las ventanas que se construyen automáticamente al compilar la aplicación, para ello se registran las tablas y columnas correspondientes de la base de datos y se define el diseño de la ventana.

Cualquier desarrollo que amplíe la funcionalidad de OpenbravoERP tiene que ser registrado y administrado desde el Diccionario de Aplicaciones. Es un módulo pasarela entre el sistema y el desarrollador.

Este módulo esta compuesto por un menú principal y tres submenús. Las opciones disponibles desde el menú se pueden distinguir en ventanas (icono lápiz) y procesos (icono tuerca). Las ventanas indican que esa opción va a gestionar uno o varios formularios de entrada y los procesos indican que la opción invoca una funcionalidad concreta del sistema. Los menús disponibles son:

- Menú principal: en él se sitúan las funcionalidades más utilizadas en el Diccionario de Aplicaciones [ver imagen 6.1], como por ejemplo el registros de módulos o la creación de ventanas.

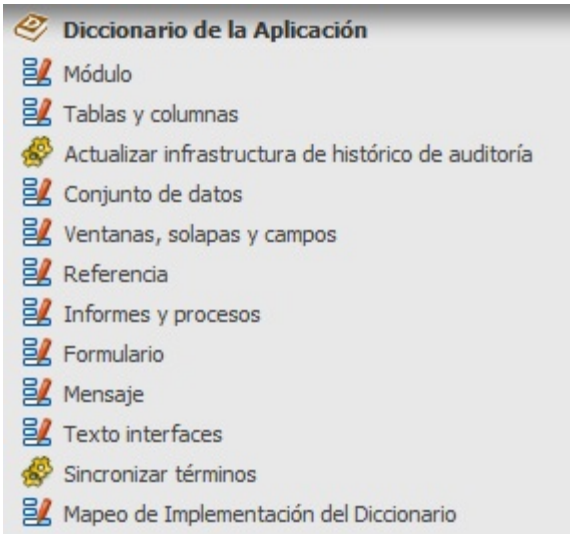



Imagen 6.1: Menú principal del Diccionario de la Aplicación

- Configuración: contiene todo tipo de opciones enfocadas al soporte del Diccionario de Aplicaciones [ver imagen 6.2].

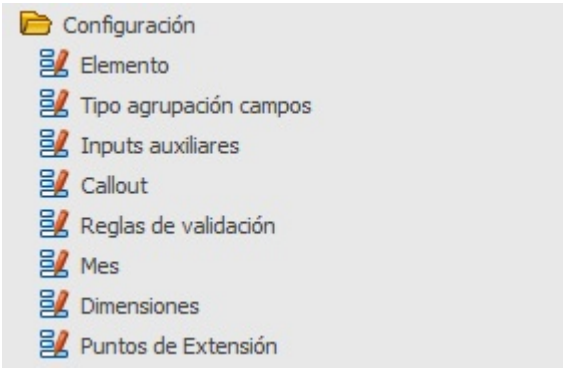



Imagen 6.2: Submenú Configuración

- Mantenimiento: en este submenú se almacenan las funcionalidades orientadas al mantenimiento de la base de datos [ver imagen 6.3].

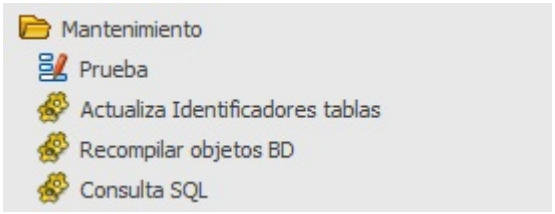



Imagen 6.3: Submenú Mantenimiento

- Test Automático: el último submenú está dedicado a la realización de tests o pruebas para los nuevos desarrollos en OpenbravoERP [ver imagen 6.4].

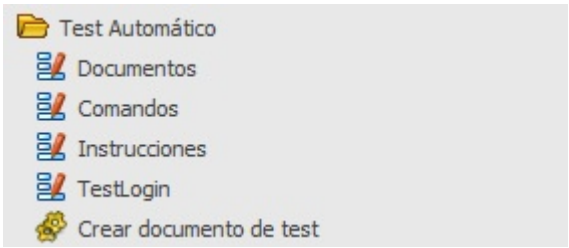



Imagen 6.4: Submenú Test Automático

A continuación vamos a estudiar la funcionalidad de cada uno de las opciones.

## 6.1.- Diccionario de la Aplicación

En este apartado se pueden localizar las opciones más comunes del Diccionario de Aplicaciones.

### 6.1.1.- Módulo

En este apartado se configuran los módulos del ERP. Es un apartado más orientado a su desarrollo ya que en él es posible configurar la descripción de cada módulo, sus dependencias a otros módulos, entre otras cosas.

Desde OpenbravoERP 2.50 se introduce el nuevo concepto de la *Modularidad*. Esto permite a los desarrolladores crear, empaquetar y distribuir Módulos de extensión para que los usuarios puedan descargárselos e instalárselos.

#### *6.1.1.1.- Introducción*

Un módulo de extensión es una pieza adicional y opcional que añade cierta funcionalidad al ERP. Varios ejemplos de módulos serían: informes nuevos, paquetes de traducción o ventanas adicionales.

Hay cuatro tipos de artefactos que se pueden incluir en un módulo de extensión:

- **Componentes del Diccionario de Aplicaciones**: *metadatos* que describen OpenbravoERP como por ejemplo: ventanas, mensajes, campos, etc.
- **Recursos software**: componentes del ERP que no se pueden expresar como *metadatos*, como por ejemplo: definiciones XML, clases Java, librerías, etc.
- **Datos de referencia**: información del negocio que son referencias de transacciones y que no suelen cambiar, como por ejemplo: gráficas, bancos, categorías de productos, etc.
- **Scripts de configuración**: ciertos archivos que configuran los módulos para que se comporten de cierta manera, como por ejemplo: ocultar pestañas, deshabilitar campos, etc.

La única limitación que presentan los módulos es que desde ellos no es posible modificar el WAD [ver capítulo 4] de la plataforma de OpenbravoERP.

Los módulos de extensión se pueden clasificar por:

- **Módulos**: contenedor básico. Dentro de un Módulo se pueden incluir todos los tipos de artefactos. Estos se utilizan para añadir nuevos elementos al ERP. Un Módulo no puede modificar elementos de otros módulos. Esto es así para evitar referencias cruzadas entre ellos.
- **Paquetes**: es un grupo de módulos. Se utilizan para facilitar la instalación en los casos que se requiera el uso de muchos módulos.
- **Plantillas**: es una combinación de un paquete y un fichero de configuración, el cuál puede cambiar el comportamiento de los *metadatos* de los módulos que estén dentro del paquete.

Los módulos son distribuidos en ficheros .obx, que contiene toda la su información comprimida en formato ZIP.

#### *6.1.1.2.- Creación*

Gracias al Diccionario de Aplicaciones vamos a poder crear módulos y realizar un seguimiento. Los pasos básicos para su creación son:

1. Registrar el módulo en el Diccionario de Aplicaciones y en el Repositorio Central.
2. Desarrollar los artefactos del módulo. Dependiendo de la funcionalidad deseada, éste incluirá uno o varios tipos de artefacto.
3. Compilar el módulo en un fichero .obx y publicarlo en el Repositorio Central.

Los pasos 1 y 3 son comunes a todos los tipos de módulos.

Hay que tener en cuenta que toda pieza de OpenbravoERP pertenece a un módulo, incluido en propio núcleo de la aplicación. Es posible hacer cambios en otros módulos (incluido el núcleo), pero no es recomendable hacerlo ya que la tarea de mantenimiento podría complicarse drásticamente.

### *6.1.1.3.- Ventanas*

En la ventana de Módulo vemos un listado de los módulos instalados y en funcionamiento [ver imagen 6.5]. Desde ella podemos crear un nuevo módulo haciendo clic en el botón "Nuevo registro". Este es un formulario inicial para introducir los datos básicos del módulo: versión, descripción, lenguaje, licencia, tipo de módulo, etc.

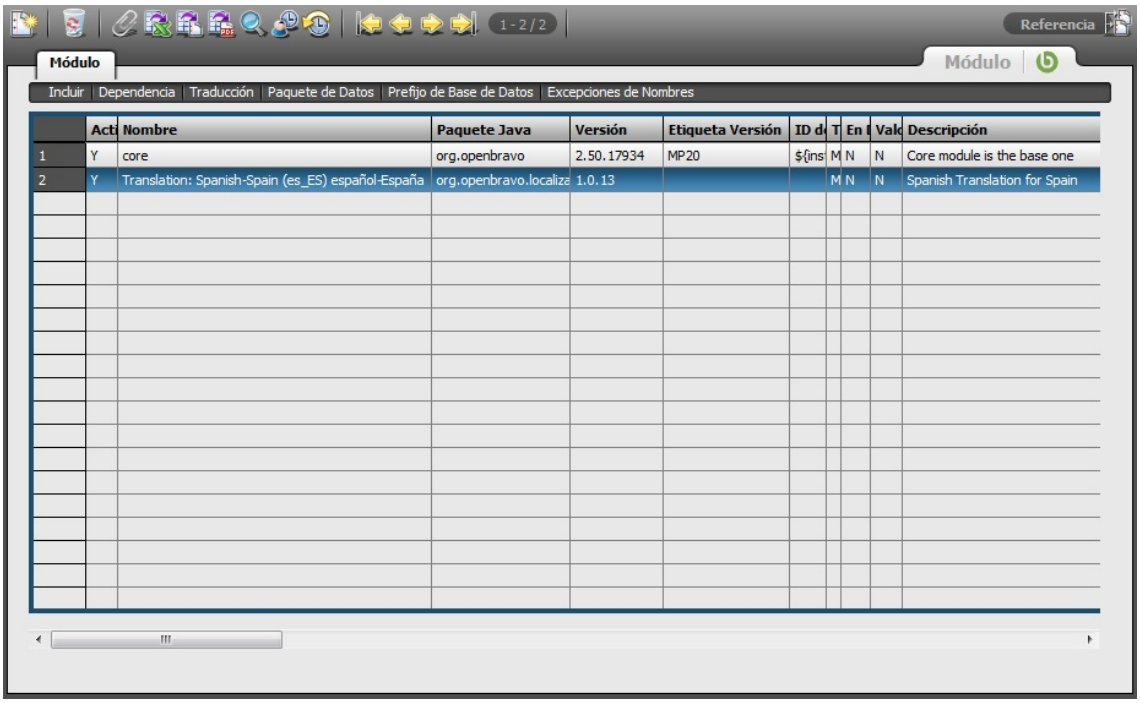



Imagen 6.5: Ventana principal de Módulo

Una vez creado, podemos gestionar otros tipos de datos mediante las pestañas adjuntas [ver imagen 6.7]. Nos darán la opción de configurar mediante las pestañas:

- **Incluir**: esta opción solo se utiliza para los módulos de extensión de tipo paquete o plantilla. En esta pestaña se configura los módulos que van a pertenecer al paquete.
- **Dependencia**: esta opción se utiliza solo para los módulos de extensión de tipo módulo. En esta pestaña se declaran los módulos de los que se va a depender. Un módulo depende de otro cuando alguno de sus procesos dependen de otros procesos que pertenecen a otros módulos. Por norma general, todos los módulos dependen del módulo núcleo o *core*.
- **Traducción**: esta opción se utiliza solo para los módulos cuya única función es la traducción. En esta pestaña se gestionan los módulos que van a estar implicados en la traducción.
- **Paquete de Datos**: esta opción configura el paquete Java asociado al módulo. Todo el código fuente debe estar incluido en el paquete Java que se declare en esta pestaña.
- **Prefijo de Base de Datos**: cuando el módulo contiene algún objeto relacionado con el esquema de base de datos es necesario configurar el prefijo del módulo.

Este prefijo debe acompañar a todos los objetos como por ejemplo el nombre de una tabla. Además, este prefijo debe ser único para todos los módulos existentes en el Repositorio Central. Openbravo gestiona esta comprobación de forma automática al registrar el módulo en el Repositorio Central.

- **Excepciones de Nombres**: esta opción se usa durante los procesos de actualización desde versiones más antiguas.

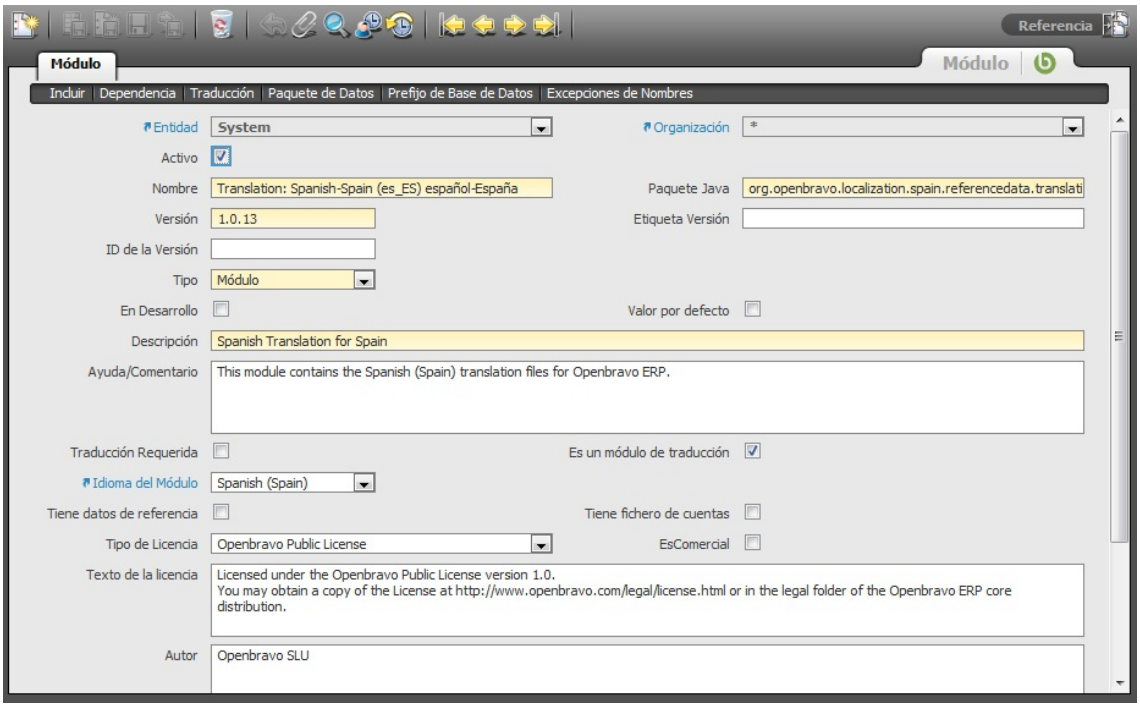



Imagen 6.7: Ventana de configuración de Módulo

Por último, cuando se termina de configurar el módulo, se debe registrar en el Repositorio Central haciendo clic en un botón de la página principal. Este registro sirve para que Openbravo pueda administrar todos los módulos existentes para su plataforma y coordinarlos mejor para su desarrollo. Gracias al Repositorio Central tenemos a nuestra disposición una gran biblioteca de módulos gratuitos y de pago que se pueden descargar con un simple clic.

#### *6.1.1.4.- Validación*

Cuando un módulo se termina y es empaquetado mediante el comando *ant package.module* se realizan una serie de comprobaciones para validarlo. Si se detecta algún error se detendrá la compilación. Las comprobaciones que se realizan son:

- El módulo debe depender del módulo núcleo o *core*, o depender de un módulo que ya dependa de él.
- La carpeta que contiene el código fuente del módulo debe coincidir con el paquete Java configurado en el Diccionario de Aplicaciones.
- El módulo debe tener configurada una licencia.
- Si el módulo es una plantilla entonces este debe depender del núcleo y la dependencia debe estar asignada como 'incluida'.
- Si el módulo incluye artefactos de la interfaz de usuario (todo lo que sea visible por el usuario en el navegador) entonces debe estar marcada la opción de 'Requiere traducción'.

### 6.1.2.- Tablas y columnas

Este apartado edita las tablas y columnas para que OpenbravoERP pueda acceder a la base de datos, además de controlar el acceso por roles.

#### *6.1.2.1.- Introducción*

OpenbravoERP almacena toda la información en una base de datos externa a la aplicación, sin embargo la base de datos se administra a través del sistema.

Para poder acceder a una tabla desde una ventana es esencial haber registrado la tabla en el Diccionario de Aplicaciones antes de usarla.

#### *6.1.2.2.- Creación*

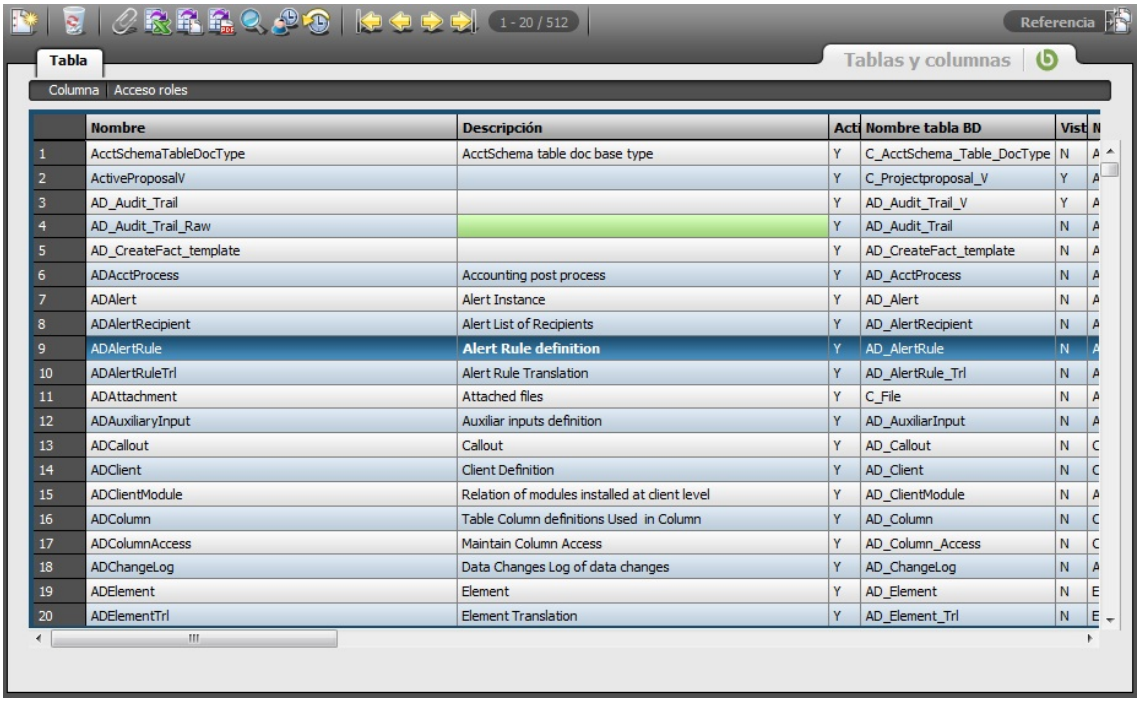
Para poder registrar una tabla en el Diccionario de Aplicaciones primero de todo hay que crearla en la base de datos. Este proceso se puede realizar mediante la ejecución de un *script* SQL o mediante el editor de tablas y columnas del cliente PostgreSQL, que en nuestro caso es PgAdmin III.

Una vez creada la tabla físicamente, ya se puede registrar la tabla mediante el formulario oportuno y asignarla a un módulo en desarrollo. Durante este proceso el Diccionario de Aplicaciones importará todos los datos de la base de datos para registrar sus columnas, nombres, tipos de datos (cadenas, números, valores boléanos), etc.

Por último, se podrá gestionar los permisos de acceso a las tablas mediante los roles disponibles.

#### *6.1.2.3.- Ventanas*

La primera ventana de todas [ver imagen 6.8] se visualiza un listado con todas las tablas registradas en el Diccionario de Aplicaciones. Estas tablas están asociadas cada una su tabla correspondiente en base de datos.




| | Nombre | Descripción | Acti | Nombre tabla BD | Vist | N |
|---|---|---|---|---|---|---|
| 1 | AcctSchemaTableDocType | AcctSchema table doc base type | Y | C_AcctSchema_Table_DocType | N | A |
| 2 | ActiveProposalV | | Y | C_Projectproposal_V | Y | A |
| 3 | AD_Audit_Trail | | Y | AD_Audit_Trail_V | Y | A |
| 4 | AD_Audit_Trail_Raw | | Y | AD_Audit_Trail | N | A |
| 5 | AD_CreateFact_template | | Y | AD_CreateFact_template | N | A |
| 6 | ADAcctProcess | Accounting post process | Y | AD_AcctProcess | N | A |
| 7 | ADAlert | Alert Instance | Y | AD_Alert | N | A |
| 8 | ADAlertRecipient | Alert List of Recipients | Y | AD_AlertRecipient | N | A |
| 9 | ADAlertRule | **Alert Rule definition** | Y | AD_AlertRule | N | A |
| 10 | ADAlertRuleTrl | Alert Rule Translation | Y | AD_AlertRule_Trl | N | A |
| 11 | ADAttachment | Attached files | Y | C_File | N | A |
| 12 | ADAuxiliaryInput | Auxiliar inputs definition | Y | AD_AuxiliarInput | N | A |
| 13 | ADCallout | Callout | Y | AD_Callout | N | C |
| 14 | ADClient | Client Definition | Y | AD_Client | N | C |
| 15 | ADClientModule | Relation of modules installed at client level | Y | AD_ClientModule | N | A |
| 16 | ADColumn | Table Column definitions Used in Column | Y | AD_Column | N | C |
| 17 | ADColumnAccess | Maintain Column Access | Y | AD_Column_Access | N | C |
| 18 | ADChangeLog | Data Changes Log of data changes | Y | AD_ChangeLog | N | A |
| 19 | ADElement | Element | Y | AD_Element | N | E |
| 20 | ADElementTrl | Element Translation | Y | AD_Element_Trl | N | E |

Imagen 6.8: Ventana principal de Tablas y columnas

Cuando se crea un nuevo registro [ver imagen 6.9] hay que configurar una serie de datos para completar la asociación de la tabla del sistema con la de base de datos. La información más importante es:

- Nombre de la tabla en el sistema.
- Nombre de la tabla en la base de datos.
- Paquete de datos: las tablas no se asignan directamente a un módulo. En su lugar es necesario indicar el paquete del módulo [ver apartado anterior]. Esto es necesario para que el DAL administre apropiadamente el empaquetamiento de los artefactos del módulo (entidades y ficheros XML).
- Nombre de la clase Java que va a contener toda la lógica de negocio.
- La ventana que va a visualizar los datos de la tabla para que se puedan administrar.

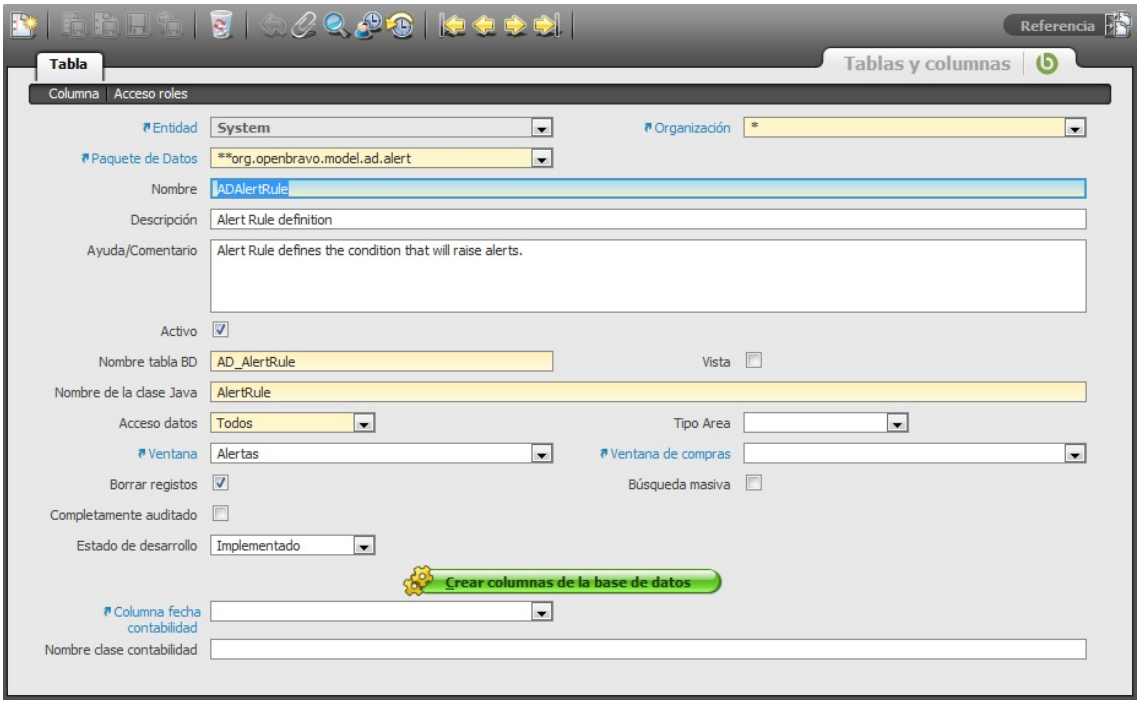
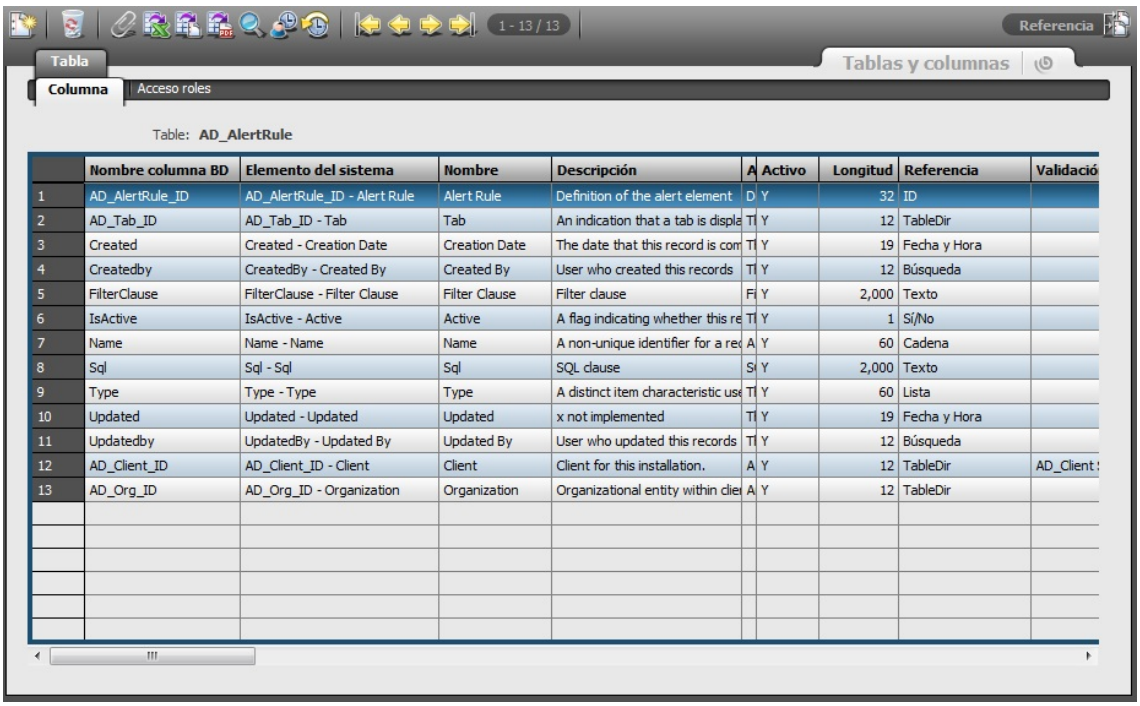



Imagen 6.9: Ventana de creación de una Tabla

Cuando se ha terminado de configurar la tabla, es posible activar la funcionalidad “Crear columnas de la base de datos”. Este proceso realiza una lectura de las columnas que contiene la tabla y las crea en el sistema automáticamente, rellenando todos los datos para su funcionamiento.

Como podemos ver en la imagen 6.10, en la pestaña “Columna” están registradas todas las columnas de la tabla. Mediante un formulario podemos modificar cualquier dato de la columna como por ejemplo la descripción, el tipo de dato, si está activa o no, etc.




| | Nombre columna BD | Elemento del sistema | Nombre | Descripción | A | Activo | Longitud | Referencia | Validació |
|---|---|---|---|---|---|---|---|---|---|
| 1 | AD_AlertRule_ID | AD_AlertRule_ID - Alert Rule | Alert Rule | Definition of the alert element | D | Y | 32 | ID | |
| 2 | AD_Tab_ID | AD_Tab_ID - Tab | Tab | An indication that a tab is displa | Tl | Y | 12 | TableDir | |
| 3 | Created | Created - Creation Date | Creation Date | The date that this record is com | Tl | Y | 19 | Fecha y Hora | |
| 4 | Createdby | CreatedBy - Created By | Created By | User who created this records | Tl | Y | 12 | Búsqueda | |
| 5 | FilterClause | FilterClause - Filter Clause | Filter Clause | Filter clause | Fi | Y | 2,000 | Texto | |
| 6 | IsActive | IsActive - Active | Active | A flag indicating whether this re | Tl | Y | 1 | Sí/No | |
| 7 | Name | Name - Name | Name | A non-unique identifier for a rec | A | Y | 60 | Cadena | |
| 8 | Sql | Sql - Sql | Sql | SQL clause | St | Y | 2,000 | Texto | |
| 9 | Type | Type - Type | Type | A distinct item characteristic use | Tl | Y | 60 | Lista | |
| 10 | Updated | Updated - Updated | Updated | x not implemented | Tl | Y | 19 | Fecha y Hora | |
| 11 | Updatedby | UpdatedBy - Updated By | Updated By | User who updated this records | Tl | Y | 12 | Búsqueda | |
| 12 | AD_Client_ID | AD_Client_ID - Client | Client | Client for this installation. | A | Y | 12 | TableDir | AD_Client |
| 13 | AD_Org_ID | AD_Org_ID - Organization | Organization | Organizational entity within clie | A | Y | 12 | TableDir | |

Imagen 6.10: Listado de las columnas de una tabla

Una vez gestionada toda la información sobre la tabla y sus columnas, mediante las pestaña “Acceso roles” se nos da la posibilidad de restringir el acceso a la tabla tan solo a una serie de roles o perfiles de usuario. Esto puede ser útil cuando para un módulo solo tienen acceso el departamento de contabilidad o los administradores.

### 6.1.3.- Conjunto de datos

Un Conjunto de datos o *dataset* es una flexible herramienta para actualizar o añadir información en la base de datos para que se complemente con la instalación de un módulo.

#### *6.1.3.1.- Introducción*

El Conjunto de datos permite definir configurar diferentes tablas y exportar sus datos en un paso. Los Conjuntos de datos son útiles para la administración de las Referencias de un Módulo.

Un claro ejemplo sería un módulo que necesite actualizar la base de datos de impuestos para un país concreto. Los datos de Referencia serían los impuestos y estos se deberían de añadir en la tabla de impuestos de la base de datos para que el módulo funcione correctamente.

Mediante la ventana del Conjunto de datos se puede definir los datos de referencia para los módulos. Los datos de Referencia son empaquetados, distribuidos e instalados juntos en el programa mediante la activación del módulo

#### *6.1.3.2.- Creación*

Para crear un Conjunto de datos primero de todo hay que crear un registro mediante el formulario principal y asociarlo a nuestro módulo en desarrollo. Si el módulo no está en desarrollo no aparecerá en la lista que nos proporciona el Diccionario de Aplicaciones.

Una vez creado iremos seleccionando una a una todas las tablas que estén implicadas en el proceso de exportación de datos. Para cada tabla indicaremos qué información deseamos obtener mediante la configuración de dos campos:

- Cláusula SQL/HQL: en este campo se introduce la consulta SQL/HQL con la que obtendremos la información filtrada para la tabla.
- Campos: en este listado indicaremos los campos que queremos obtener. Es posible que no nos interese toda la información de los registros de la tabla.

Con esto ya estaremos listos para realizar la exportación del Conjunto de datos.

El contenido del Conjunto de datos se define por las tablas definidas en la misma ventana. Primero se define que tablas se van a exportar y luego que columnas de cada tablas son ejecutadas.

### *6.1.3.3.- Ventanas*

La primera ventana que nos vamos a encontrar al entrar en el Conjunto de datos es un listado con todos los Conjuntos que existen ya en el sistema [ver imagen 6.11]. Desde aquí podemos crear un Conjunto de datos nuevo haciendo clic en “Nuevo registro”.

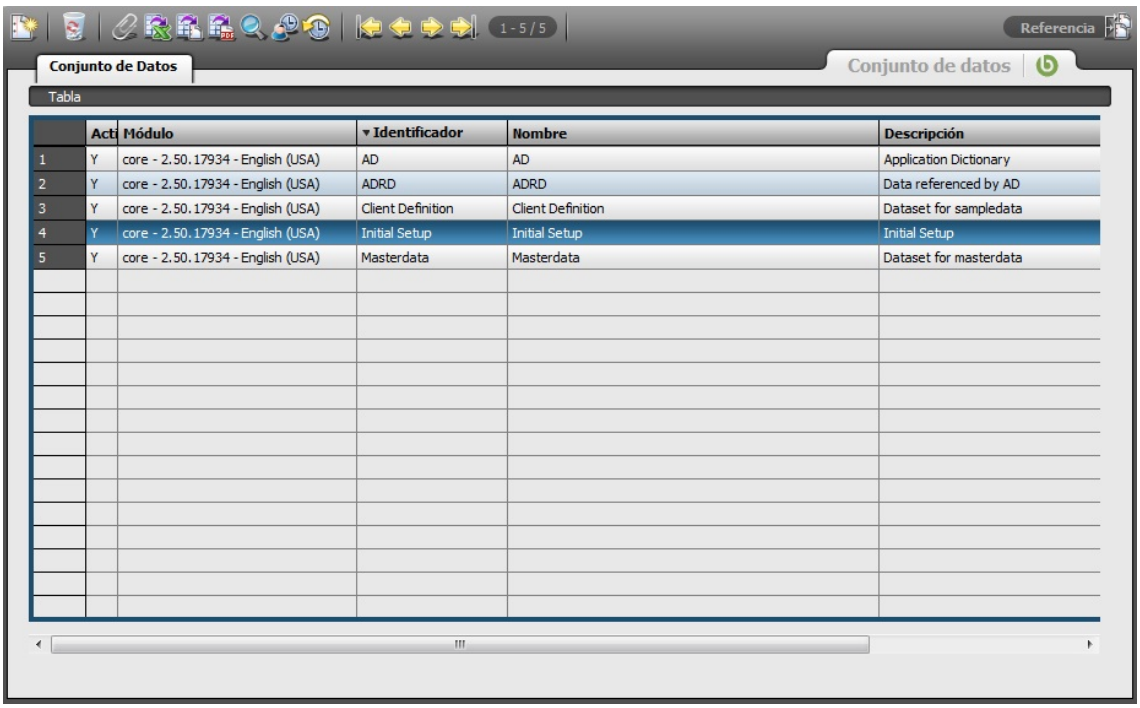



Imagen 6.11: Listado del Conjunto de datos

Cuando creemos un nuevo Conjunto de datos deberemos de configurar principalmente su nombre, el módulo al que pertenece y su accesibilidad en la aplicación [ver imagen 6.12].

Si el campo “Permitir exportar” se activa aparecerá el botón “Exportar”. Cuando se hace clic en este botón los datos se exportan al directorio */modulos/carpeta-del-modulo/* al que pertenezca el Conjunto de datos.

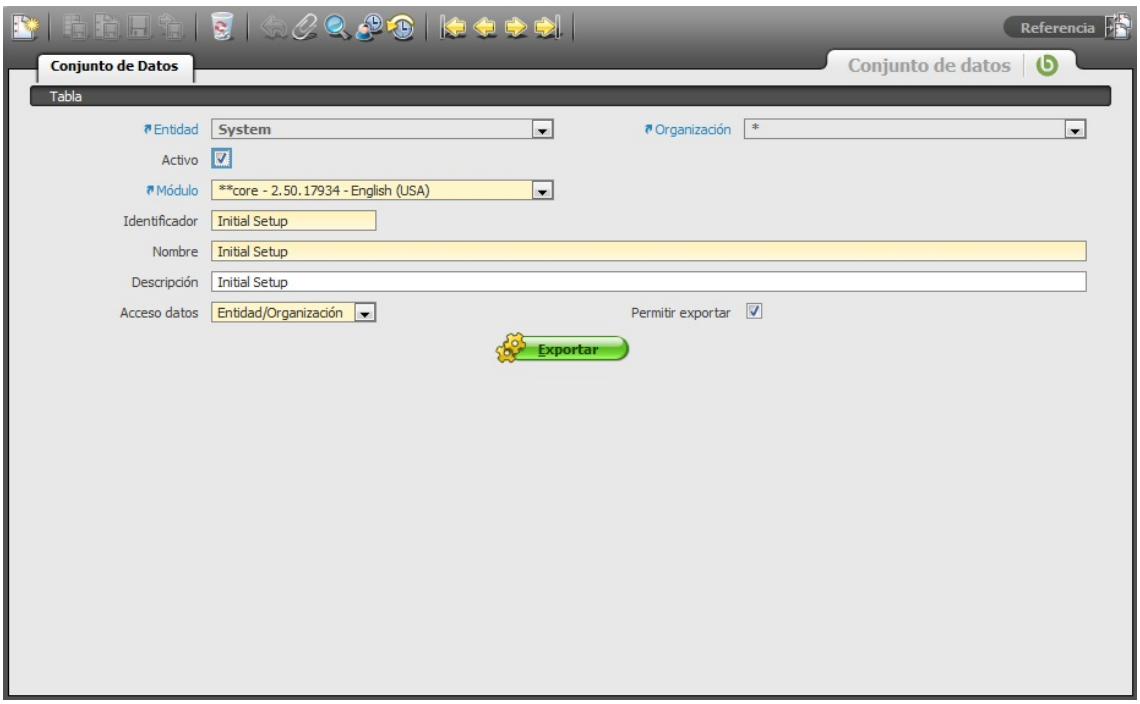



Imagen 6.12: Creación de un Conjunto de datos

Una vez guardado el Conjunto de datos se debe de configurar las tablas de la base de datos de las cuales se va a exportar información [ver imagen 6.13]. Las tablas disponibles son filtradas por el *Data Access Level*, ya que no todas pueden ser incluidas en un Conjunto de datos.

La tabla del Conjunto de Datos define qué datos de una tabla se exportarán. Esta selección se realiza mediante una consulta HQL/SQL, que es completamente editable.

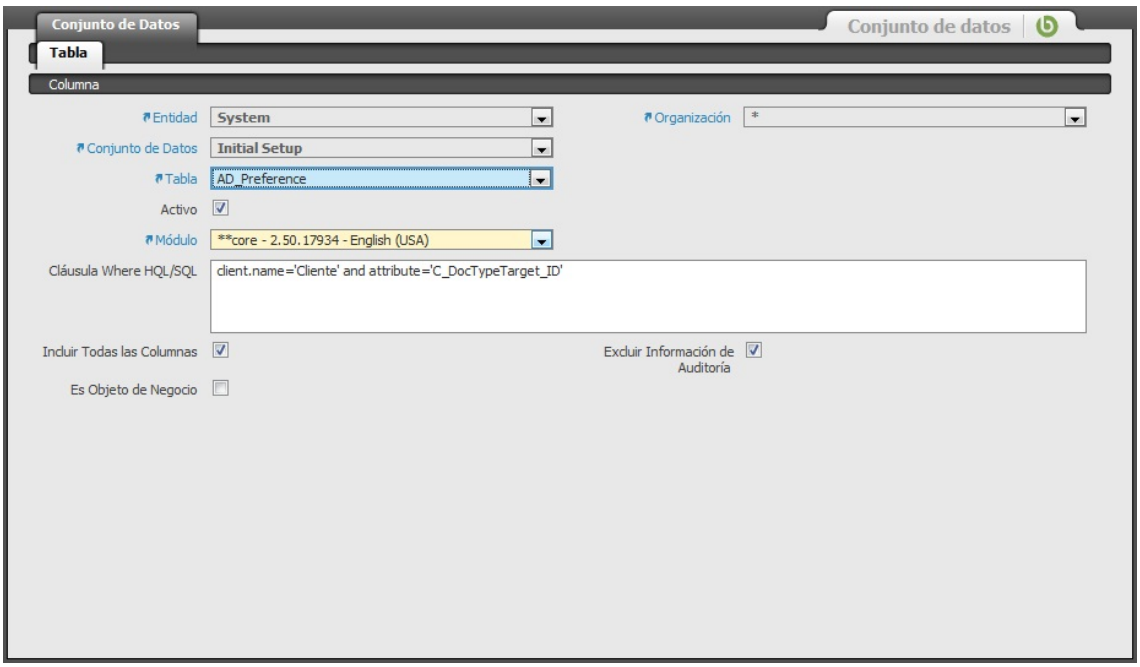



Imagen 6.13: Configuración de la tabla

Aspectos a tener en cuenta:

- Una tabla de un Conjunto de datos tiene asignado un módulo. Esto hace posible crear nuevas tablas en otros Conjuntos de datos ya existentes pero que tengan asignado nuestro módulo.
- La consulta SQL es en realidad una consulta HQL (*Hibernate Query Languaje*). Las propiedades que se pueden usar en esta consulta son las propiedades de la tabla del Conjunto de datos.
- El campo "Incluir todas las columnas" sirve para que, al exportar, todas las columnas sean exportadas excepto las que están listadas en la pestaña "Columnas".
- El campo "Excluir información de auditoría" solo es relevante cuando se activa el anterior campo.
- El campo "Es objeto de negocio" sirve para que, al exportar, se incluyan todos los registros asociados al principal. Por ejemplo, si tenemos configurado un Conjunto de datos con la tabla C_Order (pedidos), los datos de la tabla C_OrderLines (líneas de los pedidos) serán también exportados.

**Consulta HQL**

El campo "Cláusula Where HQL/SQL" es completamente editable y dota de una gran flexibilidad a los Conjuntos de datos. Aunque en su nombre indique SQL, es en realidad una consulta HQL pura. La consulta HQL (*Hibernate Query Languaje*) nos sirve para obtener información de la base de datos de OpenbravoERP pasando por la Capa de Acceso a Datos (DAL). Esta capa va a interactuar entre la aplicación y la base de datos y una de las formas de comunicarnos es mediante el lenguaje HQL.

Estas consultas tienen dos parámetros estándar:

- ClientID: su valor va a ser siempre igual al usuario que esté exportando los datos.
- ModuleID: su valor es igual al módulo del cual se esté exportando los datos.

Por último, la pestaña "Columna" actúa de filtro y nos servirá para indicar las columnas que deseamos exportar de la tabla. En caso de que el campo "Incluir todas las columnas" de la tabla esté activo, las columnas declaradas aquí serán excluidas de la exportación.

### 6.1.4.- Ventanas, pestañas y campos

En este apartado OpenbravoERP permite gestionar una de las partes visuales más importantes del sistema. Gran parte visual del ERP está basada en ventanas desarrolladas desde el Diccionario de Aplicaciones. Una ventana estándar se puede definir completamente desde aquí.

#### *6.1.4.1.- Introducción*

Las ventanas, las pestañas y los campos son contenedores de información. Su función básica es la de ofrecer un soporte visual para la información que contienen. En la imagen 6.14 podemos ver un ejemplo de todas ellas y su funcionalidad. Más concretamente:

- **Ventanas**: son los contenedores de las pestañas. Su propósito principal es agrupar un conjunto de pestañas que tengan una funcionalidad relacionada. Las ventanas se pueden enlazar al menú de la aplicación para poder acceder desde él.
- **Pestañas o solapas**: están colocadas dentro de las ventanas y se pueden ordenar jerárquicamente. Cada pestaña está enlazada a una única tabla del Diccionario de Aplicaciones. Las pestañas pueden contener una serie de campos.

- **Campos**: están colocados dentro de las pestañas. Cada campo está asociado a una columna de la tabla a la que está enlazada la pestaña.

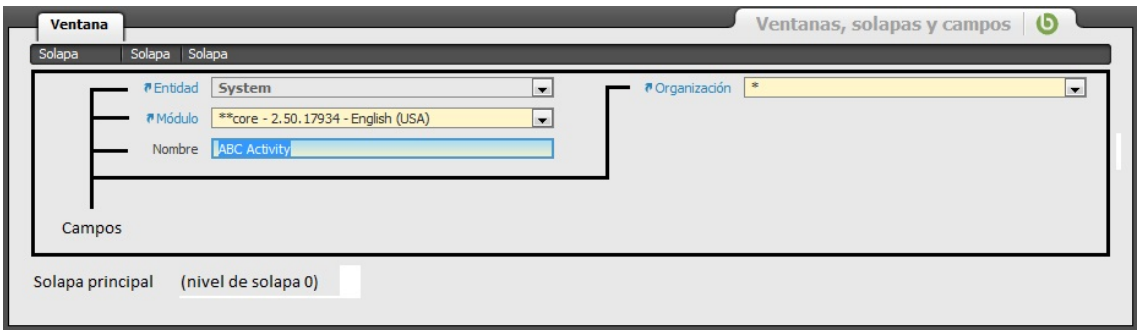



Imagen 6.14: Clasificación de contenedores

### *6.1.4.2.- Creación*

Cuando se está desarrollando un módulo nuevo es muy probable que necesitemos el soporte de las ventanas, pestañas y campos para visualizar toda la información relevante al módulo.

Dependiendo de nuestras necesidades, crearemos nuevas ventanas y pestañas. Aunque si no es necesario, se puede utilizar las ya existentes. Esto es posible gracias a que el Diccionario de Aplicaciones tiene enlazados cada ventana, pestaña y campo a su correspondiente módulo. Toda la información está muy bien gestionada y se conoce cada detalle a qué módulo pertenece.

Los tres contenedores siguen una regla común en su creación. Los tres están enlazados a un módulo. Por defecto, cuando añades una pestaña a una ventana esta se enlazará al módulo de su ventana y cuando añades un campo a una pestaña, esta se enlazará al módulo de su pestaña.

A la hora de crear un formulario empezaremos configurando la ventana, que es el contenedor principal. Luego, crearemos todas las pestañas necesarias, teniendo en cuenta que cada pestaña será un formulario enfocado a una funcionalidad concreta. El Diccionario de Aplicaciones entiende que esa funcionalidad debe estar muy relacionada con una tabla en base de datos, y es por ello que cada pestaña se centra en la administración de una tabla. Por último, crearemos los campos necesarios para cada pestaña. Cada campo recuperará una columna concreta de la tabla.

Después de realizar los cambios es necesario recompilar el sistema (comando *ant smartbuild*). Durante este proceso el WAD genera automáticamente el código Java, XSQL, HTML y XML para esa ventana y luego se compila. Esto significa que es posible implementar una completa definición de una ventana dentro del Diccionario de Aplicaciones sin necesidad desarrollar código manualmente.

### *6.1.4.3.- Ventanas*

El Diccionario de Aplicaciones nos va a permitir tanto crear nuevas ventanas, pestañas y campos como editar los ya existentes para adjuntarles nuevas pestañas y campos.

**Ventanas**

La primera ventana que visualizaremos al entrar en este apartado es un listado de las ventanas de toda la aplicación. Al entrar en una de ellas se podrá configurar los datos básicos como el nombre, el módulo al que pertenece o el tipo de ventana [ver imagen 6.15].

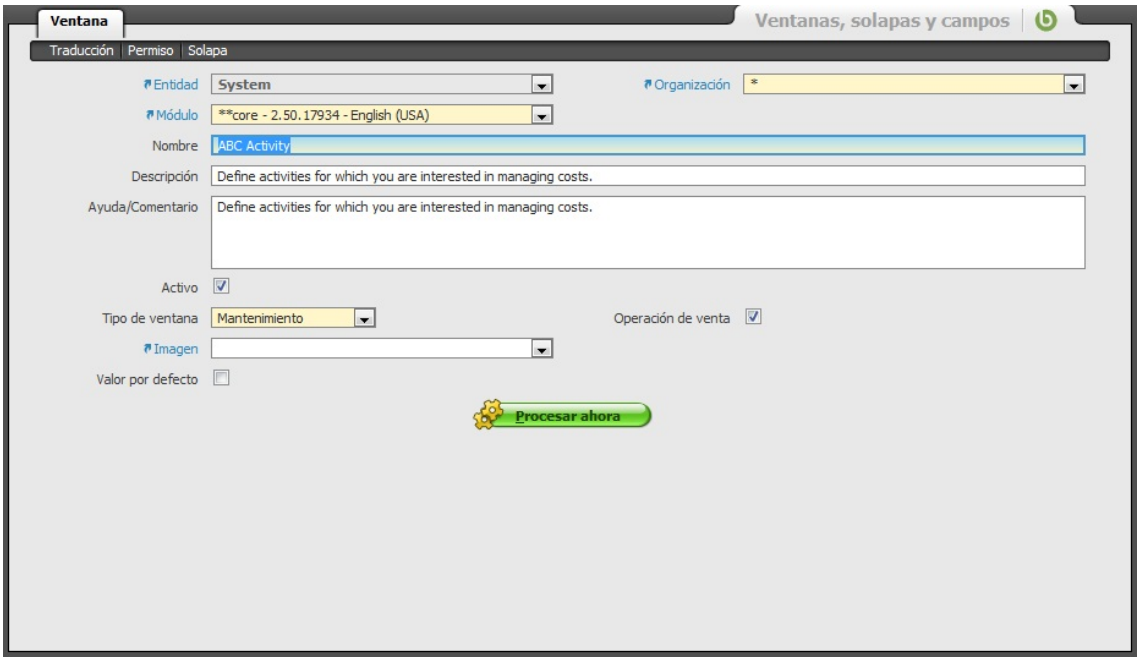



Imagen 6.15: Configuración de una ventana

Desde la pestaña de “Traducción” se configura si existe una traducción para la ventana y en qué lenguaje. Realmente la traducción va a afectar a los únicos campos visibles, el nombre de la ventana y su descripción.

La siguiente pestaña, “Permiso”, es un listado de los roles a los que se les permite el acceso a esta ventana. Su configuración es muy parecida a la vista en el apartado de Tablas y columnas.

Por último, en “Solapa” vamos a poder configurar todas las pestañas que van a estar asociadas a esta ventana.

**Pestañas**

Las pestañas son el siguiente contenedor más pequeño. Pueden haber más de una en la misma ventana y cada una de ellas está asociada a una tabla del Diccionario de Aplicaciones. La tabla debe pertenecer al mismo módulo al que pertenece la pestaña.

Cuando hay mas de una, las pestañas son visualizadas jerárquicamente, mediante una estructura tipo *árbol*. Esto significa que pueden haber subpestañas, siendo unas hijas de otras pestañas, o incluso que puedan haber distintas pestañas en el mismo nivel.

La jerarquía se especifica mediante dos campos [ver imagen 6.16]:

- **Nivel solapa**: indica el nivel en la jerarquía, siendo 0 la raíz del árbol. Para cada ventana solo puede haber asignada una solapa de nivel 0.

- **Secuencia**: es el número que define el orden de las pestañas que se visualizan. Las pestañas se ordenan ascendentemente, las menores a la izquierda y las mayores a la derecha.

La combinación de estos dos valores resulta en la jerarquía de visualización de las pestañas.

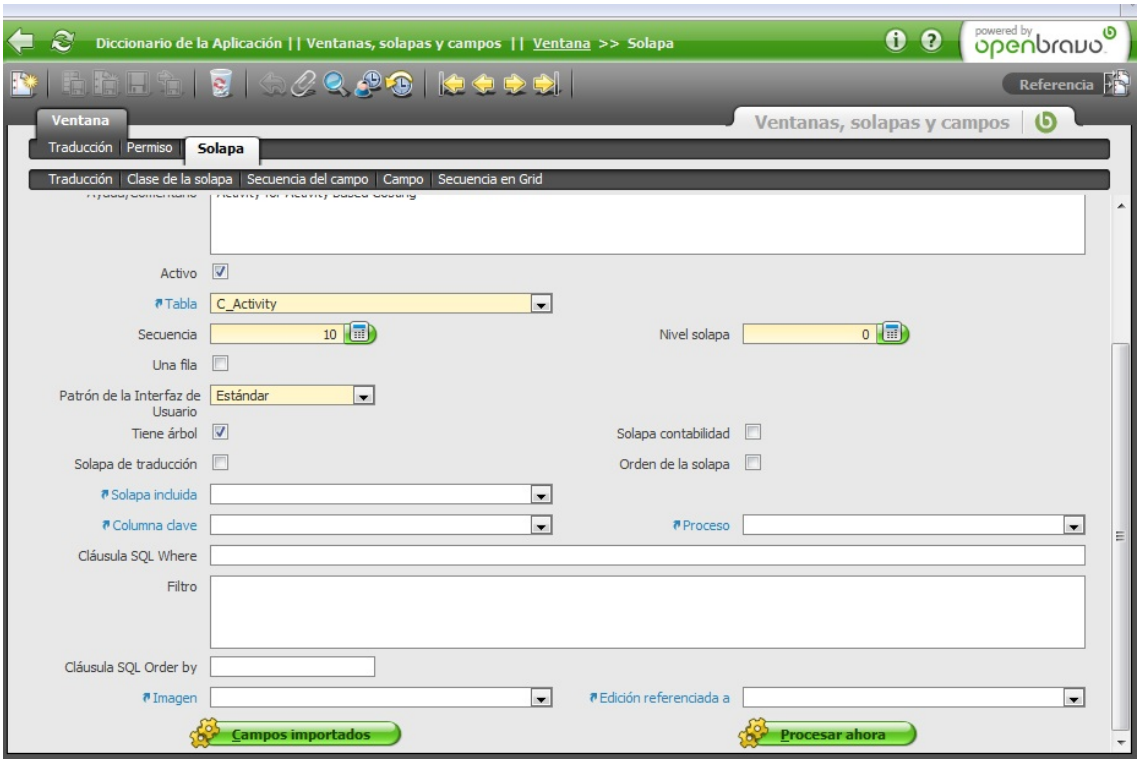



Imagen 6.16: Creación de una pestaña o solapa

En la “Clase de la solapa” se define qué clase Java va a administrar la funcionalidad del formulario. En un principio no es necesario implementar código Java ya que lo genera automáticamente el WAD. Pero es posible que para implementar la nueva funcionalidad sea necesario tocar código Java.

En la “Secuencia en Grid” aparecen dos listados con los campos de la pestaña actual. Estos listados filtran los campos que se vayan a visualizar y los que no, dependiendo del listado al que pertenezcan. Luego, en el listado que se visualiza, podemos ordenar los campos para controlar el orden de aparición en la ventana.

Cuando se edita la secuencia de campos el sistema garantiza que no se cambiarán la secuencia de los campos que no están en el módulo de desarrollo. De esta forma, si se añade un nuevo campo en una pestaña de otro módulo, la secuencia original de los campos no se verá afectada. Si se intenta realizar este cambio manualmente OpenbravoERP no te permitirá ya que esa información está en un módulo que no está en desarrollo. Esto también se aplica cuando se está añadiendo pestañas a una ventana de otro módulo.

Por último, en la pestaña “Campos”, se configuran todos los campos que se visualizarán esta pestaña.

**Campos**

Los campos son los contenedores más pequeños. Cada campo está asociado a una columna de la tabla a la que enlaza la pestaña. Entonces se visualiza y se permite la edición del campo.

En un principio es posible crear todos los campos de la tabla uno a uno, pero esta tarea es bastante laboriosa. Para agilizarla, en la pestaña principal de “Solapa” está el botón “Campos importados” cuya función es leer la tabla del Diccionario de Aplicaciones que está enlazada a la pestaña y obtener todas sus columnas. Cada columna se asocia a un campo distinto.

Cuando termina el proceso de importación, se podrá ver un listado de todos los campos creados. Al hacer clic en cualquiera de ellos aparecerá una ventana [ver imagen 6.17] con toda la información del campo rellenada: nombre, columna asociada, tipo del campo, etc.

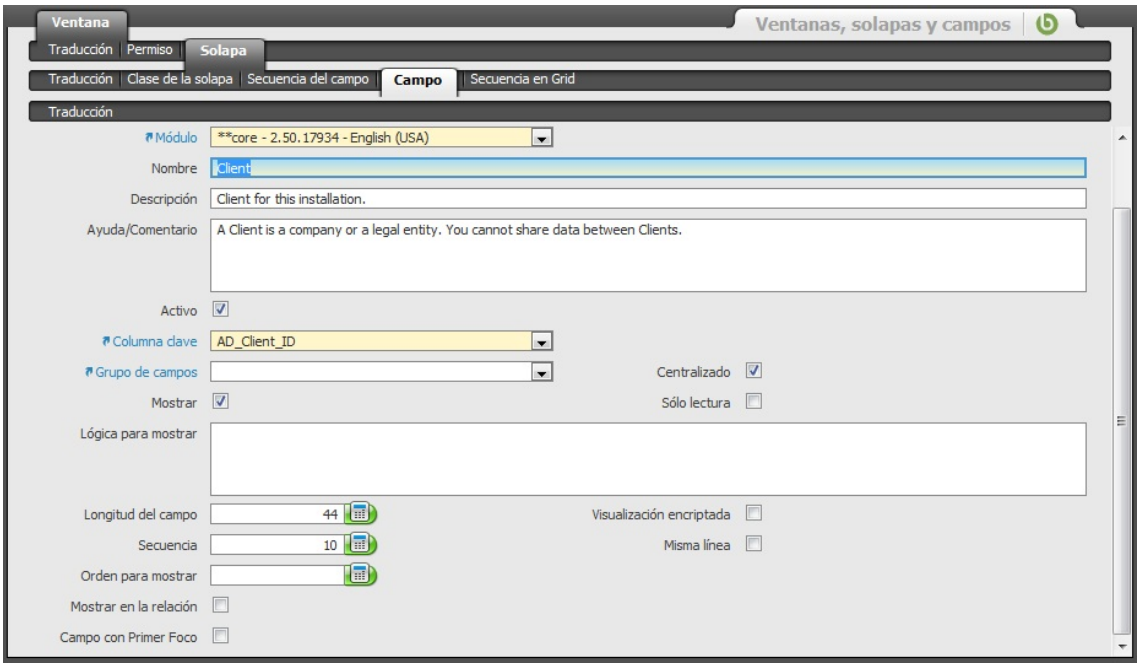



Imagen 6.17: Ventana de edición de un campo

La forma de visualizar un campo viene determinada por la referencia a la que está asociada la columna. Si la columna es de tipo bolean, se asociará a una referencia checkbox. Si la columna es de tipo *texto*, se asociará a una referencia campo de entrada de texto.

Algunos de los campos más relevantes a la hora de configurar un campo son:

- **Lógica de visualizado**: permite visualizar u ocultar un campo dependiendo del valor de otros campos. Es una expresión dinámica.
- **Mantenimiento Centralizado**: permite que el Diccionario de Aplicaciones pueda realizar un mantenimiento de este campo de forma automática.
- **Grupo de campos**: el campo puede asignarse a un grupo de campos. Sirve para que a la hora de visualizar los campos se dibuje un separador especial para recalcar el grupo. Los grupos de campos se crean en el Diccionario de Aplicaciones, en el apartado de configuración.

### 6.1.5.- Referencia

Las referencias son una parte crucial en OpenbravoERP ya que definen cómo se van a almacenar los datos. Sirven tanto para definir la información en la base de datos como para saber cómo se va a visualizar esa información en la Interfaz Gráfica de Usuario.

#### *6.1.5.1.- Introducción*

Las referencias se utilizan en OpenbravoERP con dos fines:

- **Definir el tipo de información que se almacena en una columna**: dependiendo de la referencia que tenga la columna, esta soportará diferentes tipos de información, como por ejemplo: valores numéricos, texto plano, links a otras columnas, etc.
- **Definir cómo se visualizarán los campos que están asociados a una columna**: dependiendo del tipo de información que almacena una columna su representación gráfica es distinta. Por ejemplo, una columna de tipo texto se representa como una caja de texto en la que el usuario puede introducir información y una columna binaria o *booleana* (que solo acepta dos valores posibles: 0 o 1) se representa como un *checkbox*.

#### *6.1.5.2.- Creación*

En la base de datos cada columna de cada tabla tiene asignada un tipo de datos que puede almacenar. A su vez, cada columna en el Diccionario de Aplicaciones tiene una única referencia.

Las referencias se asignan en **Diccionario de Aplicaciones > Tablas y columnas > Tabla > Columna** utilizando el campo *Referencia* para definirla y el campo *Referencia clave* para la subreferencia, en caso de que sea necesaria.

**Tipos de referencia**

Hay dos tipos básicos de referencias:

- **Referencias de base**: se asocian directamente a una columna. Ejemplos: fecha, precio, lista, etc.
- **Subreferencias**: sirven para completar la definición de la referencia de una columna. Las referencias base definidas en el núcleo de OpenbravoERP que necesitan una subreferencia son: *Lista*, *Búsqueda* y *Tabla*. Por ejemplo, para indicare que una columna es una búsqueda de Terceros es necesario definir la referencia base como *Búsqueda* y la subreferencia como *Búsqueda Terceros*.

**Referencias base del núcleo**

La siguiente lista describe las referencias base definidas en el núcleo de OpenbravoERP.

- **Referencias numéricas**: Entero, Número, Cantidad y Cantidad General son usados para albergar valores numéricos. Cuando definimos una referencia de este tipo es posible configurar un valor mínimo y otro máximo. En la imagen 6.18 vemos la representación visual de un campo numérico. Los formatos de los números vienen definidos y se pueden modificar en el fichero *config/Format.xml*.

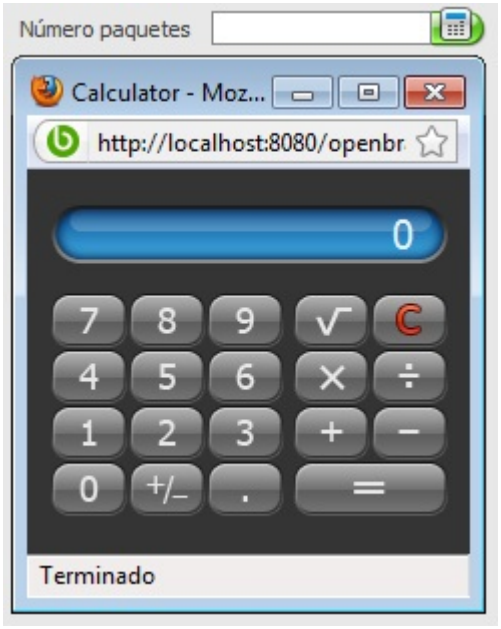



imagen 6.18: Campo numérico y calculadora adjunta

- **Precio**: es similar a las referencias numéricas pero tiene el objetivo de almacenar cantidades monetarias. Esta referencia no permite definir valores máximo y mínimo.
- **String, Text y Memo**: son las referencias utilizas para los textos. La diferencia entre ellos es la longitud máxima del texto que pueden almacenar. *String* es el que admite menos longitud y se representa con una caja de texto. *Text y memo* se representan con un área de texto de diferentes números de línea.
- **Link**: esta referencia se utiliza para representar enlaces URL (*Uniform Resource Locator*), como por ejemplo: http://www.openbravo.com.
- **Date, Time y DateTime**: son usados para almacenar horas y fechas. *Date* muestra la fecha sin horas, *Time* muestra la hora sin la fecha y *DateTime* muestra los dos a la vez. En la imagen 6.19 vemos un campo de tipo fecha y la ventana para poder elegir más cómodamente su valor. El formato para su visualización se puede configurar en el fichero *Openbravo.properties*.

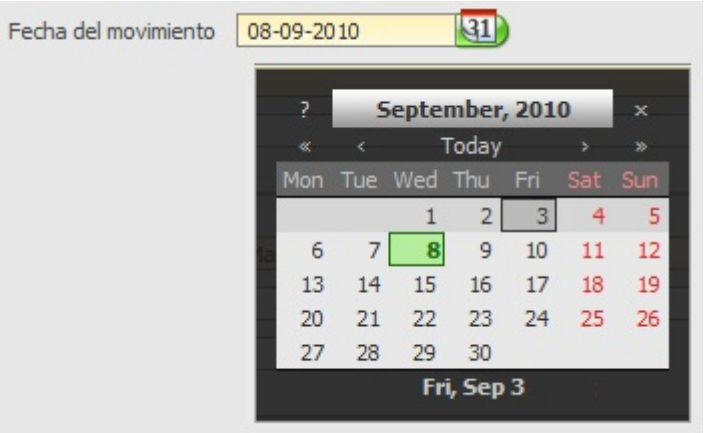



Imagen 6.19: Campo de tipo fecha

- **Si/No**: estas referencias son usadas para aceptar valores boléanos (si o no). Son visualizadas mediante un checkbox. Si no está marcado contiene el valor “no” y si lo está contiene el valor “si” [ver imagen 6.20].


Imagen 6.20: Referencia booleana

- **Botón**: el valor de las columnas que tienen esta referencia no es directamente modificable por los usuarios. Los botones son usados para invocar procesos [ver imagen 6.21].


Imagen 6.21: Referencia botón

- **Lista**: es una referencia utilizada para limitar los valores de una columna a unos valores predefinidos de una lista [ver imagen 6.22]. Todos los valores incluidos en una lista de validación están incluidos en el módulo, donde se definen las listas de validaciones. En Openbravo 2.50 no puedes ni añadir ni modificar las listas de validación en un módulo distinto al desarrollado.

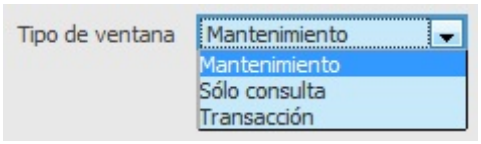



Imagen 6.22: Referencia lista

- **ID**: es una referencia especial utilizada para almacenar claves primarias de las tablas. No suelen ser visualizados en la Interfaz Gráfica de Usuario ya que no tiene mucho sentido para el usuario final ver el número identificativo.

Dependiendo del tipo de dato que vayamos a almacenar en las columnas de nuestro módulo utilizaremos un tipo de referencia u otro. Y en caso que ninguno de los existentes se acople a nuestras necesidades podremos crear una nueva referencia desde este apartado.

#### *6.1.5.3.- Ventanas*

Al entrar en la ventana de Referencia del Diccionario de Aplicaciones se muestra un listado con todas las referencias existentes. En él podremos buscar la que necesitemos.

Desde esta ventana es posible crear una nueva referencia para cuando necesitemos crear una que no existe [ver imagen 6.23]. Para ello será necesario indicar el nombre de la nueva referencia y el módulo donde se va a implementar. De cara al desarrollo, será necesario también indicar en qué clases Java se implementan: su código de lógica, su protocolo de actuación el en WAD y su visualización en la Interfaz Gráfica de Usuario. Estas clases sólo serán necesarias implementarlas para algunos tipos de referencia.

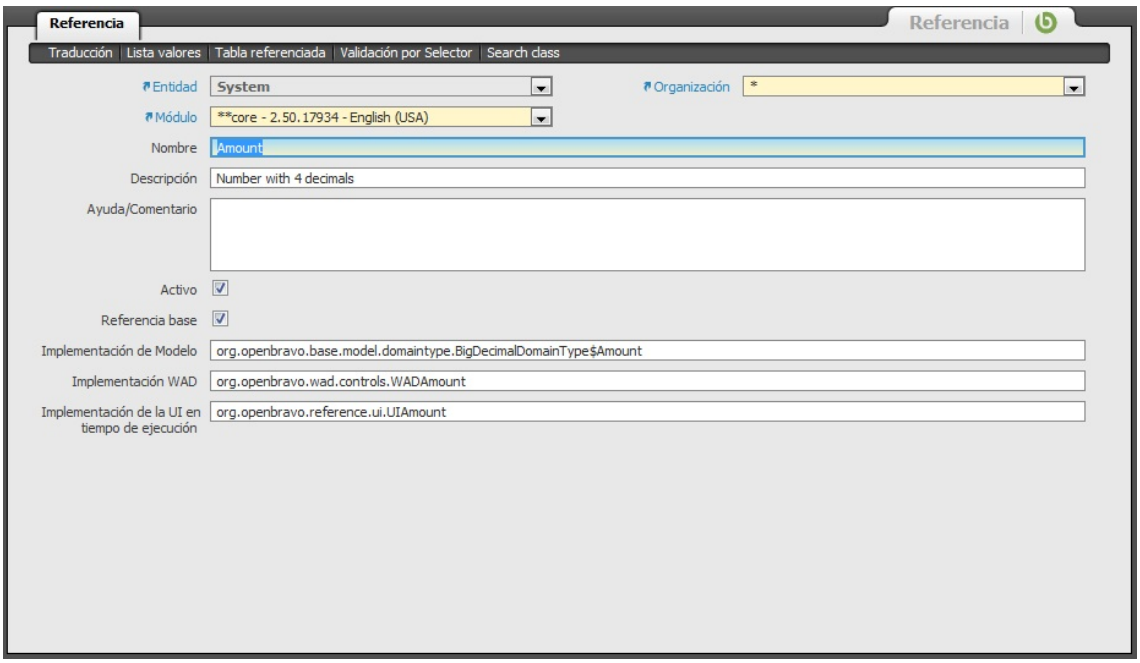



Imagen 6.23: Ventana de creación de una referencia

Una vez creada, podemos indicarle las traducciones necesarias desde la pestaña "Traducción".

En caso de que la referencia sea del tipo *Lista*, deberemos indicar todos los valores permitidos para este tipo en la pestaña "Lista valores" [ver imagen 6.24]. Cada valor debe de contener una referencia al módulo donde está desarrollado, un identificador (que será el valor utilizado a nivel de programación) y un nombre descriptivo.

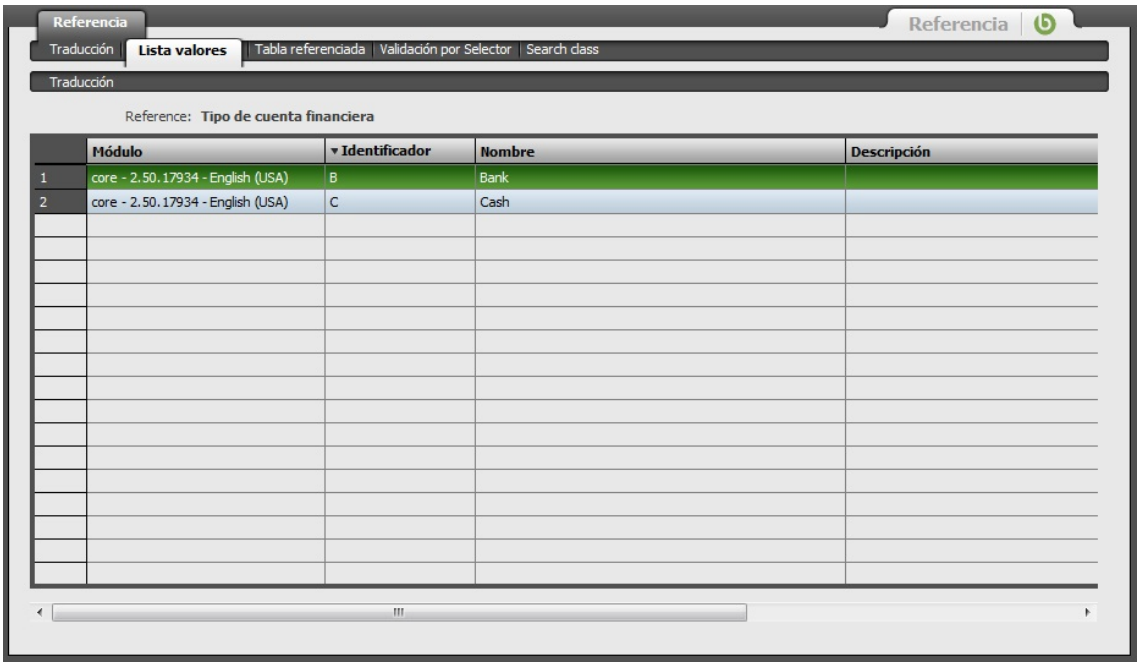



Imagen 6.24: Listado de valores de una Referencia

Cuando se esté creando una subreferencia será necesario indicar a qué tabla se está haciendo referencia. Esto se hace mediante la pestaña "Tabla referenciada" [ver imagen 6.25]. En ella se puede configurar la tabla del Diccionario de Aplicaciones, la columna a la

que se va hacer referencia y, en caso de necesitarlo, una consulta SQL para filtrar los resultados de la tabla.

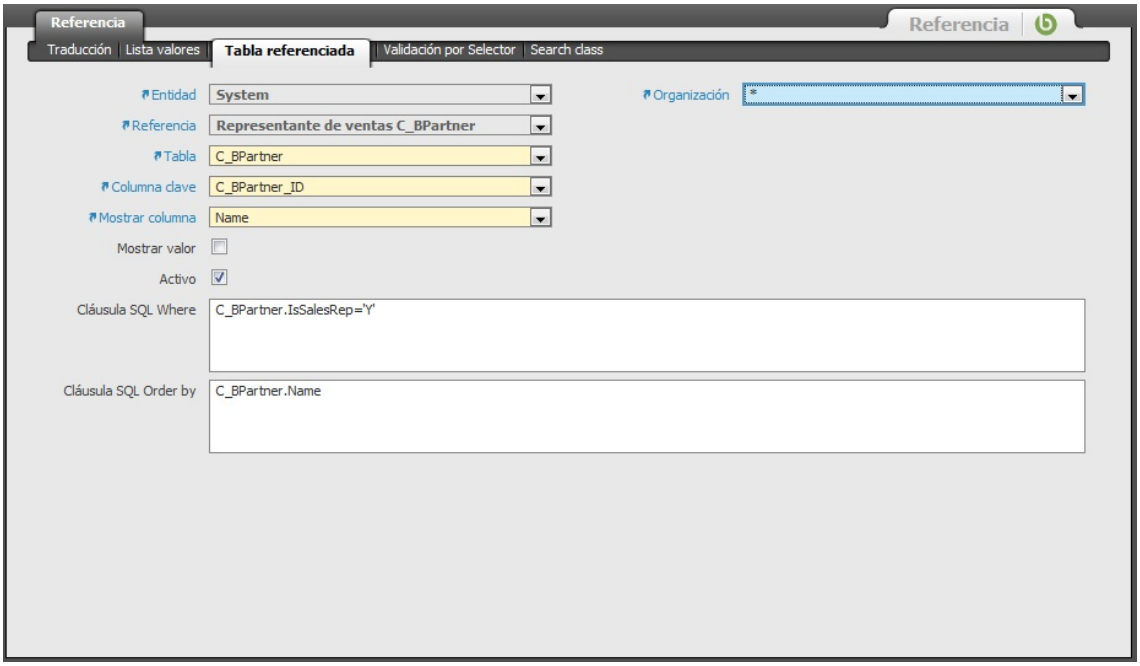



Imagen 6.25: Tabla referenciada de una Referencia

En algunos casos para rellenar un campo en vez de una lista se usa una ventana emergente en la que, con ayuda de los filtros, se selecciona el registro deseado. Estos campos se construyen con el tipo de referencia selector.

En la pestaña “Validación por selector” se define la tabla y columna referenciada y en la subpestaña “Columnas de validación por selector” otras columnas que el selector necesite. Las ventanas emergentes se construyen manualmente.

#### *6.1.5.4.- Validación*

Para validar algunos tipos de referencias como por ejemplo las listas, es posible crear una validación que actúe de filtro sobre los resultados.

Cuando una columna tiene una validación, el WAD genera un *callout*[7] que es invocado y que realiza el proceso de filtrado.

Para definir una validación hay que ir al Diccionario de aplicaciones, al apartado Setup. Más adelante, en el apartado 6.2.5 se verá como crear una validación.

### 6.1.6.- Informes y procesos

En este apartado vamos a introducir el concepto de los informes y los procesos y de cómo están gestionados en OpenbravoERP.

[7] Un *callout* es una porción de código que se ejecuta al realizar una acción

### *6.1.6.1.- Introducción*

En esta ventana se dan de alta todos los informes y procesos a los que se quiere acceder directamente desde la aplicación. Si un informe o proceso necesita parámetros de entrada estos se definen desde la misma ventana.

**Informes**

Un informe sirve para recuperar datos de la base de datos y visualizarlos de forma que el usuario pueda obtener información útil. En Openbravo existen un grupo de informes predefinidos para cada módulo. En ellos se intenta generalizar las necesidades, pero en la mayoría de las ocasiones, nos será necesario implementar un nuevo informe para obtener realmente la información que se necesita.

Un informe suele permitir la introducción de parámetros para filtrar la información a recuperar. Por ejemplo, en la mayoría de informes necesitaremos dos parámetros básicos, la fecha de inicio y la de fin. El resultado se puede obtener tanto en un fichero HTML como en PDF.

OpenbravoERP soporta dos tipos de informes:

- **JasperReports**: es una librería para la generación de informes implementada en Java. OpenbravoERP 2.50 permite la vinculación de informes con JasperReports 3.0.1. Gracias a ella la interfaz se genera automáticamente.
- **Servlet Reports**: básicamente es un *servlet* que se comunica con la base de datos y genera todos los contenidos de la página.

La herramienta iReport es un constructor / diseñador de informes visual y fácil de usar para JasperReports. Está escrito en Java. Esta aplicación permite que los usuarios corrijan visualmente informes completos con cartas, imágenes, subinformes, etc. Además, está integrado con JFreeChart, una de las bibliotecas gráficas OpenSource más difundida para Java.

**Procesos**

Un proceso es una serie de acciones sistemáticas con un fin. Típicamente un proceso recibe una serie de parámetros y utilizándolos se realiza un proceso para obtener un resultado. OpenbravoERP define dos tipos de procesos:

- **Procesos Java**: son funciones implementadas en Java.
- **Procesos PL/SQL**: son procesos implementados y almacenados en la base de datos.

Una vez que se ha definido un proceso este puede ser invocado directamente desde el menú o a través de un botón.

### *6.1.6.2.- Creación*

La estructura de un informe y de un proceso en OpenbravoERP es muy parecida. Ambos tienen partes en común y es por ello que en el Diccionario de Aplicaciones

se ha juntado su creación desde la misma ventana.

**Informes**

Cuando se crea un informe se debe de implementar primero de todo el cuerpo del informe. Dependiendo del tipo de informe que hagamos, JasperReports o Servlet Reports (también llamado “informe manual”), se utilizará unas herramientas u otras.

Para crear un informe JasperReports hará falta instalar la aplicación iReports y desde ahí, diseñar el informe y definir su consulta SQL.

Para crear un informe manual será necesario implementar un *Servlet* Java que recupere la información y la visualice. Para ello deberá de extender la clase Java *HttpBaseServlet*, que contiene los procesos requeridos para enlazar el informe a OpenbravoERP.

Un informe suele estar compuesto por una serie de ficheros con el mismo nombre pero con diferente extensión. En la siguiente lista se detalla cada uno de ellos:

- **Fichero controlador (Java)**: es el resultado de la compilación del modelo XSQL. Contiene todos los comandos necesarios en Java que administran el informe.
- **Fichero de visualización de plantilla (JRXML, HTML o FO)**: contienen el código necesario para representar el informe en pantalla. Dependiendo de donde se vaya a visualizar el informe será necesario un tipo de fichero u otro.
- **Fichero de configuración (XML)**: en él se detalla toda la información que configura el informe, como por ejemplo los parámetros.
- **Fichero de Modelo (XSQL)**: incluye todos los comandos SQL requeridos para recuperar/actualizar los datos.

Una vez creado el informe físicamente hay que registrarlo dentro del Diccionario de Aplicaciones para que OpenbravoERP sepa de su existencia y localización. Esto se hace mediante la ventana “Informes y Procesos”.

Por último, es necesario compilar el informe mediante el comando *ant compile.development –dTab=xxx*. Este proceso realiza las siguientes operaciones:

- Convierte el fichero modelo *.xsql en una clase Java que se utiliza para acceder a los datos.
- Compila todas las clases para el informe.
- Traduce el informe en los idiomas que están configurados en base de datos.
- Genera el mapeo para introducir las referencias a la clase en el fichero web.xml que es utilizado por Tomcat.
- Compila los recursos generados manualmente (no los automáticos: Diccionario de Aplicaciones).
- Copia los resultados compilados en el directorio /Openbravo dentro de Tomcat para que se pueda acceder desde la aplicación.

La visualización final de un informe se hace mediante el fichero *.jrxml, que se pueden utilizar mediante el programa iReport 2.0.4. Es posible también visualizar un formulario codificado en .html o .fo, pero su mantenimiento se hace más complicado.

**Procesos**

Un proceso es básicamente una porción de código que se va a ejecutar cuando el usuario lo indique.

Para crear un proceso se debe seguir los siguientes pasos:

1. Crear el nuevo proceso desde el Diccionario de Aplicaciones.
2. Definir los parámetros del procedimiento.
3. Crear el código que contiene la funcionalidad deseada. En caso de ser un proceso Java será necesario crear la clase Java que implemente la funcionalidad. Y en caso de que sea un proceso PL/SQL, será necesario crear el procedimiento almacenado en la base de datos.
4. Crear el acceso al procedimiento asignándole una nueva opción en el menú de la izquierda (Configuración General > Aplicación > Menú). También es posible crear un botón asociado al procedimiento.
5. Compilar.

#### *6.1.6.3.- Ventanas*

La primera ventana que se visualiza al entrar contiene un listado de todos los informes y los procesos registrados en el sistema.

A la hora de crear un nuevo informe o proceso será necesario especificar [ver imagen 6.26]:

- Identificador: en el caso de los informes el identificador se corresponde a su nombre. En el caso de los procedimientos este campo debe coincidir exactamente con el nombre de la función que implementa el proceso.
- Acceso datos: define quien va a tener acceso al informe o proceso.
- Patrón de la Interfaz de usuario: puede contener dos valores estándar o manual.
    - El patrón estándar sirve para que el propio OpenbravoERP (mediante su herramienta WAD) genere automáticamente un pop-up[8] para que el usuario introduzca los parámetros del informe o proceso.
    - En caso de ser manual, la interfaz es administrada por el desarrollador. Se deben implementa las clases Java necesarias ya que no se generan ventanas automáticas.
- Informe Jasper: este campo debe estar activo si se está creando un informe JasperReports.
- Servicio externo: este campo debe estar activo si se está creando un informe Servlet Reports.

[8] Ventana emergente

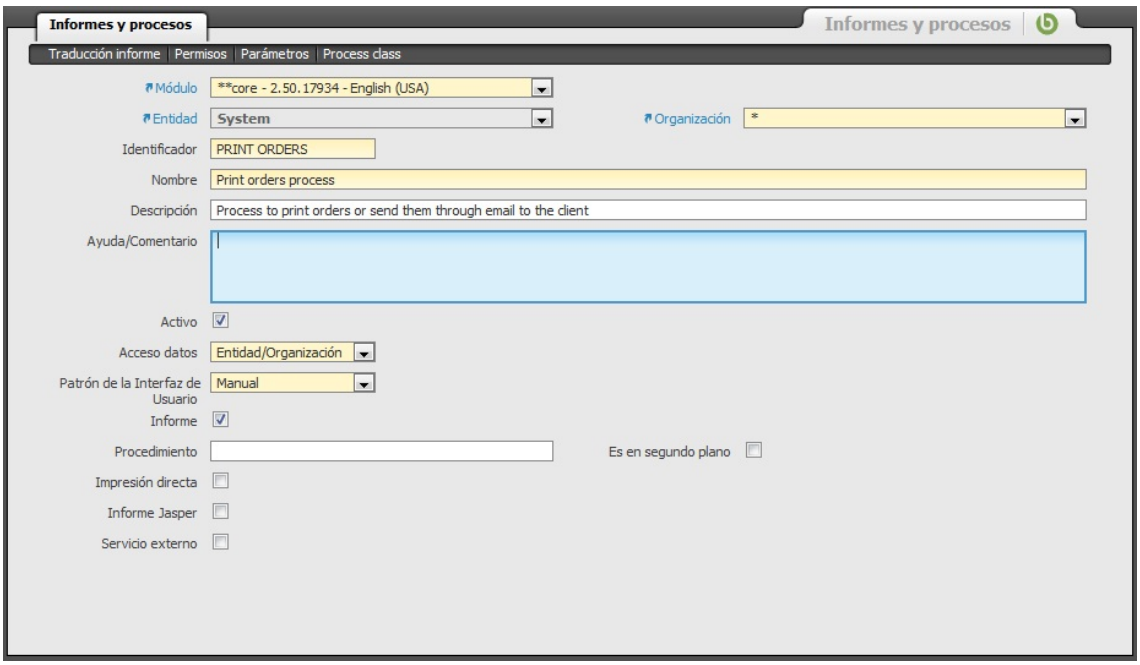



Imagen 6.26: Ventana de creación de un informe o proceso

Una vez creado el informe o proceso, es posible que este requiera una serie de parámetros para poder funcionar correctamente. Para definirlos, disponemos de la pestaña “Parámetros” [ver imagen 6.27]. En ella vamos a poder configurar un parámetro mediante los siguientes campos:

- Secuencia: define el orden de los parámetros.
- Nombre columna db: define el nombre del parámetro en la plantilla JRXML. El nombre debe coincidir exactamente con el nombre de la columna en base de datos.
- Elemento del sistema: define el Elemento usado a la hora de renderizar este parámetro.
- Referencia: define que tipo de Referencia se va a utilizar a la hora de visualizar el parámetro.
- Referencia clave: aquí se debe indicar la subreferencia en caso de que sea necesario.
- Obligatorio: define si su valor se puede dejar en blanco.

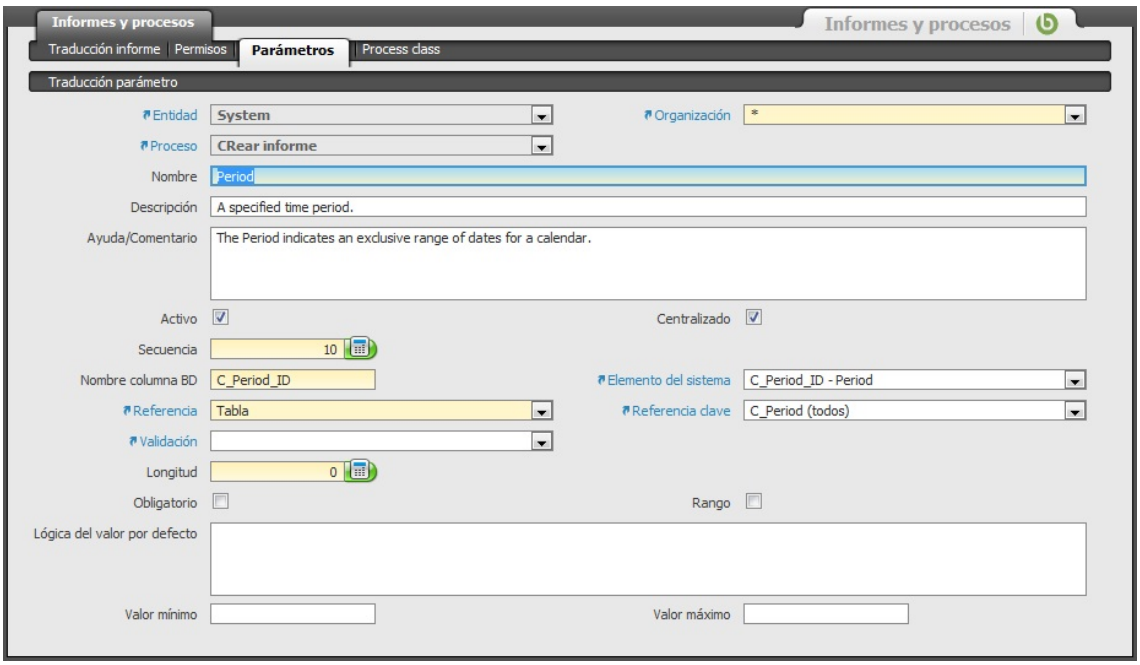



Imagen 6.27: Ventana de creación de los parámetros

En el caso de estar implementando un proceso Java o un informe es necesario indicar qué clase java lo va a implementar. Esto se hace mediante la pestaña “Process class” [ver imagen 6.28].

Esta pestaña permitirá configurar:

- Acción: especifica que tipo de elemento se está implementando: informe, formulario, proceso, ventana, etc.
- Nombre de la clase java: especifica el nombre del paquete Java. En él va implícito la localización de la clase Java dentro del proyecto Openbravo.

Cuando se está creando un informe, además de registrar la clase Java en el Diccionario de Aplicaciones, es necesario mapear la clase para que el proceso sea accesible desde el fichero web.xml. Esto lo hará visible para Tomcat y podremos invocar el informe desde una URL.

La subpestaña “Process mapping” [ver imagen 6.29] podemos realizar el mapeo del proceso. En la ventana indicaremos:

- Nombre del mapeo: especifica la URL relativa a partir de la carpeta */erpCommon*. La URL debe terminar en .pdf o .html.
- Valor por defecto: si está activo indica que se aplicarán los parámetros que estén por defecto.

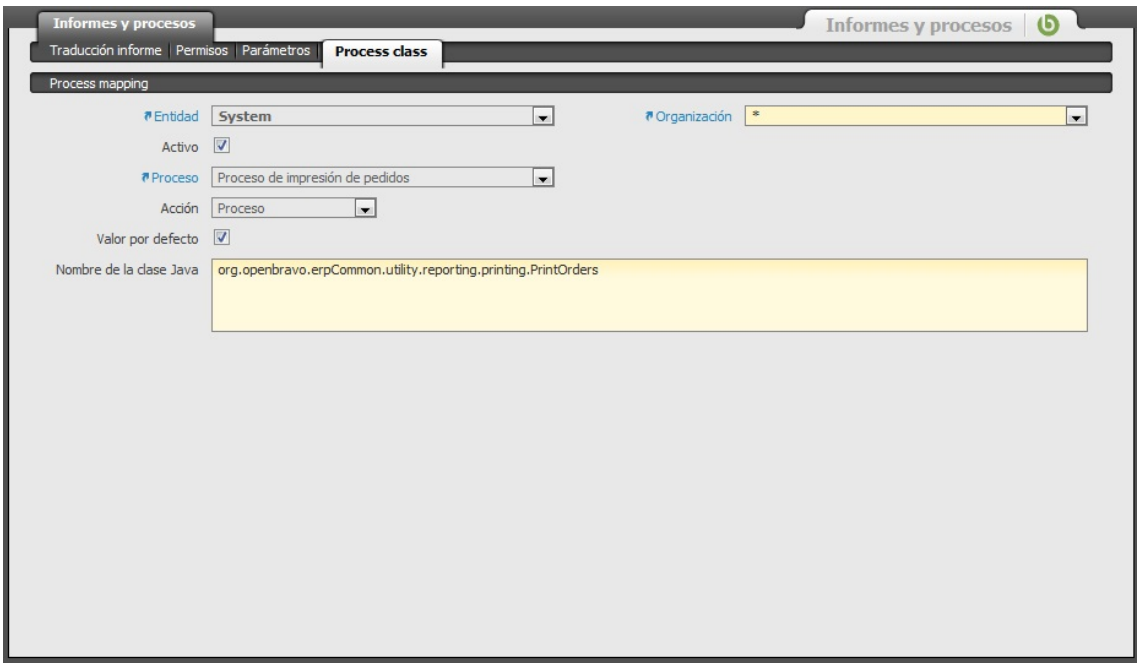



Imagen 6.28: Ventana de asociación de una clase Java

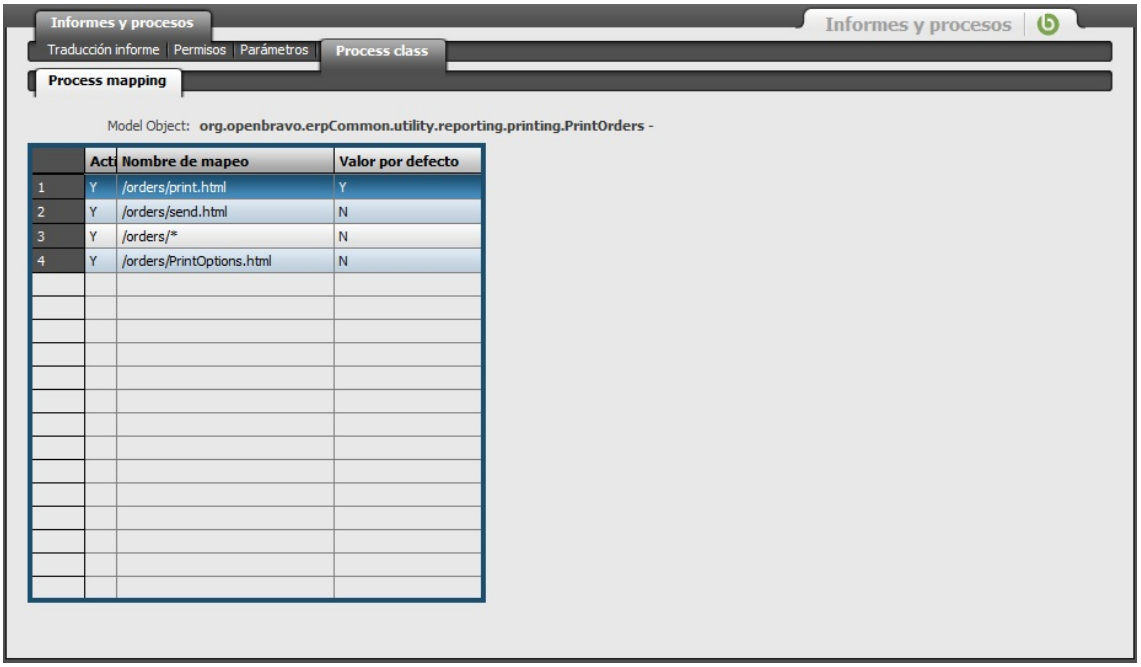



Imagen 6.29: Mapeo de una clase Java

### 6.1.7.- Formulario

En este apartado vamos a ver la función de los formularios y su configuración en OpenbravoERP.

#### *6.1.7.1.- Introducción*

Un formulario es una ventana generada manualmente que sirve para introducir, modificar o borrar datos. A diferencia de las ventanas estándar, los formularios permiten la entrada de

datos más complejos y permite al usuario introducir datos que vayan a ser utilizados en más de un lugar.

Los formularios le dan al sistema ERP mucha flexibilidad. Con ellos es posible muchos tipos de herramientas como un formulario de contacto o un formulario para exportar datos.

### *6.1.7.2.- Creación*

El proceso de creación de un formulario es muy similar al de un informe o proceso.

Un formulario suele estar compuesto por una serie de ficheros cada uno con su funcionalidad. Estos ficheros se corresponden con los mencionados en la creación de informes [ver apartado 6.2.6]. Todos ellos deben estar localizados en el directorio */src/org/openbravo/erpCommon/ad_forms/*.

Para crear un formulario se debe empezar implementando su código fuente (tanto la visualización en HTML como su funcionalidad en Java y SQL). El código Java debe construir un *Servlet* que extienda el ya existente en OpenbravoERP *HttpBaseServlet*. para que el formulario funcione.

Una vez creado el formulario físicamente hay que registrarlo dentro del Diccionario de Aplicaciones para que OpenbravoERP sepa de su existencia y localización.

Por último, es necesario compilar el formulario mediante el comando *ant compile.development –dTab=xxx*.

### *6.1.7.3.- Ventanas*

Los formularios son un artefacto más del Diccionario de Aplicaciones. A la hora de crear uno nuevo deberemos asignarle un módulo para que pueda ser empaquetado. En la ventana de creación de formularios [ver imagen 6.30] se tiene que configurar el nombre del formulario, quien va a tener los permisos para acceder y el nombre de la clase Java asociada.

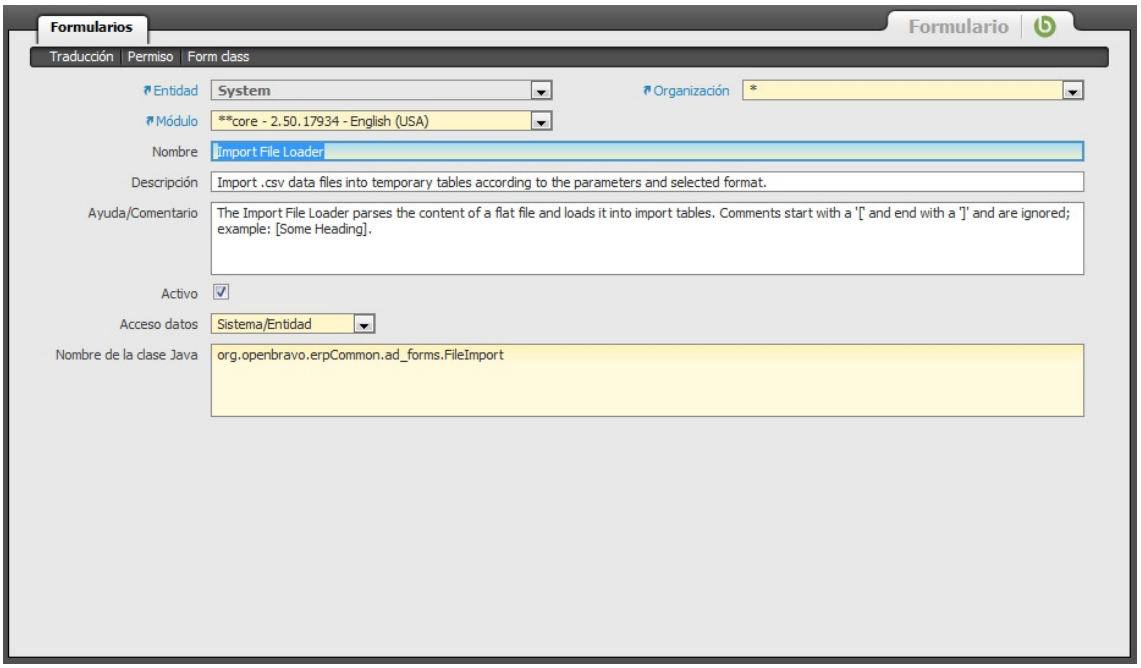


Imagen 6.30: Ventana de creación de un formulario

En la pestaña “Form Class” se define la clase Java que va a implementar el formulario [ver imagen 6.31]. Ésta es un puente para poder mapear el formulario. Los campos “Formulario especial” y “Nombre de la clase Java” se corresponden con los campos “Nombre” y “Nombre de la clase Java” de la ventana anterior.

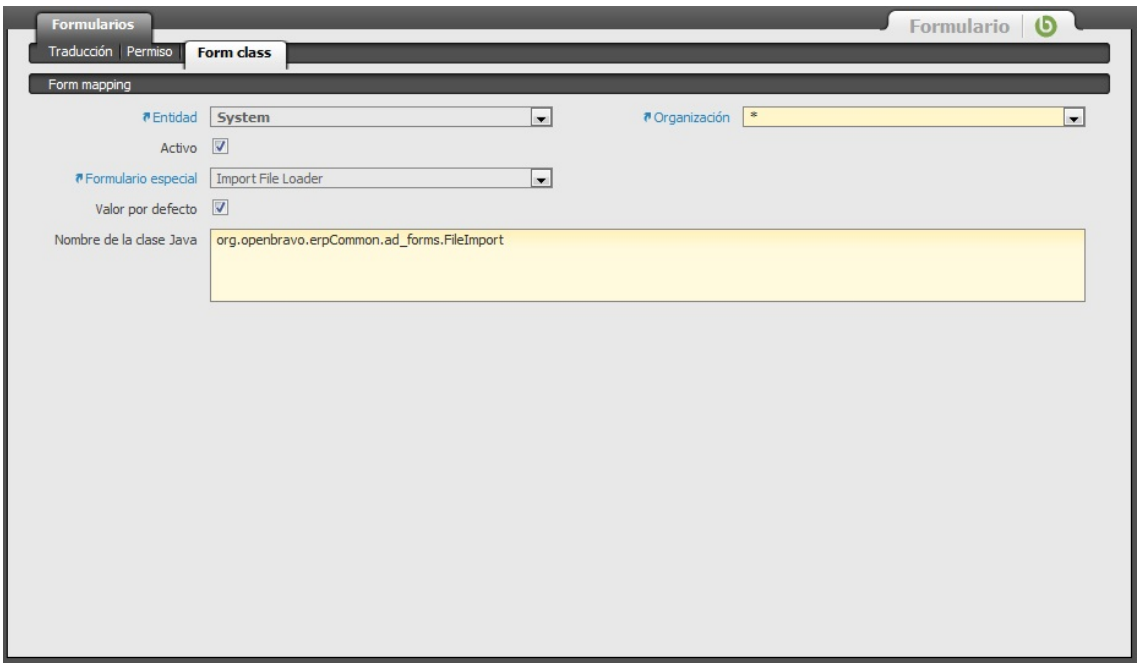



Imagen 6.31: Clase Java asociada al formulario

Por último, es necesario definir el mapeo del formulario mediante la pestaña “Form mapping”. El mapeo, como ya hemos comentado anteriormente, sirve para ejecutar el formulario en el navegador Web a través de una URL. Mediante esta ventana [ver imagen 6.32] es posible mapear cualquier fichero dentro del paquete del módulo. Por ejemplo:

*/org.company.mymodule.form/MyForm.html*.

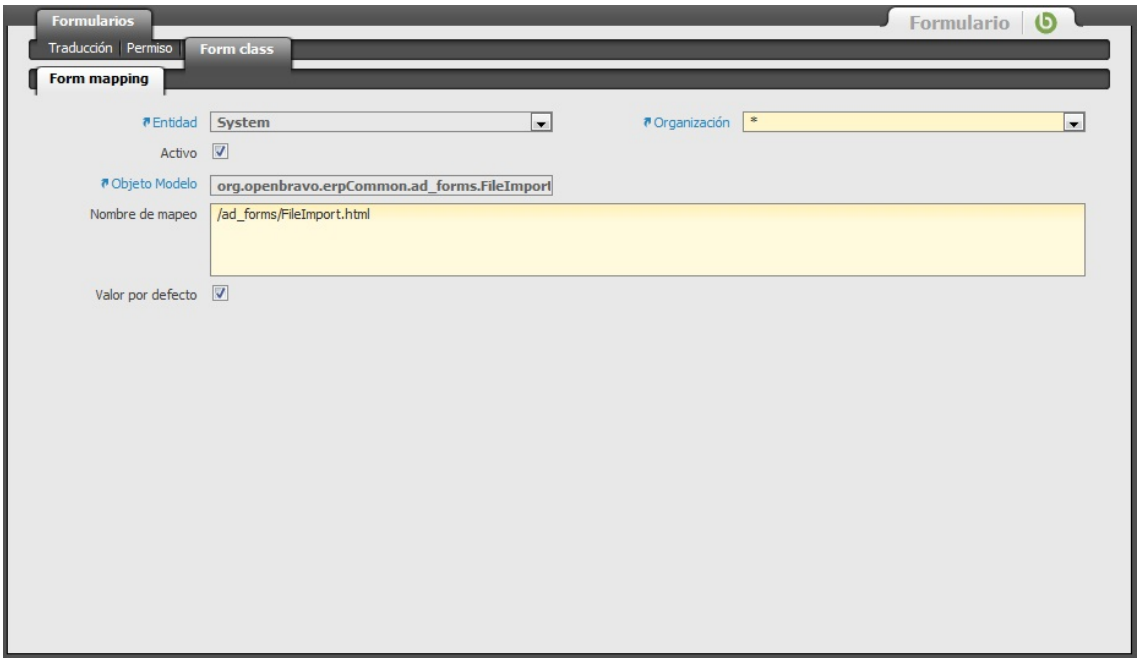


Imagen 6.32: Mapeo de la clase Java

### 6.1.8.- Mensaje

En este apartado vamos a ver la funcionalidad de los mensajes en OpenbravoERP.

#### *6.1.8.1.- Introducción*

Los mensajes son utilizados para hacer que el usuario visualice cierta información, por norma general se visualizan cuando un proceso es completado o cuando ocurre un evento (por ejemplo un error).

Existen distintos tipos de mensaje en OpenbravoERP:

- **Mensajes de éxito**: son visualizados tras la ejecución de un proceso que ha acabado en éxito. En este caso se visualiza un texto con el fondo de color verde [ver imagen 6.33].

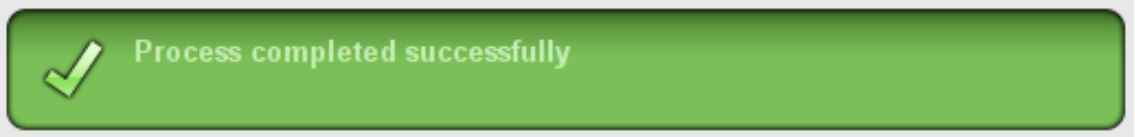



Imagen 6.33: Mensaje de éxito

- **Mensajes de error**: son visualizados tras la ejecución de un proceso que ha acabado en error. En este caso se visualiza un texto descriptivo del error para que el usuario pueda intentar corregirlo y con un fondo de color rojo [ver imagen 6.34].

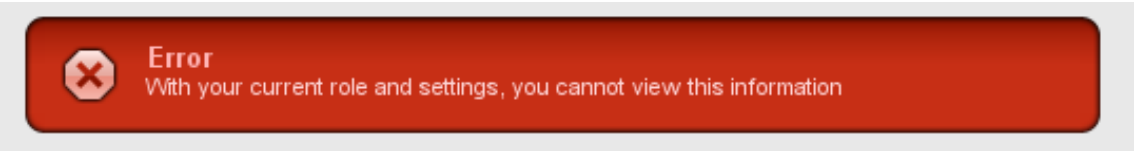



Imagen 6.34: Mensaje de error

- **Mensajes de alerta**: son visualizados en cuando el usuario tiene que ser alertado de ciertas condiciones, como por ejemplo cuando el cliente elegido ha sobrepasado el límite establecido, etc. En este caso se visualiza un texto descriptivo de la alerta sobre un fondo de color amarillo[ver imagen 6.35].

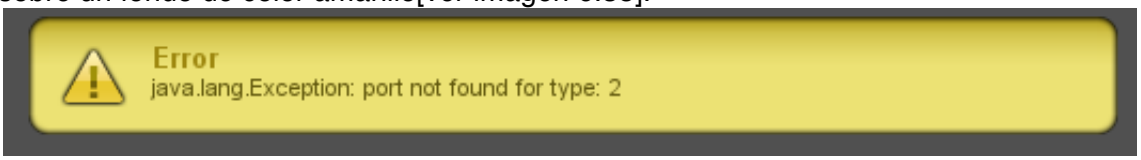



Imagen 6.35: Mensaje de alerta

- **Mensajes de información**: son visualizados cuando se quiere comunicar algo al usuario, como por ejemplo la funcionalidad de una ventana. En este caso se visualiza el texto elegido sobre un fondo de color azul [ver imagen 6.36].

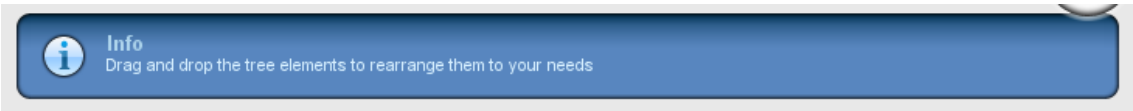



Imagen 6.36: Mensaje de información

### *6.1.8.2.- Creación*

Todos los mensajes se administran desde el Diccionario de Aplicaciones. Para poder utilizarlos primero de todo hay que crearlos desde la opción del menú Mensajes.

Una vez registrado el mensaje con su texto, tipo e identificador es posible invocarlo desde el código fuente. Puesto que es posible interactuar con OpenbravoERP desde distintos lenguajes de programación (Java, XML y SQL) cada uno invoca el mensajes mediante una función concreta de la API.

### *6.1.8.3.- Ventanas*

Cuando entramos en la ventana de Formulario nos encontramos con un listado de todos los mensajes registrados en OpenbravoERP. Cuando editamos o añadimos un nuevo mensaje nos aparecerá una ventana en la que tendremos que configurar [ver imagen 6.37]:

- Identificador del mensaje: nos servirá para luego invocar el mensaje desde el código fuente.
- Modulo: módulo al que pertenece.
- Tipo de mensaje: aquí se especifica la función del mensaje: error, informativo, éxito, etc.
- Mensaje de texto: descripción del texto que se visualizará en la ventana emergente.
- Consejo: texto auxiliar.

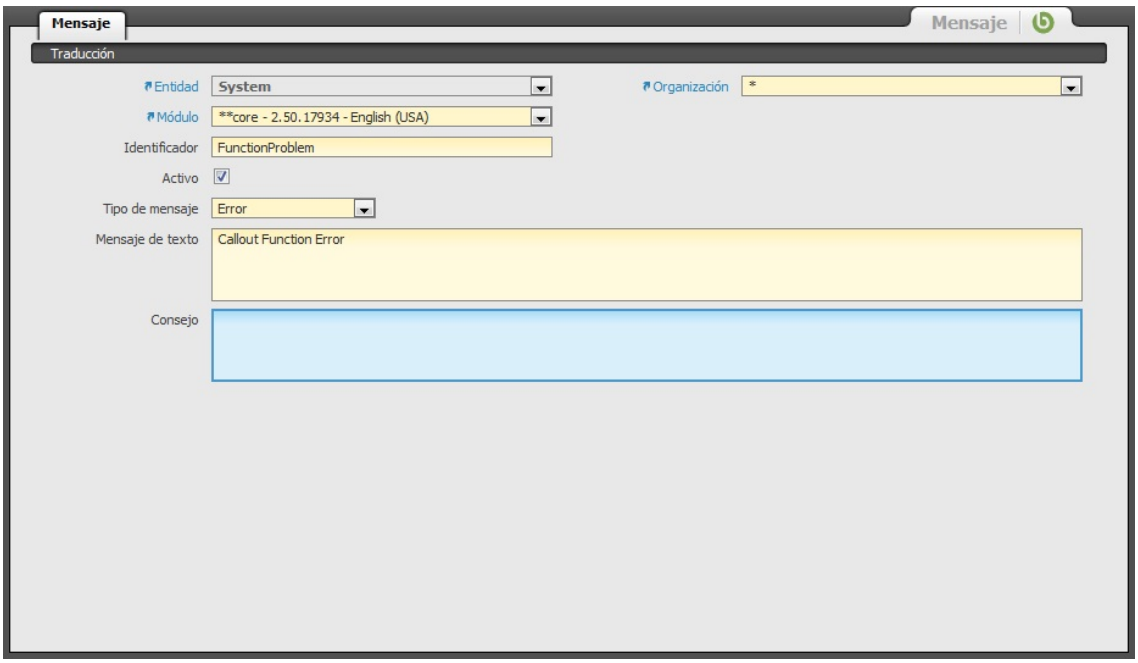



Imagen 6.37: Ventana de creación de un mensaje

En la pestaña “Traducción” es posible indicar las traducciones del mensaje.

### 6.1.9.- Texto interfaces

Este apartado es un complemento para realizar las traducciones a otros idiomas.

#### *6.1.9.1.- Introducción*

El objetivo de la ventana de Texto interfaces es gestionar las traducciones de los textos localizados en las ventanas manuales y de los documentos generados. En muchos de los apartados del Diccionario de Aplicaciones existe la pestaña para traducir el artefacto que se está editando: Ventana, solapas y campos, Informe y procesos, formulario, etc. Pero existe una serie de artefactos como por ejemplo las ventanas manuales que la mejor forma de traducirla en OpenbravoERP es mediante esta opción.

#### *6.1.9.2.- Creación y Ventanas*

Para crear la traducción de un documento basta con crear un registro nuevo e indicar en el formulario [ver imagen 6.38]:

- Módulo al que pertenece el documento.
- Texto a encontrar en el documento. Es el segmento de texto que se quiere traducir. Lo más probable es que en el documento existan varios que sea necesario traducir. En este caso crearemos un registro nuevo por cada segmento de texto.
- Nombre del archivo que contiene el texto. Si este campo se deja vacío la traducción se aplica a todos los archivos donde aparezca el texto y los que no tengan traducción específica.

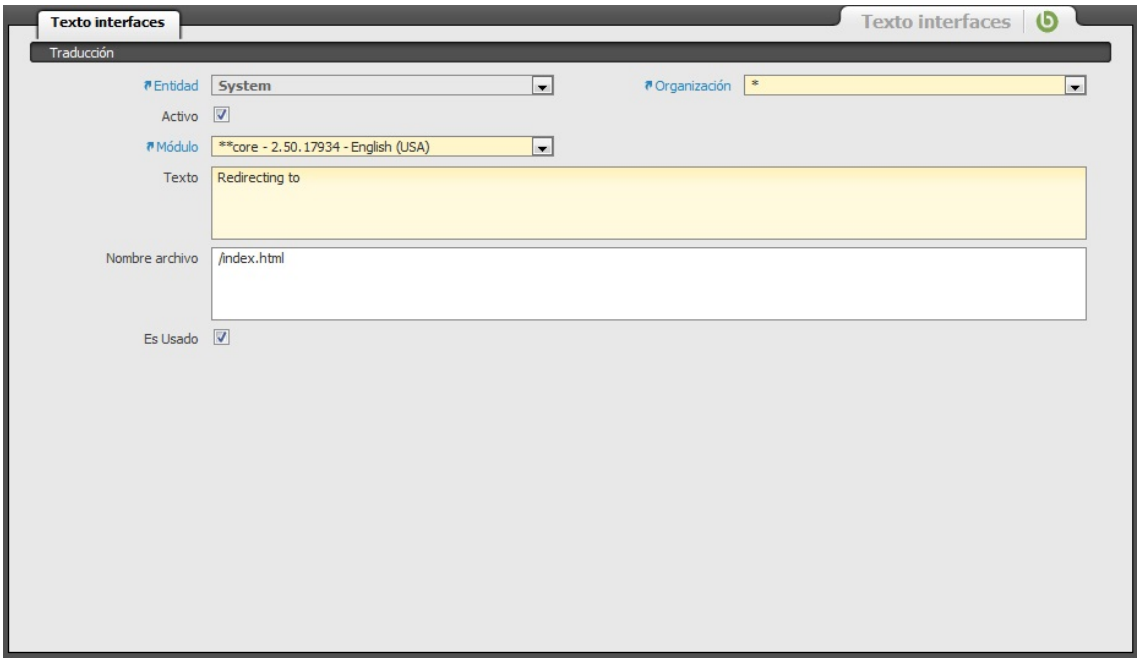



Imagen 6.38: Ventana de creación de un texto de interfaz

Una vez registrado el texto, en la pestaña “Traducción” configuraremos la traducción del mismo a los idiomas deseados [ver imagen 6.39].

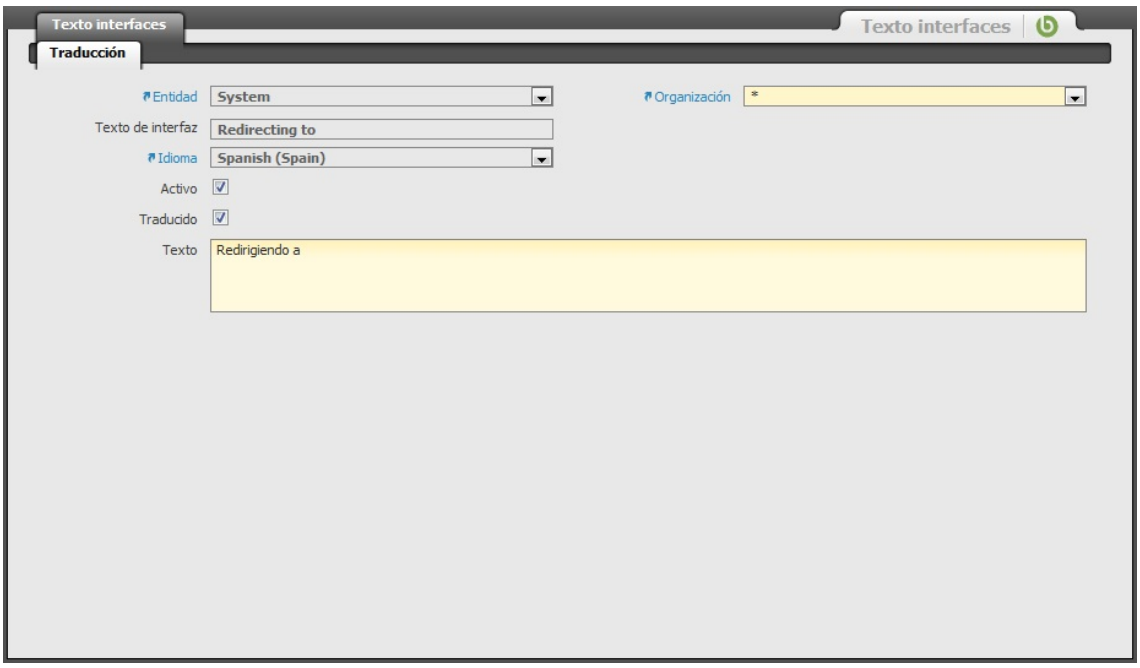



Imagen 6.39: Traducción de un texto

### 6.1.10.- Mapeo de Implementación del Diccionario

En este apartado vamos a ver la funcionalidad del Modelo-Implementación y cómo configurarlo en el Diccionario de Aplicaciones.

#### *6.1.10.1.- Introducción*

Las aplicaciones Web como OpenbravoERP tienen un *Descriptor de Despliegue*. Esto es un fichero XML (*web.xml*) que define todo lo que necesita la aplicación y que el servidor (Tomcat) necesita saber.

OpenbravoERP genera el fichero web.xml al compilar la aplicación a partir de la información almacenada en las tablas *AD_Model_Object* y *AD_Model_Object_Mapping*.

Algunos de los objetos que se pueden configurar en el ERP, como por ejemplo los *Servlets*, necesitan este tratamiento especial para que puedan funcionar en el sistema. Estos objetos siguen lo que se denomina el concepto de Modelo-Implementación.

**Modelo – Implementación**

El concepto de Modelo – Implementación separa la parte lógica de un desarrollo (componentes del Diccionario de Aplicaciones) de la parte física (clases Java). La tabla *AD_Model_Object* de la base de datos de OpenbravoERP hace referencia a los componentes del Diccionario de Aplicaciones y las clases (*Servlet*) que los implementan. Este modelo tiene las siguientes ventajas:

1. Permite implementar de forma genérica reglas que se van a aplicar a todos los componentes del Diccionario de Aplicaciones como: seguridad, navegación, filtros, etc.
2. Este es el mecanismo para configurar automáticamente el fichero web.xml donde se declaran las clases (*Servlets*) del sistema.

Existe un pequeño número de excepciones a esta descripción. Son algunos *Servlets* que están instalados en el contexto de Openbravo pero que no están enlazados a ningún componente del DA. Por ejemplo ejemplos los *Servlets* de *DataGrid* o *AuditInfo*.

#### *6.1.10.2.- Creación*

Existen distintos tipos de objeto que se pueden mapear:

- **Servlets**: El uso más común del mapeo de implementación es para definir los *Servlets* asociados a los objetos del Diccionario de Aplicaciones. Cuando una pestaña, un formulario, un informe o un proceso es definido en el Diccionario de Aplicaciones, automáticamente se crea una nueva entrada en *AD_Model_Object* y *AD_Model_Object_Mapping* que apunta a la clase Java que implementa ese objeto y el mapeo del *Servlet*.
- **Filtros**: Filtran cierta información del objeto el objeto Java seleccionado. Se suelen utilizar para realizar filtrados en el mapeo. Se aplican a un conjunto de ficheros mediante la directiva "*", como por ejemplo: /Web/*. Se debe especificar la secuencia para saber que filtro de los instalados se ejecuta antes.
- **Oyentes**: Los oyentes son clases Java que se ejecutan cuando ocurre un tipo de evento concreto.

Existen ocasiones en las que es necesario definir un Servlet que no implemente un artefacto del Diccionario de Aplicaciones. Para estos casos se configura el mapeo desde la ventana de "Mapeo de Implementación del Diccionario".

#### *6.1.10.3.- Creación y Ventanas*

La primera ventana que se visualiza al entrar en el Mapeo permite el registro de un objeto [ver imagen 6.40]. Aquí se deben de configurar los siguientes parámetros:

- Tipo de objeto: permite seleccionar el tipo de objeto que se va a mapear en el fichero web.xml. Los valores más comunes son: Servlet, Filtro y Oyente.
- Nombre de la clase Java: indica el paquete y nombre de la clase Java que implementa el Objeto.
- Cargar al iniciar: es un valor entero opcional. Si el valor es 0 o mayor indica el orden de ejecución de los *Servlets*. Los *Servlets* con mayor número son invocados después de los de menor número.
- Secuencia: es un valor entero opcional. Se utiliza únicamente para los objetos de tipo Oyente. Indica el orden de ejecución de los Oyentes empezando por el que tenga menor secuencia.

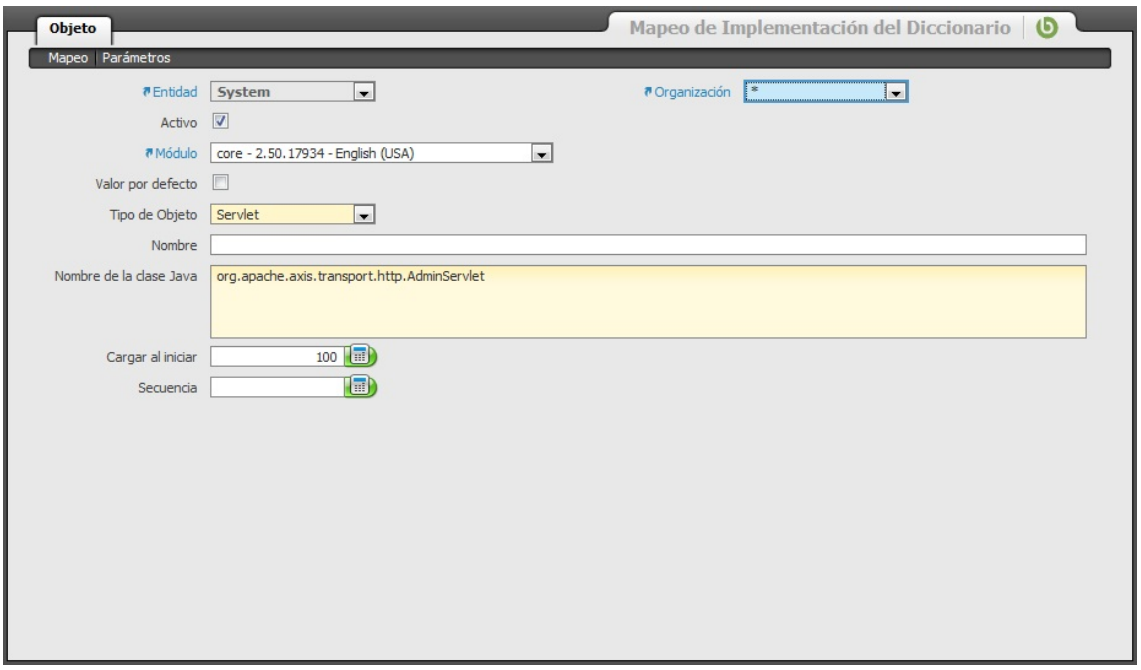



Imagen 6.40: Ventana de registro de un Objeto

Una vez creado el Objeto podemos mapearlo en el fichero web.xml mediante la pestaña “Mapeo”. Aquí debemos indicar el nombre del mapeo, que se corresponde con la URL del fichero deseado [ver imagen 6.41].

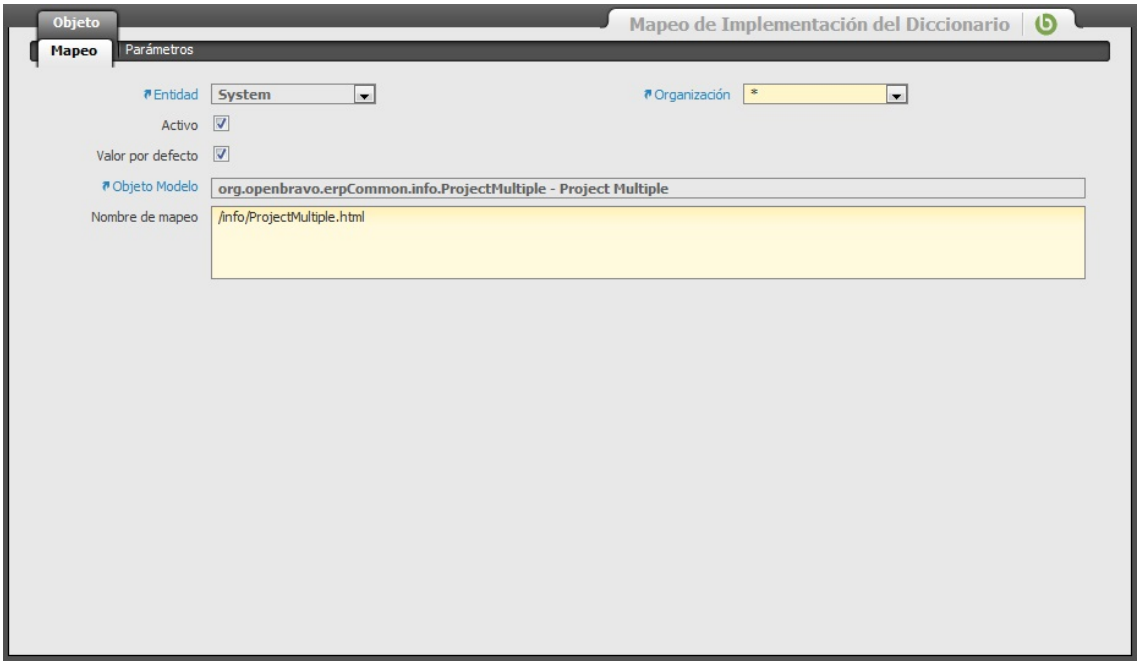



Imagen 6.41: Mapeo de un Objeto

Por último, en la pestaña “Parámetros” podemos configurar todos los parámetros que le vayamos a pasar al Objeto [ver imagen 6.42].

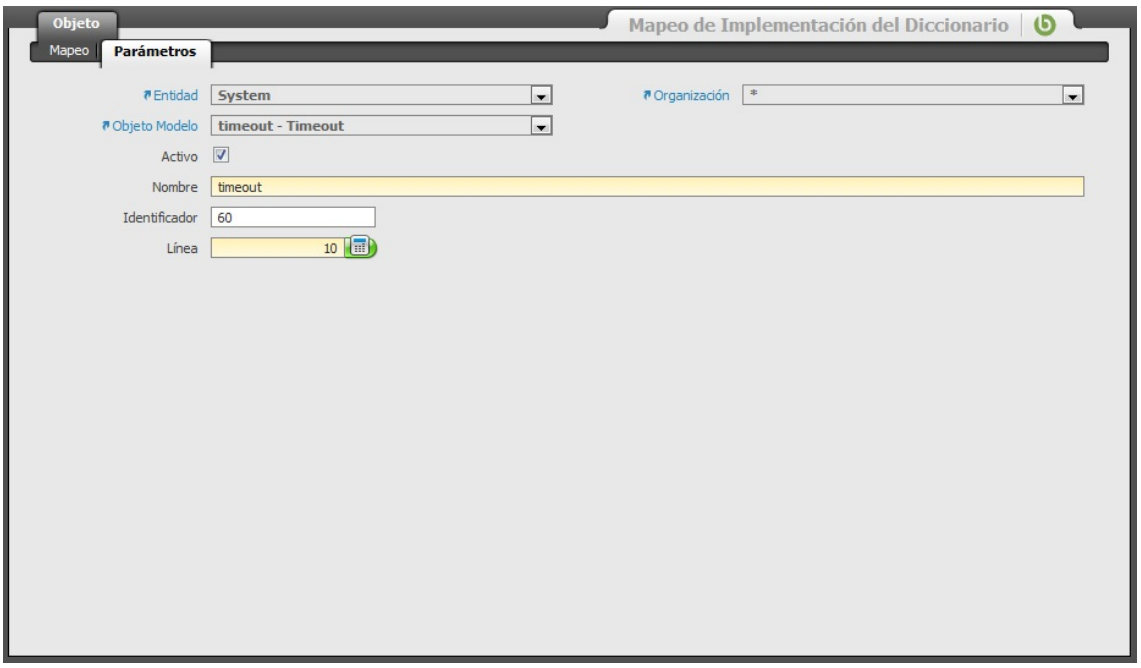



Imagen 6.42: Registro de los parámetros de un Objeto

### 6.1.11.- Actualizar infraestructura de histórico de auditoría

Este es un proceso que facilita la tarea de auditoría en el ERP.

#### *6.1.11.1.- Introducción*

El histórico de auditoría permite completar el seguimiento de todos los cambios realizados en cualquier tabla/entidad de OpenbravoERP. Esta característica puede ser activada por tablas y entonces cualquier registro insertado, actualizado o borrado permite su visualización de la información de auditoría a través de la aplicación de OpenbravoERP.

Esta capacidad está incluida en la aplicación nativa desde la versión 2.50MP12. Para las versiones anteriores es necesario instalar un módulo de auditoría. Sin embargo, este proceso está disponible solo para los clientes de OpenbravoERP *Professional Edition*.

#### *6.1.11.2.- Proceso*

Para activar la funcionalidad de histórico de auditoría es necesario que el administrador realice estas dos tareas:

- Activar el histórico de auditoría para una o más tablas del sistema.
- Ejecutar el proceso *Actualizar infraestructura de histórico de auditoría*.

**Activar el histórico de auditoría para una tabla**

Para activar o desactivar la capacidad de histórico de auditoría para una tabla seguir los siguientes pasos:

1. Registrarse en el sistema como Administrador del Sistema.
2. Ir a la opción de Diccionario de la Aplicación -> Tablas y Columnas.
3. Seleccionar la tabla a editar.

4. Marcar/Desmarcar la casilla “Completamente auditado”.

**Ejecutar el proceso de auditoría**

El sistema de auditoría utiliza una serie de *triggers* (lanzadores de base de datos) para recoger toda la información de auditoría. Estos *triggers* necesitan ser regenerados después de que alguna de estas acciones se realice:

- Se ha activado o desactivado la opción de auditoría en alguna tabla.
- Ha habido algún cambio estructural (nuevas columnas, cambio de la referencia de alguna columna) en la tabla auditada.

Para regenerar los *triggers* basta con ejecutar el proceso *Actualizar infraestructura de histórico de auditoría* [ver imagen 6.43].


Imagen 6.43: Ventana de inicio del proceso de auditoría

### 6.1.12.- Sincronizar términos

Este es un proceso encargado del mantenimiento de los Elementos del sistema.

#### *6.1.12.1.- Introducción*

Uno de los artefactos básicos del Diccionario de Aplicaciones es el Elemento. Un Elemento define el texto asociado que se visualizará junto a un campo en una ventana. Esto se hace así para poder administrar de una forma más ordenada todos los textos del sistema y sus traducciones.

El proceso *Sincronizar términos* tiene como función crear automáticamente los Elementos para las columnas que no existen y que están asociados a algún campo. Copia la información de los elementos de las columnas y campos a los que están enlazados.

#### *6.1.12.2.- Proceso*

Este proceso [ver imagen 6.44] se ejecuta como Administrador del Sistema. Básicamente este proceso busca todas las columnas de base de datos que no tengan enlazado un Elemento y lo crean automáticamente a partir de la información de la columna. Todo esto se hace teniendo en cuenta el módulo al que pertenece la columna. Si esta hace referencia a otro módulo cuyo Elemento ya está registrado, se reutilizarán los datos del Elemento para enlazar a la nueva columna.

Cuando se crea o edita un campo carece de sentido definirle un texto o descripción puesto que estos valores serán rellenados automáticamente por el proceso *Sincronizar Términos* a partir del Elemento que tendrá enlazado.

En el caso de no querer que este proceso se realice con algún campo basta con ir a la ventana *Diccionario de la Aplicación > Ventanas, solapas y campos > Solapa > Campo* y desactivar la opción “Centralizado”.

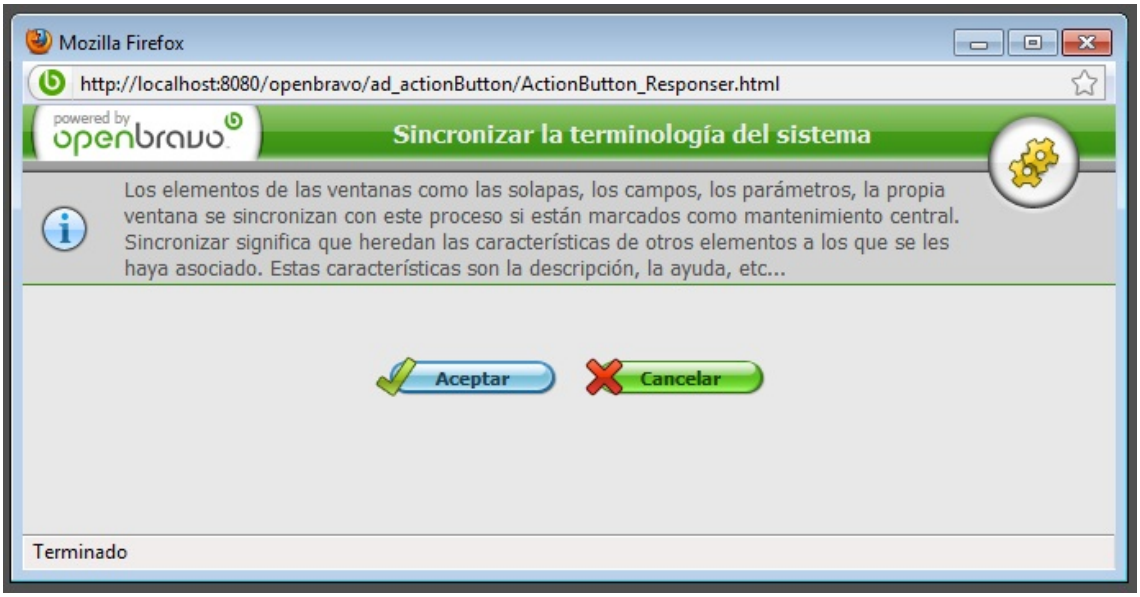



Imagen 6.44: Ventana de inicio del proceso de sincronización

## 6.2.- Configuración

En este apartado vamos a ver las opciones agrupadas en el submenú *Configuración*. Estas opciones van enfocadas al mantenimiento de elementos sencillos del Diccionario de Aplicaciones.

### 6.2.1.- Elemento

En esta opción estudiaremos la funcionalidad de los Elementos como artefactos básicos del Diccionario de Aplicaciones.

#### *6.2.1.1.- Introducción*

Un Elemento define el texto asociado que se visualizará junto a un campo en una ventana. Cada columna en el Diccionario de Aplicaciones está enlazada a un elemento. Gracias a

que un Elemento puede estar enlazado a más de una columna, un mismo texto puede ser reutilizado en distintas ventanas de la aplicación.

Además, los Elementos son una forma eficaz de gestionar las traducciones ya que centralizan todos los textos que se visualizan en la Interfaz Gráfica.

Los elementos se pueden editar desde la ventana **Configuración -> Elemento**. Pero por norma general los elementos no se editan directamente ya que son administrados automáticamente por el proceso *Sincronizar términos*.

### *6.2.1.2.- Creación*

Los Elementos deben crearse tras haber registrado las columnas de base de datos en el Diccionario de Aplicaciones. No es lo más común crear los Elementos manualmente ya que de esto se encarga de forma automática el proceso *Sincronizar términos*.

### *6.2.1.3.- Ventanas*

La primera ventana que aparece al entrar en esta opción permite la edición y creación de los Elementos [ver imagen 6.45]. Los campos más importantes son:

- Nombre Columna DB: se corresponde con el nombre de la columna de la base de datos a la que va a enlazar.
- Etiqueta de impresión: texto que se visualizará junto al campo que enlace este Elemento.
- Pedidos de Compra: los cuatro últimos campos corresponden a los textos asociados en el caso de que se esté visualizando el Elemento en un pedido de compra.

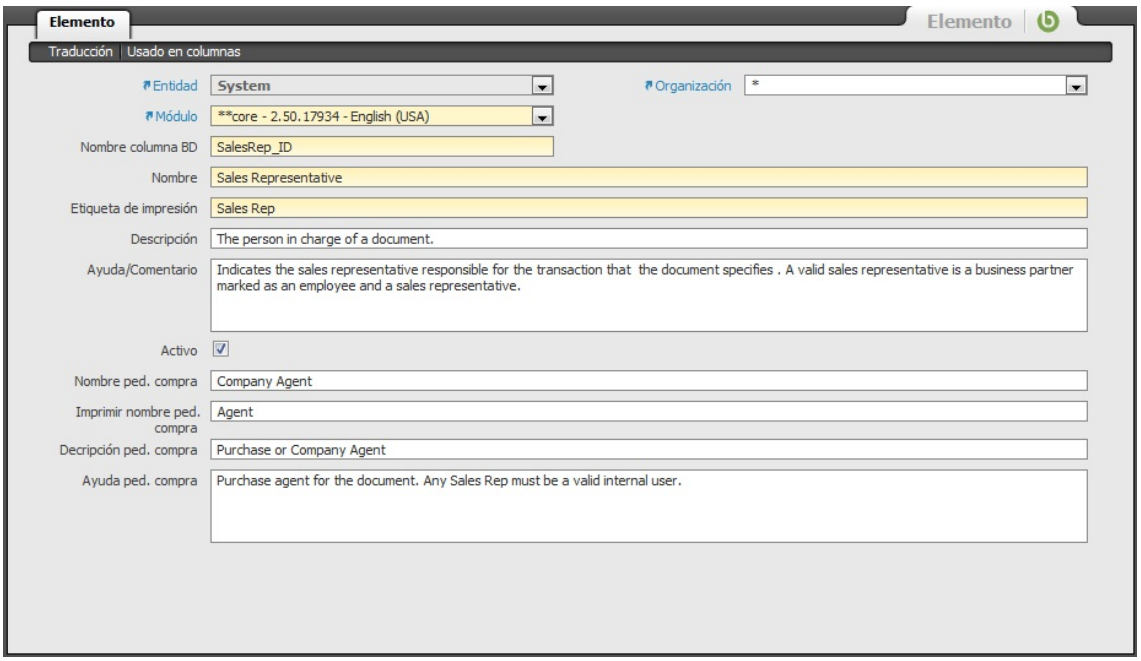



Imagen 6.45: Ventana de creación de un Elemento

En la pestaña “Traducción” se pueden crear las traducciones de los idiomas deseados para el Elemento.

Por último, en la pestaña “Usado en columnas”, se visualiza el listado de columnas y la tabla de base de datos que están enlazadas al Elemento [ver imagen 6.46]. Desde aquí se pueden crear nuevos enlaces, pero como hemos comentado antes, esto lo realiza automáticamente el proceso de *Sincronizar términos*.

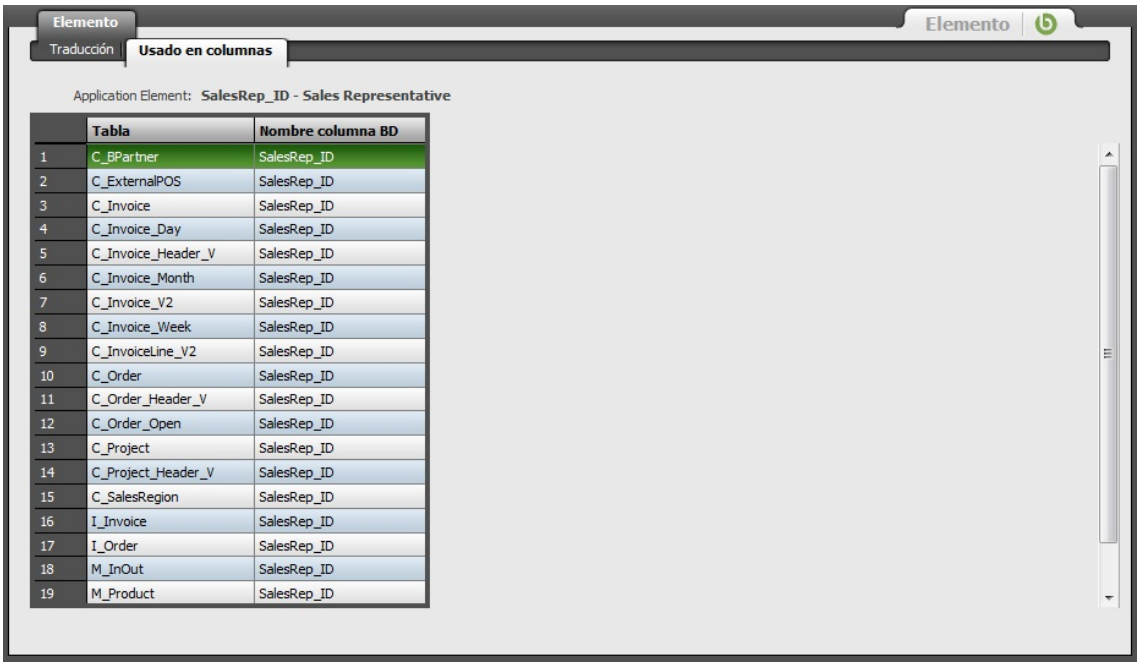



Imagen 6.46: Listado de columnas enlazadas al Elemento

### 6.2.2.- Tipo agrupación campos

En este apartado vamos a ver cómo gestionar los grupos de campos desde el Diccionario de Aplicaciones.

#### *6.2.2.1.- Introducción*

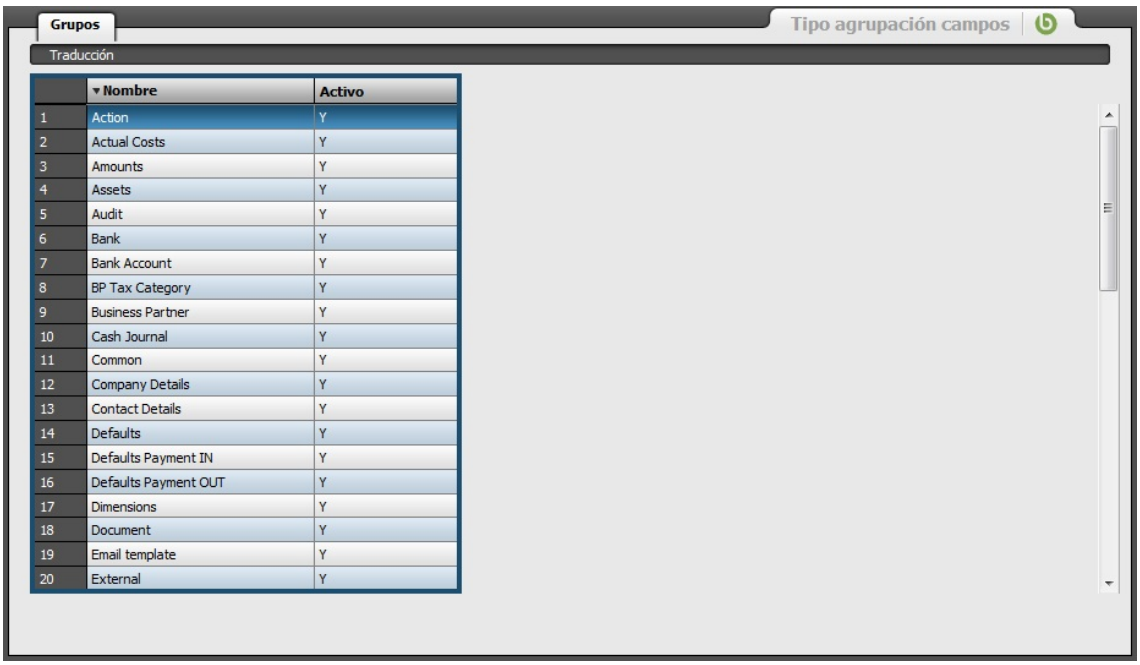
Las agrupaciones son conjuntos de campos que abarcan la misma función. Su única función es categorizar los campos.

Un ejemplo de grupo de campos es *Importes*, que agrupa datos relativos al importe de un producto: precio, precio tarifa, descuento, impuesto, moneda, etc.

#### *6.2.2.2.- Creación y ventanas*

En la ventana principal es posible ver un listado de todos los campos existentes en el Diccionario de Aplicaciones [ver imagen 6.47]. Este apartado consta de una ventana en la que se pueden editar los campos y crear nuevos y de una pestaña para traducir los textos.

Una vez creado el grupo, es posible asignarlo a un campo desde la ventana *Ventanas, solapas y campos > Solapa > Campos*.




| | Nombre | Activo |
|---|---|---|
| 1 | Action | Y |
| 2 | Actual Costs | Y |
| 3 | Amounts | Y |
| 4 | Assets | Y |
| 5 | Audit | Y |
| 6 | Bank | Y |
| 7 | Bank Account | Y |
| 8 | BP Tax Category | Y |
| 9 | Business Partner | Y |
| 10 | Cash Journal | Y |
| 11 | Common | Y |
| 12 | Company Details | Y |
| 13 | Contact Details | Y |
| 14 | Defaults | Y |
| 15 | Defaults Payment IN | Y |
| 16 | Defaults Payment OUT | Y |
| 17 | Dimensions | Y |
| 18 | Document | Y |
| 19 | Email template | Y |
| 20 | External | Y |

Imagen 6.47: Listado de grupos de campos

### 6.2.3.- Inputs auxiliares

En este apartado vamos a ver qué es un input auxiliar y cómo crearlo.

#### *6.2.3.1.- Introducción*

Un input auxiliar es un campo asociado a una pestaña de forma que pueda acceder a datos no definidos en la misma que no sean variables de sesión. Se usa de forma análoga a los campos de la pestaña, pudiendo definir valores por defecto de otros campos, lógicas de mostrado, etc.

Por ejemplo, un input auxiliar serviría para seleccionar automáticamente en un formulario de pago la forma de pago deseada. En caso de que solo se desee cobrar en metálico, el input auxiliar forzaría a que el desplegable que contiene las formas de pago posibles solo se pudiera elegir una, en metálico. Aunque las otras opciones existan, se fuerza al usuario a seguir la regla.

#### *6.2.3.2.- Creación y ventanas*

Para configurar un Input auxiliar se requiere elegir la pestaña sobre la que actuará, indicarle un nombre para el input auxiliar, e indicar un código de validación que establezca su valor en cada momento [ver imagen 6.48]. Este código debe ir implementado en HSQL.

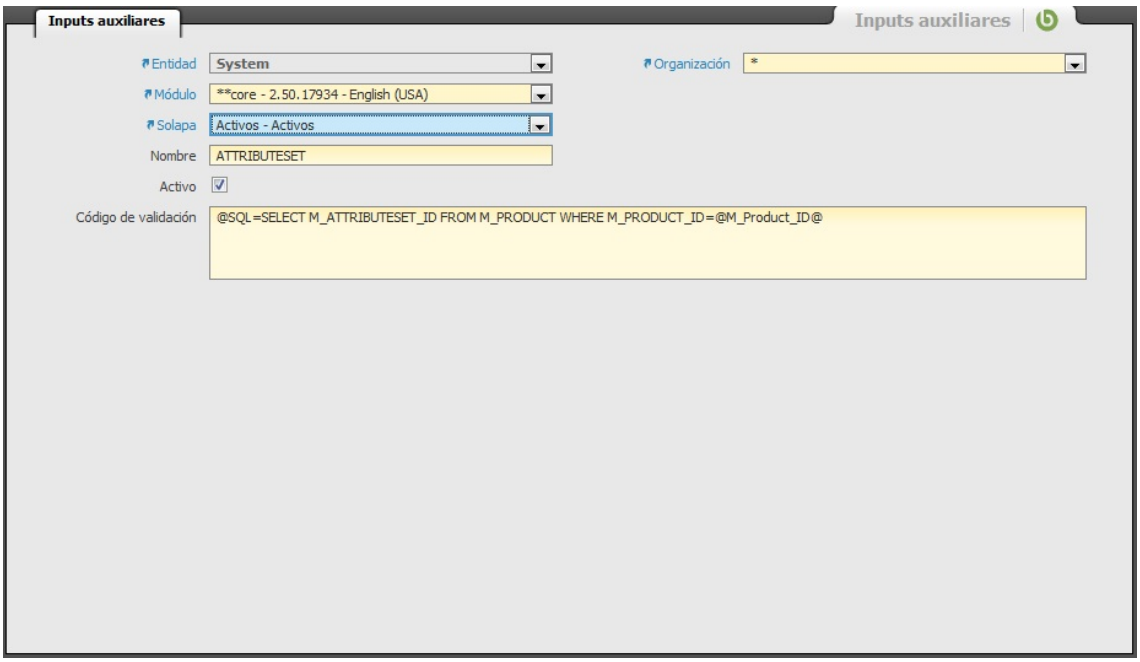



Imagen 6.48: Ventana de creación de un input auxiliar

### 6.2.4.- Callout

En este apartado vamos a estudiar la funcionalidad de los *Callout* y de cómo configurarlos en OpenbravoERP.

#### *6.2.4.1.- Introducción*

Un *Callout* es una pieza de código Java que es invocada cuando un campo es modificado. Está asociado a la columna de cuyo campo se lanzará la acción.

Los *Callouts* se suelen utilizar para:

- Recuperar o cambiar dinámicamente información de la Interfaz Gráfica después de que el usuario la modifique. Por ejemplo escribir el precio cuando un usuario selecciona un producto.
- Mostrar mensajes después de la modificación de un campo.
- Ejecutar una función JavaScript.

#### *6.2.4.2.- Creación*

Los *Callout* están implementados con *Servlets* Java, así que pueden realizar prácticamente cualquier acción. Se debe tener en cuenta que son invocados cuando el campo el modificado, esto ocurre antes de almacenar su valor en base de datos, así que es posible que el registro no exista en base de datos todavía.

Es posible que la funcionalidad de un *Callout* se parezca a la de un *Trigger*, pero no es así. La principal diferencia entre ambos es que:

- **Los *Callouts* se ejecutan en el *front-end***: es la parte de la aplicación que da la cara al usuario. Los *Callouts* tienen acceso a la interfaz gráfica, pueden modificarla, etc.

- **Los *Triggers* se ejecutan en el *back-end***: es la parte de la aplicación que no puede ver el usuario. Los *Triggers* se almacenan en la base de datos y tienen mayor comunicación con ella. Desde aquí es más recomendable ejecutar las funciones que se encargan de la consistencia de la base de datos.

### *6.2.4.3.- Ventanas*

La primera ventana que aparece nos va a permitir registrar los Callouts [ver imagen 6.49]. Para ello tan solo deberemos indicar el módulo a que pertenece, su nombre y su descripción.

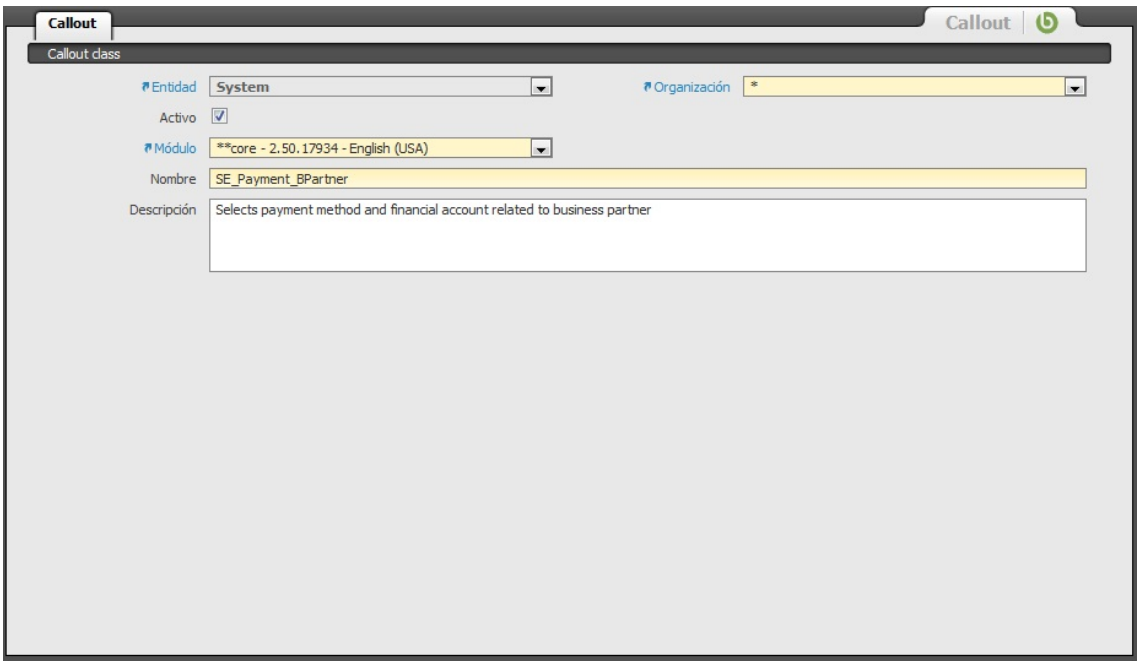



Imagen 6.49: Ventana de creación de un *Callout*

Una vez creado, en la pestaña “Callout class” identificaremos qué clase Java va a implementarlo [ver imagen 6.50]. La clase debe ir mapeada (asociarla a una URL) cuando el *Callout* tenga alguna interacción con la Interfaz de usuario. Para ello utilizaremos la pestaña “Callout mapping”.

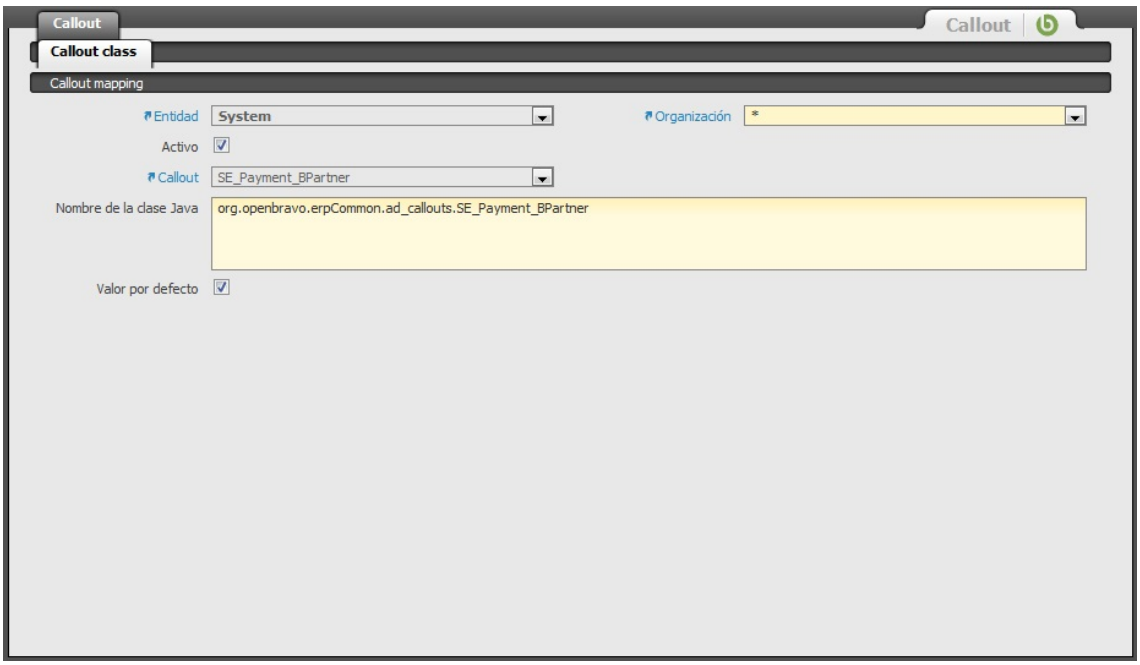



Imagen 6.50: Configuración de la clase de un *Callout*

### 6.2.5.- Reglas de validación

En esta ventana se puede definir todas las reglas de validación existentes en la aplicación para la correcta introducción y mantenimiento de los datos.

#### *6.2.5.1.- Introducción*

Las reglas de validación consisten en una consulta SQL (la cláusula *where,* es decir, la condición) que será aplicada a la consulta que se ejecuta para recuperar la información que contiene una lista desplegable. Es posible hacer referencia a las columnas de las tablas ajenas en el código de validación.

Por ejemplo, para comprobar que una columna pertenece a una tabla que ya existe, se utiliza una regla de validación. Evitando así la introducción de columnas referidas a tablas que no existen.

#### *6.2.5.2.- Creación y ventanas*

En la ventana de creación de una regla de validación debemos de configurar su nombre, el tipo de lenguaje a utilizar para montar el filtro, y el código que queremos que actúe de filtro [ver imagen 6.51].

Es posible realizar el filtro en distintos lenguajes: Java, JavasSript o SQL. Pero la mayoría de reglas utilizan SQL gracias a su flexibilidad y a que es un lenguaje pensado para realizar consultas.

Es posible utilizar variables se sesión en el filtro. Las variables de sesión están rodeadas por los símbolos @ y dentro contienen el nombre predefinido de una variable. Por ejemplo utilizaríamos @AD_Client_ID@ para hacer referencia al identificador del cliente que ha iniciado sesión en la aplicación.

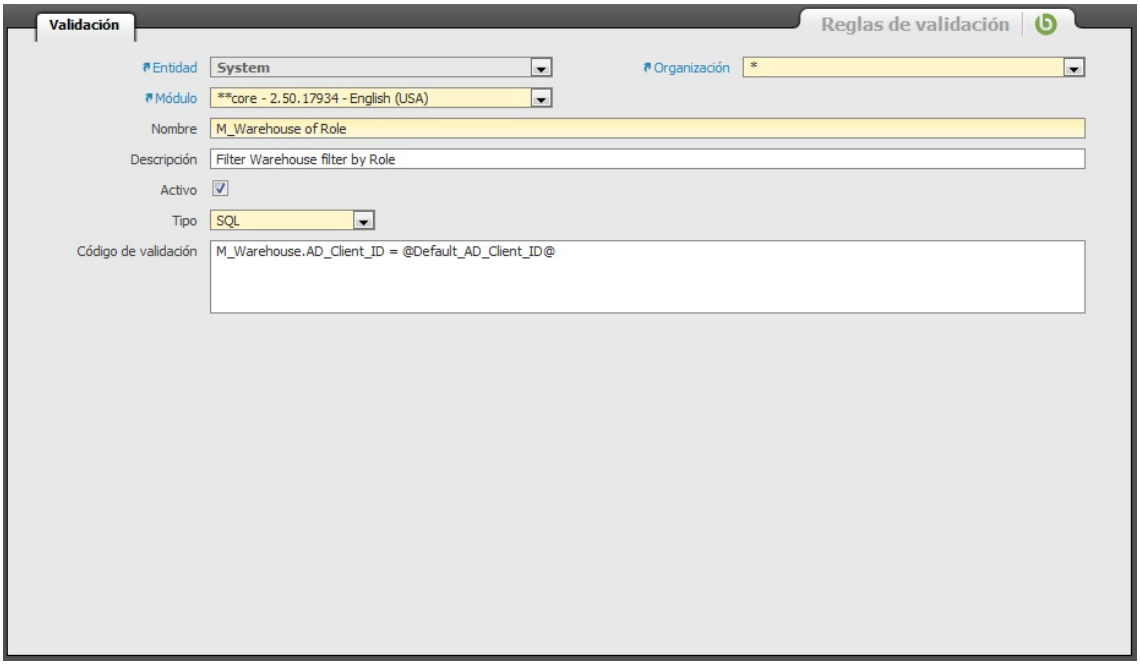



Imagen 6.51: Creación de una regla de validación

### 6.2.6.- Mes

En este apartado vamos a ver cómo gestiona OpenbravoERP los meses del año.

#### *6.2.6.1.- Introducción*

OpenbravoERP es un sistema multi-lenguaje que está pensado para ser instalado en cualquier país. Desde esta opción se permite configurar los meses del año para que sean utilizados luego por todo el sistema.

#### *6.2.6.2.- Creación y ventanas*

En la ventana principal de los meses podemos ver un listado con todos los meses que hay registrados en el sistema [ver imagen 6.52]. Desde la misma ventana es posible crear o editar los meses, indicando el nombre y el trimestre al que pertenecen.

En la pestaña “Traducción” es posible configurar la traducción del nombre del mes en otros idiomas.

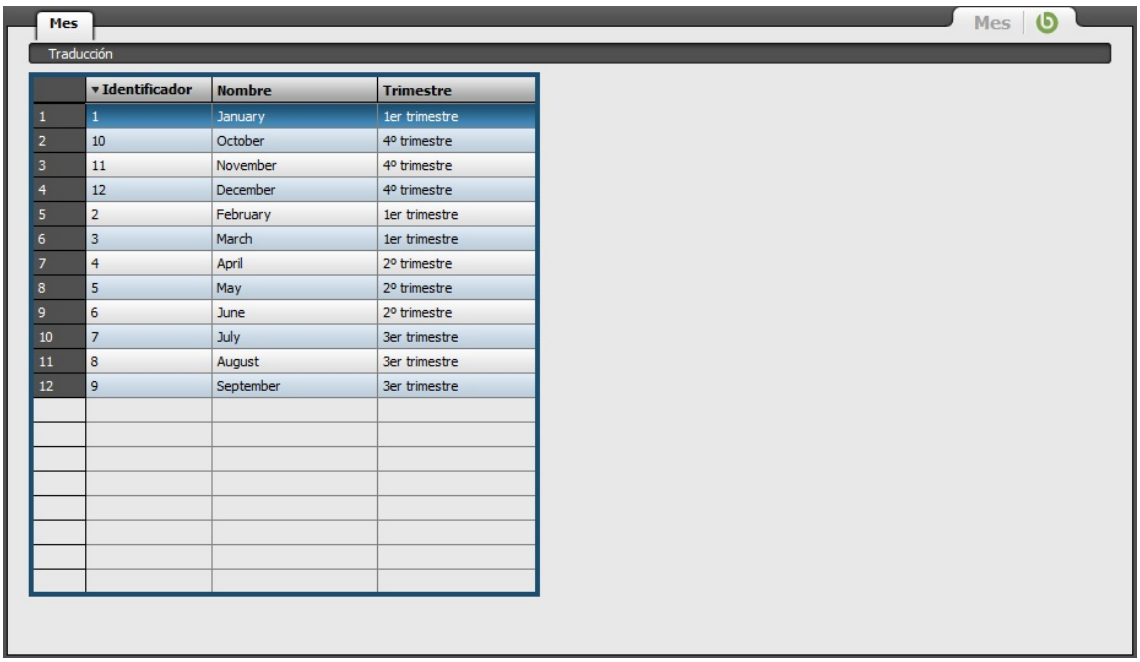



Imagen 6.52: Ventana con el listado de los meses del sistema

### 6.2.7.- Puntos de Extensión

En este apartado vamos a estudiar lo que es un punto de extensión, para qué sirve y cómo configurarlo en OpenbravoERP.

#### *6.2.7.1.- Introducción*

Los puntos de extensión son una herramienta para añadir funcionalidad en algunas de las acciones que realiza el sistema. La forma de incluir nueva funcionalidad es empleando procedimientos almacenados programados en PLSQL y asociarlos a los puntos de extensión existentes.

El inconveniente de los puntos de extensión es que solo ciertas partes del código están preparadas para poder extenderlas. Es decir, no se puede realizar cualquier acción, solo se puede en aquellos puntos que están configurados para ello. Además, la creación de nuevos puntos de extensión está restringida y solo los desarrolladores de Openbravo pueden crearlas.

La ventaja es que para los puntos de extensión ya creados, es posible añadirle casi cualquier funcionalidad de una forma muy sencilla. Por ejemplo, es posible añadir un procedimiento para que sea ejecutado cuando un Pedido es procesado.

#### *6.2.7.2.- Creación*

Un módulo de extensión puede ser definido en cualquier procedimiento de cualquier módulo.

Todos los puntos de extensión son parecidos, se emplea código en el procedimiento donde se quiere añadir funcionalidad y se registra en la ventana de los puntos de extensión.

Los puntos de extensión disponibles por ahora son:

- **C_Order_Post**: es invocado al final de la función *C_Order_Post1*, que se encarga de procesar los pedidos.
- **C_Invoice_Post**: es invocado al final de la función *C_Invoice_Post*, que se encarga de procesar las facturas.
- **M_Inout_Post**: es invocado al final de la función *M_Inout_Post*, que se encarga de completar las remesas.
- **M_Inout_Create**: es invocado dentro de la función *M_Inout_Create*, que se encarga de crear las remesas a partir de los pedidos.
- **M_Inout_Cancel**: es invocado dentro de la función *M_Inout_Cancel*, que se encarga de cancelar las remesas de los pedidos.

#### *6.2.7.3.- Ventanas*

La ventana principal permite gestionar los puntos de extensión de la aplicación. Es posible crear nuevos puntos asociados a algún módulo en desarrollo, pero no es posible crear o modificar los del núcleo.

En esta ventana hay que configurar el módulo donde va a estar el punto de extensión, su nombre y descripción [ver imagen 6.53].

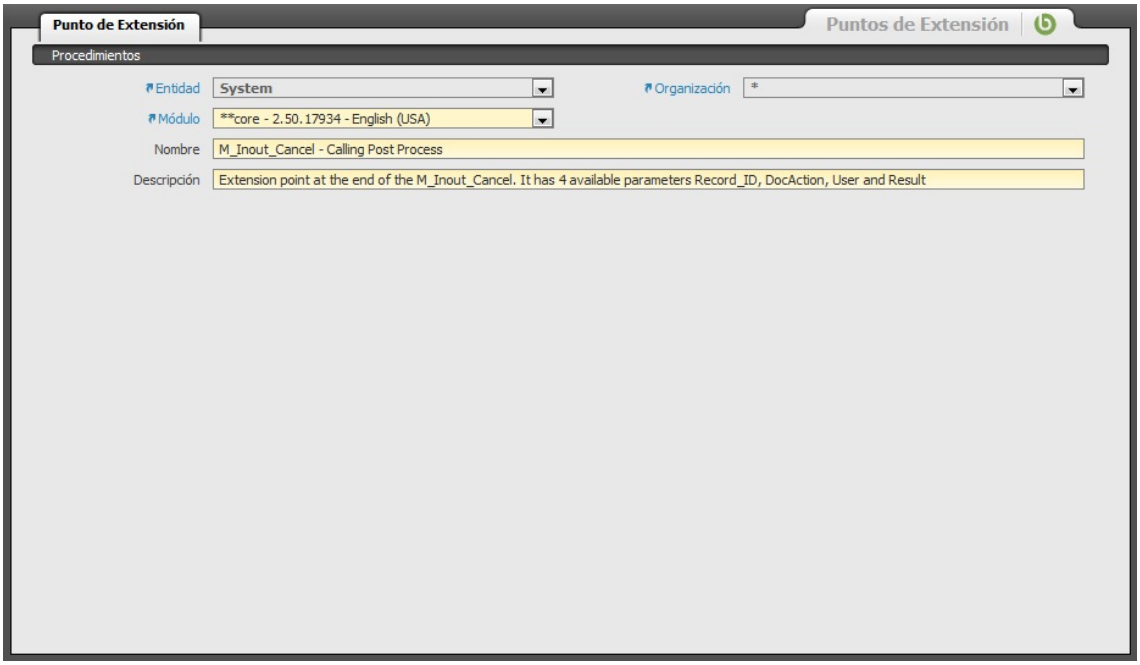



Imagen 6.53: Ventana de creación de un punto de extensión

En la pestaña de “Procedimientos” es posible añadir todos los procedimientos que van a añadir funcionalidad para ese punto de extensión. Tan solo deberemos indicar en qué módulo está el procedimiento y más abajo, su invocación [ver imagen 6.54].

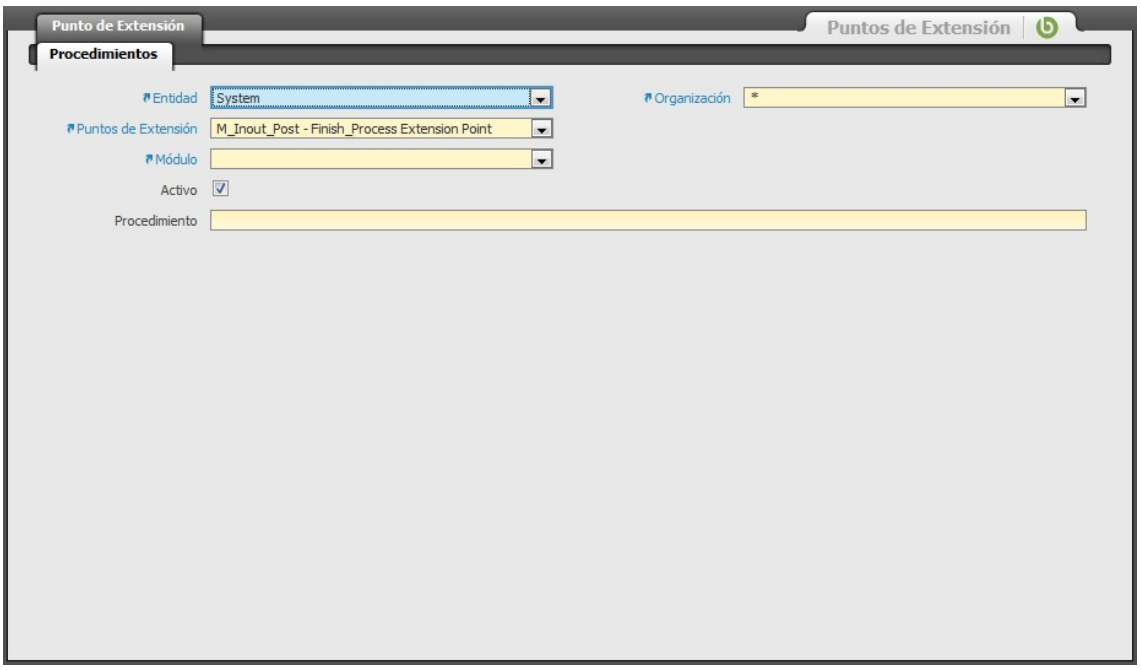



Imagen 6.54: Ventana de asociación de un procedimiento

## 6.3.- Mantenimiento

En este apartado vamos a ver las opciones que da OpenbravoERP para realizar un mantenimiento de su base de datos.

### 6.3.1.- Prueba

La ventana de pruebas nos va a permitir hacer un chequeo de las referencias de OpenbravoERP.

#### *6.4.1.1.- Introducción*

A la hora de desarrollar un módulo, actualizar el sistema o simplemente realizar alguna tarea de mantenimiento, es posible que se cree alguna inconsistencia en el sistema.

Gracias a la ventana de Prueba, el Diccionario de Aplicaciones nos permite comprobar el correcto funcionamiento de las Referencias.

#### *6.4.1.2.- Creación y ventanas*

En la ventana principal vamos a poder crear nuevas pruebas, o almacenarlas para luego chequear el sistema más adelante mediante el botón “Procesar ahora”.

Esta ventana contiene un campo por cada tipo de Referencia en el sistema [ver imagen 6.55]. Nos encontramos con un campo que nos pregunta por un número entero, por un importe, por una fecha, etc.

El funcionamiento consiste en rellenar cada uno de los campos, almacenar la prueba, y lanzar el proceso “Procesar ahora” para que el Diccionario de Aplicaciones pueda realizar la comprobación de la consistencia de las Referencias.

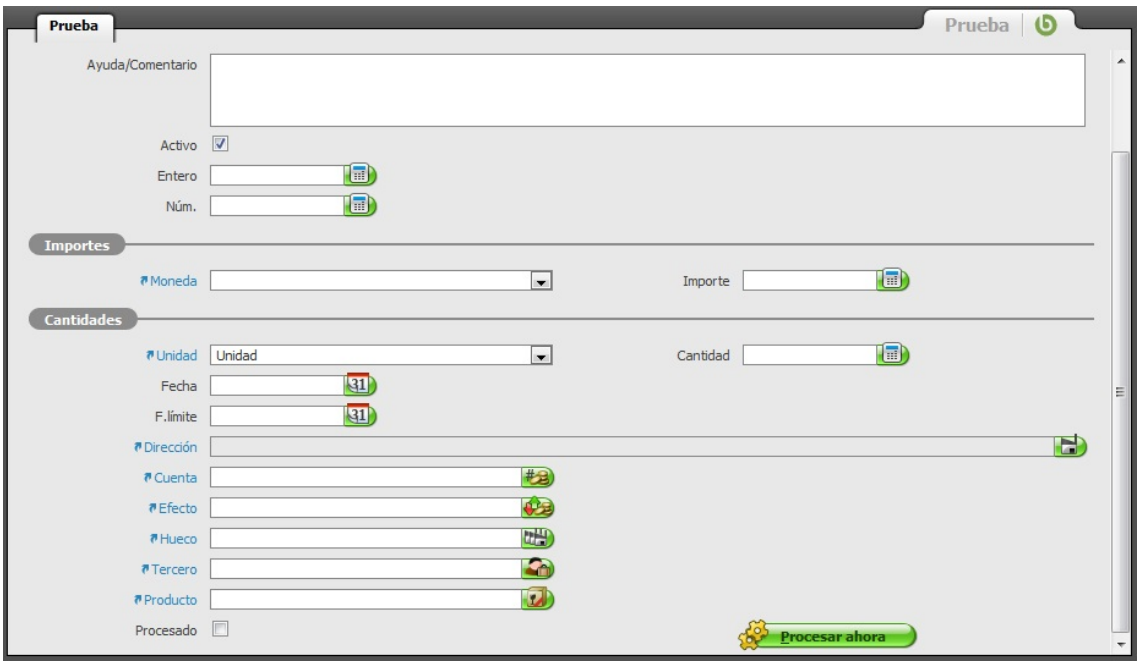



Imagen 6.55: Ventana de creación de pruebas

### 6.3.2.- Actualiza Identificadores tablas

En este apartado vamos a ver la funcionalidad del actualizador de identificadores de las tablas.

#### *6.3.2.1.- Introducción*

El actualizado de identificadores de las tablas es un proceso de mantenimiento cuya función principal es mantener la consistencia de base de datos.

En base de datos, la mayoría de los registros están relacionados a otras tablas. Por ejemplo, cuando se crea un pedido existe una relación con un cliente. Esto en base de datos se refleja mediante una relación o clave ajena (integración referencial). Esta relación se visualiza en una ventana mediante un desplegable que visualiza la información deseada del cliente.

#### *6.3.2.2.- Proceso*

Este proceso se encarga del mantenimiento de todos los campos relacionados, actualizando los identificadores de cada tabla.

Cada vez que se crea una ventana que contiene alguna relación a otras tablas es aconsejable ejecutar este proceso. Basta con aceptar la actualización [imagen 6.55] y a los pocos minutos finaliza el proceso indicando los registros actualizados.

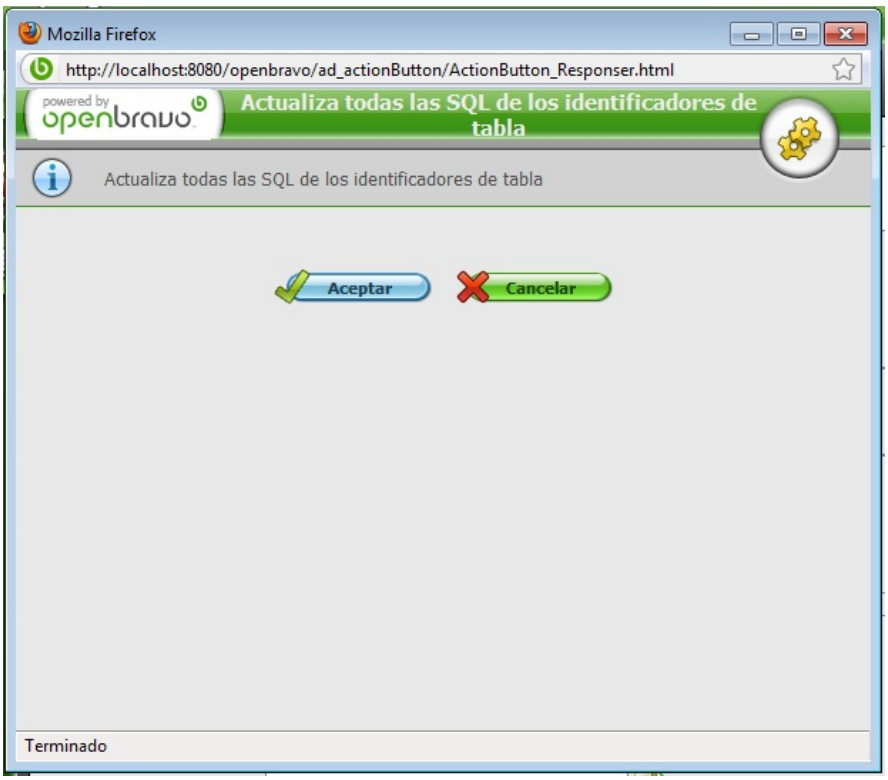



Imagen 6.55: Proceso actualizador de identificadores

### 6.3.3.- Recompilar objetos DB

En este apartado vamos a estudiar la funcionalidad de este proceso.

#### *6.3.3.1.- Introducción*

El proceso Recompilar objetos DB se encarga de compilar todos los artefactos de la base de datos: funciones, procedimientos almacenados, *triggers*, vistas, etc.

Esto nos servirá para, después de haber implementado código en la base de datos, comprobar su sintaxis y que está correctamente configurado.

#### *6.3.3.2.- Proceso*

Para invocar este proceso tan solo es necesario navegar a la opción *Diccionario de la Aplicación > Mantenimiento > Recompilar objetos DB* y hacer clic en el botón Aceptar.

Internamente, el proceso invoca la función del sistema *DBA_RECOMPILE_DB*.

### 6.3.4.- Consulta SQL

En este apartado vamos a ver la funcionalidad del proceso Consulta SQL.

#### *6.3.4.1.- Introducción*

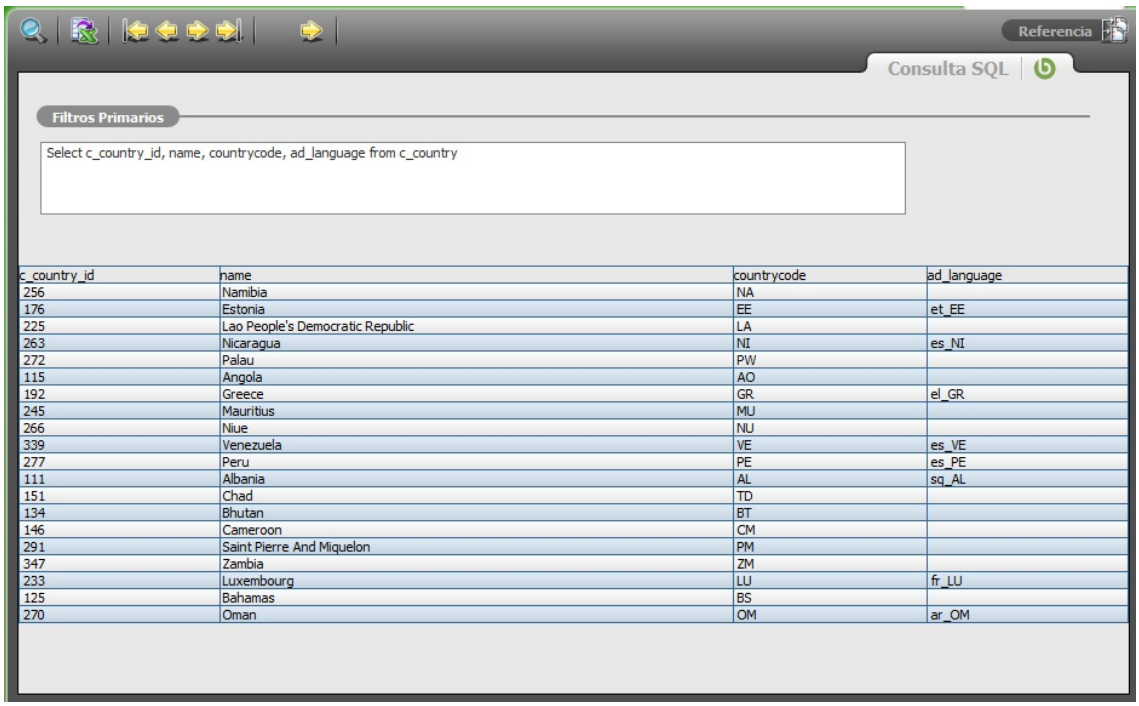
Para las tareas de mantenimiento es necesario a veces conocer qué información hay exactamente en las tablas de base de datos.

Con el proceso Consulta SQL se permite realizar consultas directamente a la base de datos de OpenbravoERP. Es como un cliente de base de datos, como pgAdmin 3, pero que solo sirve para realizar consultas. Esto es útil para cuando el administrador del sistema solo quiere conocer algún dato de las tablas del sistema.

#### *6.3.4.2.- Proceso*

Para ejecutar una consulta SQL bastan con entrar en la opción *Diccionario de la Aplicación > Mantenimiento > Consulta SQL*, introducir la consulta y ejecutarla mediante el botón Lupa [ver imagen 6.57].

Es importante tener en cuenta que en este proceso solo se permiten realizar consultas. Cualquier proceso que implique la modificación de la base de datos (actualización, inserción o borrado) está bloqueado su uso.




| c_country_id | name | countrycode | ad_language |
|---|---|---|---|
| 256 | Namibia | NA | |
| 176 | Estonia | EE | et_EE |
| 225 | Lao People's Democratic Republic | LA | |
| 263 | Nicaragua | NI | es_NI |
| 272 | Palau | PW | |
| 115 | Angola | AO | |
| 192 | Greece | GR | el_GR |
| 245 | Mauritius | MU | |
| 266 | Niue | NU | |
| 339 | Venezuela | VE | es_VE |
| 277 | Peru | PE | es_PE |
| 111 | Albania | AL | sq_AL |
| 151 | Chad | TD | |
| 134 | Bhutan | BT | |
| 146 | Cameroon | CM | |
| 291 | Saint Pierre And Miquelon | PM | |
| 347 | Zambia | ZM | |
| 233 | Luxembourg | LU | fr_LU |
| 125 | Bahamas | BS | |
| 270 | Oman | OM | ar_OM |

Imagen 6.57: Ventana para la ejecución de consultas SQL

## 6.4.- Test Automático

En este último apartado del Diccionario de Aplicaciones se gestionan una serie de elementos para realizar tests.

### 6.4.1.- Introducción

Los tests son pruebas que se realizan a los nuevos desarrollos para comprobar su que su funcionalidad es la esperada y que el sistema sigue siendo consistente. Sin ellos, realizar un mantenimiento de todas las funcionalidades del sistema es una tarea muy costosa y repetitiva. Cada vez que se desarrolla un elemento que interacciona con otras funcionalidades resulta necesario probar todas ellas, ya que el nuevo elemento podría ser inconsistente con la funcionalidad existente. Es por ello que se van a utilizar herramientas especiales para realizar tests automáticos.

Existen muchas herramientas implicadas en el testeo de los desarrollos en OpenbravoERP. Podemos encontrar:

- **Selenium**: es una herramienta Web diseñada para testear aplicaciones Web. Incluye una suite de aplicaciones útiles para realizar tests automáticos.
- **JUnit**: es una poderosa herramienta para realizar casos de test en Java. Gestiona los parámetros y el comportamiento que se debe esperar de la aplicación.
- **FireBug**: es un componente adicional asociado al navegador FireFox. Permite analizar y testear la Interfaz Gráfica de la aplicación: ficheros XML, HTML, JavaScript, CSS, etc.
- **TestLink**: es una aplicación Web utilizada para gestionar los casos de test y poder compartirlos y coordinarlos entre los desarrolladores del sistema.

Los tests diseñados cumplir en la medida de lo posible:

- Ser simples. Cuanto más sencillo, más fácil de mantener.
- Ser independientes de otros tests.
- Realizar sólo una acción.
- Cargar sus precondiciones (entorno del test) por si mismo antes de la ejecución.
- No afectar a otros test o una nueva ejecución pos si mismos.

Se prefiere múltiples tests pequeños que se centren en la misma funcionalidad de la aplicación a que los tests sean grandes y complicados.

### 6.4.2.- Creación

En la última versión de OpenbravoERP 2.50 se incluye un submenú llamado Test Automático.

Los Tests Automáticos es una herramienta que permite gestionar los casos de tests desde la misma aplicación. A diferencia de las herramientas mencionadas en el apartado anterior, la integración de los Tests Automáticos es máxima para con el ERP.

Este apartado consta de los siguientes elementos [ver imagen 6.58]:

- **Documento**: su objetivo es describir y documentar un caso de tests. Estos test son la unidad más básica y sencilla para hacer una prueba.
- **Comando**: agrupa un listado de Documentos y para cada uno de ellos le configura una serie de parámetros de entrada para el test.
- **Instrucción**: configura un test automático con el cual se pueden realizar una serie de pruebas con un único fin. Cada Instrucción está compuesta por una serie de comandos, los cuales ejecutan pequeños casos de test.
- **TestLogin**: administra las instrucciones ejecutadas en el sistema.

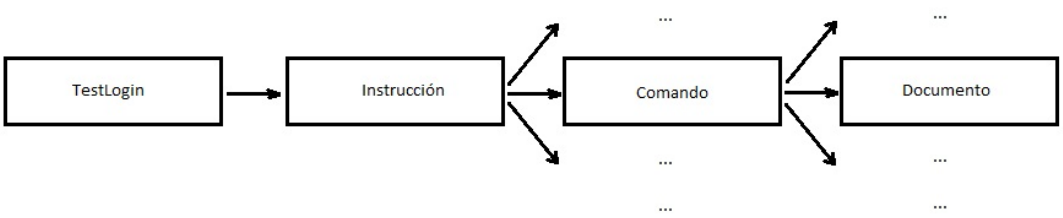



Imagen 6.58: Esquema de creación de un Test Automático

El Test Automático del Diccionario de Aplicaciones es una herramienta todavía en desarrollo y no existe prácticamente documentación sobre su funcionalidad. Es por ello que no podemos estudiar más a fondo las ventanas que lo componen.

# 7.- Desarrollo de prácticas docentes para incluir nuevas funcionalidades a OpenbravoERP

En este capítulo vamos a aplicar parte de lo aprendido a lo largo de la memoria para desarrollar dos prácticas didácticas. Como vimos en el primer capítulo, uno de los objetivos de este proyecto es el aprendizaje de las herramientas del ERP.

Estas dos prácticas pueden considerarse boletines de prácticas completamente usables ya que han sido probados en un entorno real. Cada una se centra en una funcionalidad concreta de OpenbravoERP:

- **Práctica 1: Implantación de OpenbravoERP**. Trata sobre las modalidades que permite configurar el sistema para realizar una implantación correcta.
- **Práctica 2: Ampliando la funcionalidad de OpenbravoERP**. En ella se realiza una pequeña modificación del sistema para demostrar su flexibilidad a la hora de ampliar su funcionalidad.

## 7.1.- Práctica 1: Implantación de OpenbravoERP

En la primera práctica didáctica nos vamos a centrar en las metodologías a utilizar a la hora de implantar el ERP en una empresa.

### 7.1.1.- Objetivo

El Objetivo de esta práctica es realizar una implantación básica del ERP Openbravo y con ello aprender la funcionalidad y el uso de las herramientas ETL para este tipo de tareas.

### 7.1.2.- Resumen

Empezaremos la práctica preparando el entorno de trabajo con el que vamos a realizar la práctica. Para ello utilizaremos una máquina virtual con el ERP pre-instalado para poder empezar cuanto antes con la implantación. En la segunda parte de la práctica importaremos un listado clientes desde un fichero CSV a OpenBravo utilizando las herramientas que nos ofrece el propio sistema. Y por último realizaremos otra importación de clientes, pero esta vez utilizando la herramienta Pentaho Data Integration, más comúnmente conocido como Kettle ETL.

### 7.1.3.- Introducción

Las herramientas ERP gestionan información muy valiosa de las organizaciones. Muchas veces, por necesidad o por el simple hecho de mejorar los Sistemas de Información, estas organizaciones necesitan actualizar su software o incluso desechar su software actual para comenzar a utilizar otras soluciones acorde con sus necesidades. Para ello es necesario mover todo ese conocimiento a otro formato de almacenamiento para que esas nuevas herramientas puedan entender esa información. Este proceso se conoce como “implantación”.

La implantación de un nuevo sistema supone estudiar el formato de almacenamiento del software origen y destino, de forma que se realice un "match" (emparejamiento) entre ambos para que se pueda transferir la información. La implantación es una tarea bastante meticulosa que requiere cierto orden.

En esta práctica vamos a ver dos posibles vías para realizar una importación de clientes en OpenBravoERP:

- Herramientas propias: por una parte podemos utilizar las herramientas que nos proporciona OpenBravo. Las ventajas son básicamente un fácil manejo y consistencia de la información dentro de la base de datos. Por contra, son de funcionalidad bastante limitada.

- Herramientas externas: son las llamadas ETL (Extraction, Transformation and Loading). Se encargan de estructurar unos flujos de datos para cumplir con la funcionalidad deseada. Suelen estructurarse con diagramas de flujo. Tienen la desventaja que para la implantación, implican un nivel de conocimiento de la base de datos destino mucho mayor, pero ofrecen una flexibilidad mucho mayor y no son dependientes de la plataforma.

### 7.1.4.- Desarrollo

#### *7.1.4.1.- El entorno de trabajo*

A continuación vamos a describir los pasos necesarios para poner en funcionamiento el entorno de trabajo necesario para poder realizar la práctica.

**Instalación de OpenbravoERP**

Primero de todo vamos a ejecutar en nuestro sistema el software OpenbravoERP para poder empezar a trabajar. Puesto que la instalación es un proceso que puede llevar unas horas y no es el objetivo de esta práctica vamos a utilizar una máquina virtual en la que viene ya correctamente instalado en el sistema operativo Linux[9]. Para poner en marcha el entorno de trabajo:

1- Ves a la siguiente dirección https://www.vmware.com/products/player/ y descárgate el cliente VMWare Player.

2- Instálatelo haciendo doble clic.

3- Ejecuta VMWare. Al abrirse el programa, haz clic en el botón "Open" y selecciona la máquina virtual[10] "c:\OpenbravoERP\OpenbravoERP-2.50MP6-x86.vmx".

---

[9] Vamos a utilizar lo que se denomina la aplicación virtual de OpenbravoERP. Podéis encontrar toda la documentación en http://wiki.openbravo.com/wiki/Virtual_appliances#Community_Appliance

[10] La máquina virtual de OpenbravoERP la podéis encontrar en http://sourceforge.net/projects/openbravo/files/01-openbravo-appliances/2.50MP8/

4- Una vez arrancado el SO, se nos pedirá en modo consola el usuario y contraseña de acceso al sistema, a lo que introduciremos “openbravo” en ambos. Para poder salir de la consola y poder controlar el cursor presionar: ctrl. + Alt

Una de las mejores bazas de Openbravo es que está pensado para que se pueda acceder desde cualquier ordenador conectado a Internet, por lo que se puede ejecutar desde un navegador Web.

Una vez preparada la máquina virtual tan solo tendremos que abrir un navegador desde nuestro SO y acceder a la dirección IP[11] que se indica en la consola virtual de Openbravo (por ejemplo http://192.168.2.101). Aparecerá una página Web con un formulario donde nos pedirá un usuario y su clave, a lo que introduciremos “Openbravo” para el usuario y “openbravo” para la clave.

**Conexión SSH a la Base de Datos**

Por razones de seguridad, el SO ejecuta un cortafuegos que bloquea cualquier acceso externo excepto los puertos 22 (SSH), 80 (HTTP) y 443 (HTTPS). Así que para conectarnos a la base de datos con una herramienta externa es necesario antes establecer un túnel SSH.

Para crear ese túnel vamos a utilizar el cliente SSH PuTTY. Empezaremos ejecutando el archivo putty.exe y seguiremos estos pasos [ver imágenes a continuación]:

1. Seleccionar la categoría **Session**.

2. En la caja **Host name**, poner la dirección IP de Openbravo: 192.168.2.101.

3. Seleccionar la categoría **Connection > SSH > Tunnels**.

4. En el campo **Source port** poner 5433.

5. En el campo **Destination** poner localhost:5432.

6. Hacer clic en **Add**.

7. Hacer clic en **Open** para abrir el túnel SSH. Es posible que abra una alerta de seguridad de Putty. Hacer clic en **Si**.

8. Poner el usuario “openbravo” y contraseña “openbravo”.

Para que las herramientas que acceden a la base de datos este túnel debe estar **Abierto** durante toda la sesión.

[11] De ahora en adelante, para facilitar la lectura de la práctica vamos a simular que la IP que nos proporciona la consola virtual de OpenBravo es 192.168.2.101 (lo más probable es que el VmWare Player os proporcione otra)

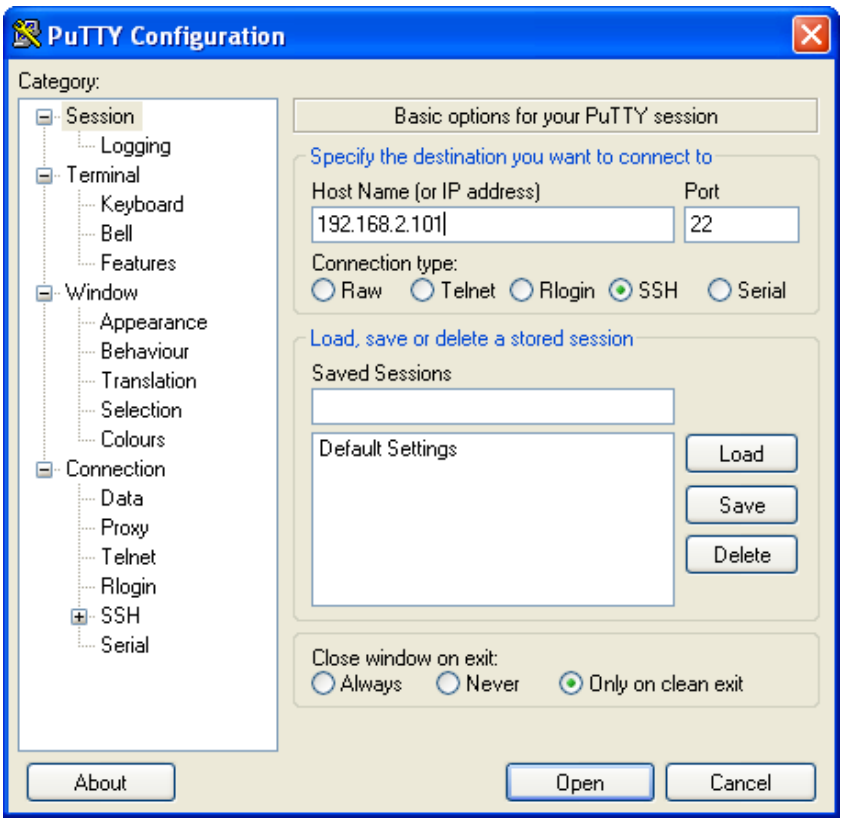



Imagen 7.1: Paso 1 configuración PuTTY

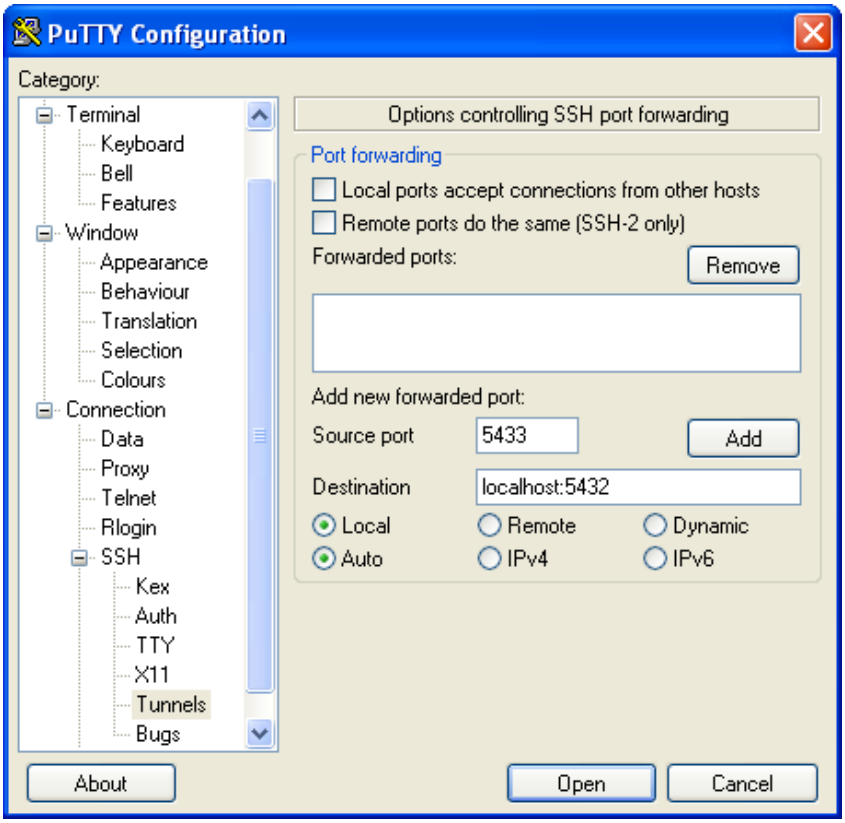



Imagen 7.2: Paso 2 configuración PuTTY

**Configuración de pgAdmin III**

Más adelante se nos va a presentar la necesidad de acceder a la base de datos de OpenBravo para consultar sus tablas. Para poder manipular la base de datos es necesario conectarnos al gestor PostgreSQL con el cliente pgAdmin III. A continuación vamos a configurar la conexión con este programa:

1. Ejecutar Inicio > Programas > pgAdmin III

2. Hacer clic en el botón  para añadir una nueva conexión.

3. Rellenar los siguientes datos:

    a. **Nombre:** OpenbravoERP

    b. **Servidor:** localhost

    c. **Port:** 5433

    d. **Username:** tad

    e. **Password:** tad

Una vez conectados desplegaremos la pestaña openbravo para poder ver todas las tablas (doble clic para conectarse al servidor).

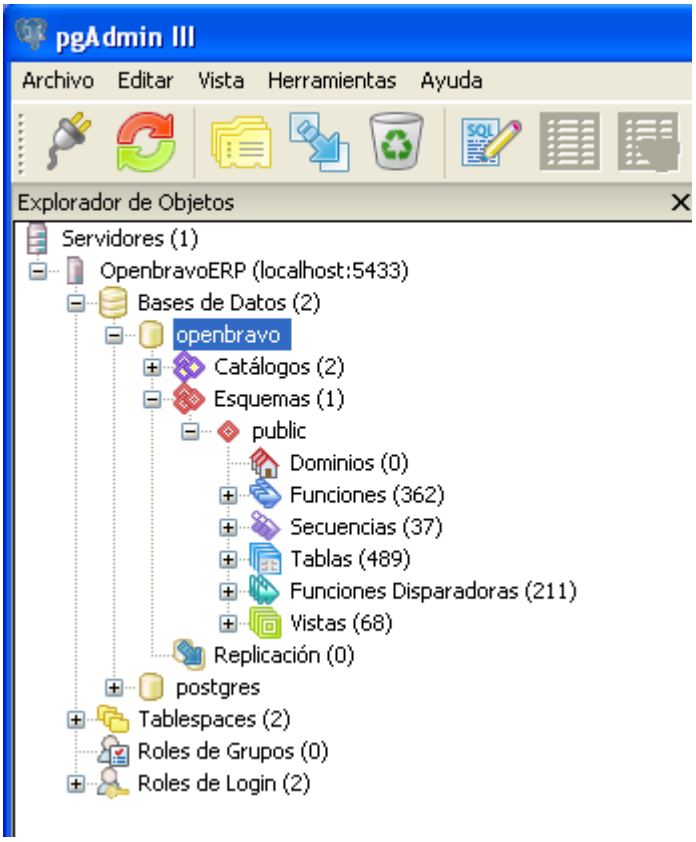



Imagen 7.3: Propiedades de la base de datos

Hasta aquí llega la configuración del entorno de trabajo. Una vez puestas en marcha todas las aplicaciones vamos a empezar con la implantación.

**Guardar el estado de la máquina virtual**

Cuando el PC del laboratorio se reinicia vuelve a un estado "original" y borra todas las modificaciones que haya habido en la anterior sesión. Para poder seguir el trabajo que hagáis en esta práctica para la siguiente sesión será necesario que guardéis el estado de la máquina virtual de OpenBravoERP en una memoria USB. Esto es, copiar el fichero *OpenbravoERP-2.50MP6-x86.vmdk* de 2GB aproximadamente, en un USB y en la próxima práctica restaurar el fichero del PC con el que tengáis en el USB.

### *7.1.4.2.- Implantación con herramientas propias*

En este apartado vamos a estudiar las herramientas que dispone OpenbravoERP para realizar la implantación de nuestro sistema. El objetivo principal va a ser importar información externa (como por ejemplo un listado de clientes que nos proporciona un antiguo software de la empresa) al nuevo sistema ERP.

Como hemos comentado antes, estas herramientas suelen ser limitadas ya que es muy complejo abarcar todas las funcionalidades posibles y es por eso que Openbravo acota la funcionalidad básica a la importación de los siguientes ítems:

- Business Partners (Clientes, proveedores, patrocinadores, etc.)
- Productos (products)
- Cuentas (accounts)
- Pedidos (orders)
- Presupuestos (Budgeds)
- Impuestos (taxes)

Nuestro objetivo en este apartado va a ser el importar a OpenbravoERP los clientes de nuestra empresa. Los clientes vienen descritos en el fichero *clientes1.csv*:

```
Codigo,Nombre
LA001,Ladrillos Nueva Era s.l.
LA002,Ladrillos Orange s.l.
LA003,Gutierrez e hijos s.l.
LA004,Obras Lituania s.l.
LA005,Construcciones Laberínticas s.l.
```

La importación de estos datos se realiza en tres pasos:

1. Definir el formato CSV.
2. Importar el fichero CSV a una tabla temporal.

3. Volcado de la tabla temporal.

**Paso 1: Definir el formato CSV**

Los datos a importar han de estar en un fichero CSV (*comma separated file*). Es necesario primero de todo establecer el formato del CSV, es decir, el orden de las columnas.

A la hora de importar es necesario que toda la información esté correctamente definida para que el proceso no falle. Por ejemplo, los grupos de clientes de la empresa deben existir en Openbravo antes de realizar la importación, ya que si no fallará. Esto significa que si hay algún error en el proceso de importación hay probablemente algún error en los datos de importación.

Para realizar cualquier tipo de importación asegúrate de tener permisos de administrador. Vamos a comenzar con la definición del formato:

1. Selecciona *Master Data Management* > *Import Data* > *Import Loader Format*. Como puedes observar, ya hay una serie de formatos predefinidos y que podemos utilizar, pero en nuestro caso vamos a crear uno nuevo que se amolde al fichero que queremos importar.

2. Pulsa el botón para crear un nuevo registro.

3. Pon en el campo Name: “Importación de clientes básica”.

4. Cuando se crea un nuevo formato hace falta seleccionar una tabla temporal (I_xxx) donde irá la información. Cada tabla temporal está asociada a uno de los ítems mencionados anteriormente. Como vamos a importar clientes, selecciona *I_BPartner*.

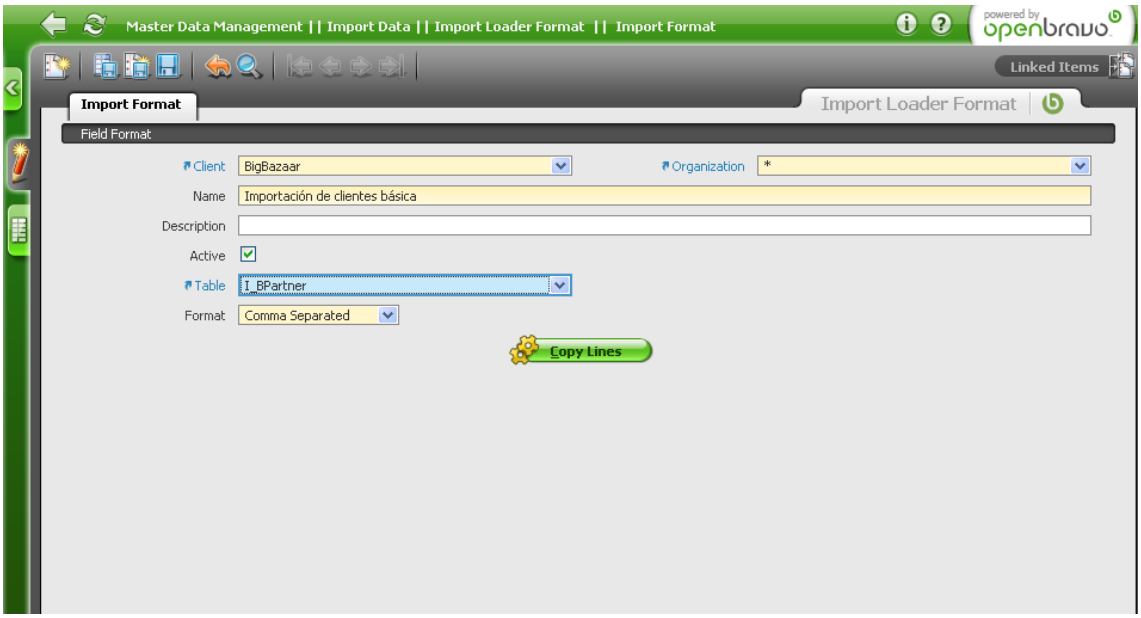



Imagen 7.4: Paso 1, definición del nuevo formato de importación

5. Haz clic en el tab *Field format*. Por cada columna que haya en nuestro fichero csv deberemos poner una fila en esta lista. Cabe decir que el nombre de la columna no importa, solo el orden.

6. Haz clic en el icono de añadir.

7. Podrás ver un campo que se llama *Sequence Number*. Es el número que se le asigna a la columna para indicar el orden por el que se deben leer todas las columnas. Cuanto menor sea, antes se leerá. En nuestro caso vamos a crear las columnas en orden, así que no es necesario modificar este campo.

8. Nombra el primer campo como “Clave” y asócialo a la columna “Value” (el despegable contiene todas las columnas de la tabla c_bpartner). El campo *Data Type* debe ser “String”.

9. Guarda el registro.

10. Crea un nuevo registro y nómbralo “Nombre”, asócialo a la columna “Name” y con el *Data Type* igual a “String”.

11. Guarda el registro.

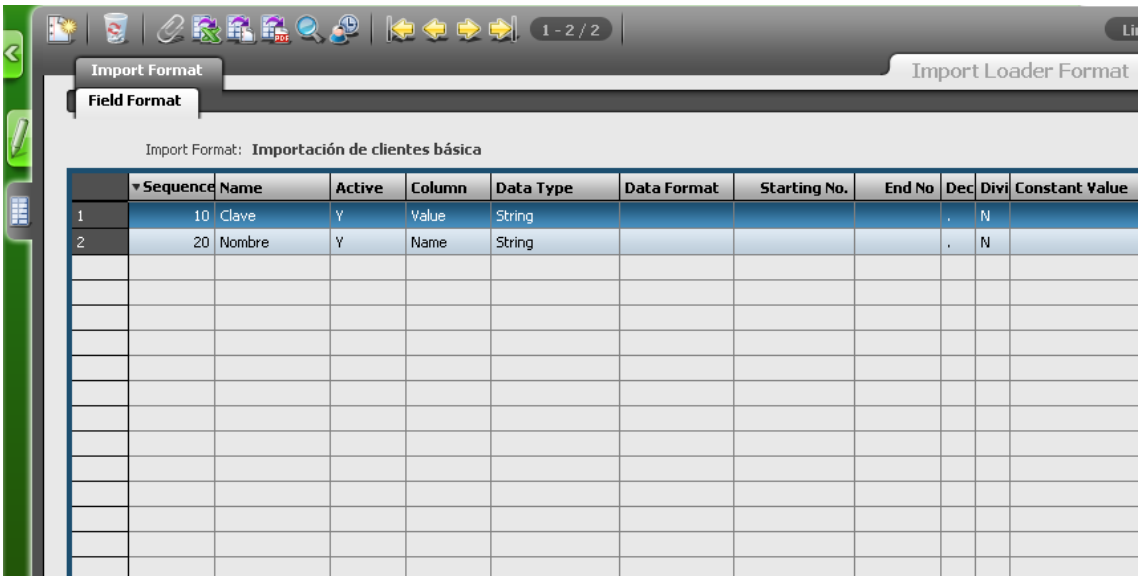



Imagen 7.5: Paso 1, definición de las columnas del formato de importación

Es posible que en alguna de las columnas hagan referencia a un ID (por ejemplo, c_bpartner_id que es el número interno de Openbravo para identificar a un cliente). En caso de que quisiéramos asignarle ese número a una columna la única forma de ver estos IDs es acceder a la base de datos directamente, por ejemplo con la herramienta pgAdmin III.

Nota: los campos definidos como una constante no deben aparecer en el fichero a importar.

Cada importación tiene unos valores obligatorios dependiendo de la tabla donde quieras insertar los valores. Por ejemplo:

- Bussines partners: los valores obligatorios son nombre y clave (valor de búsqueda). Por lo que no necesitaremos insertar ninguna columna más en el fichero *clientes1.csv*.

- Productos: los valores obligatorios son: clave, nombre, tipo de producto, cliente y categoría.

Y continuaría por cada tipo de ítem.

**Paso 2: Importar el fichero CSV a una tabla temporal**

Después de definir el formato de importación vamos a importar la información que contiene el fichero a Openbravo:

1. Selecciona *Master Data Management* > *Import Data* > *Import File Loader*.

2. Selecciona el fichero a importar *clientes1.csv* y el formato "Importación de clientes básica.

3. Selecciona *Header first line* para indicar que el fichero CSV contiene el nombre de las columnas.

4. Para probar la importación, haz clic en *Ok*. Si todo ha ido bien verás la información del fichero debajo del todo.

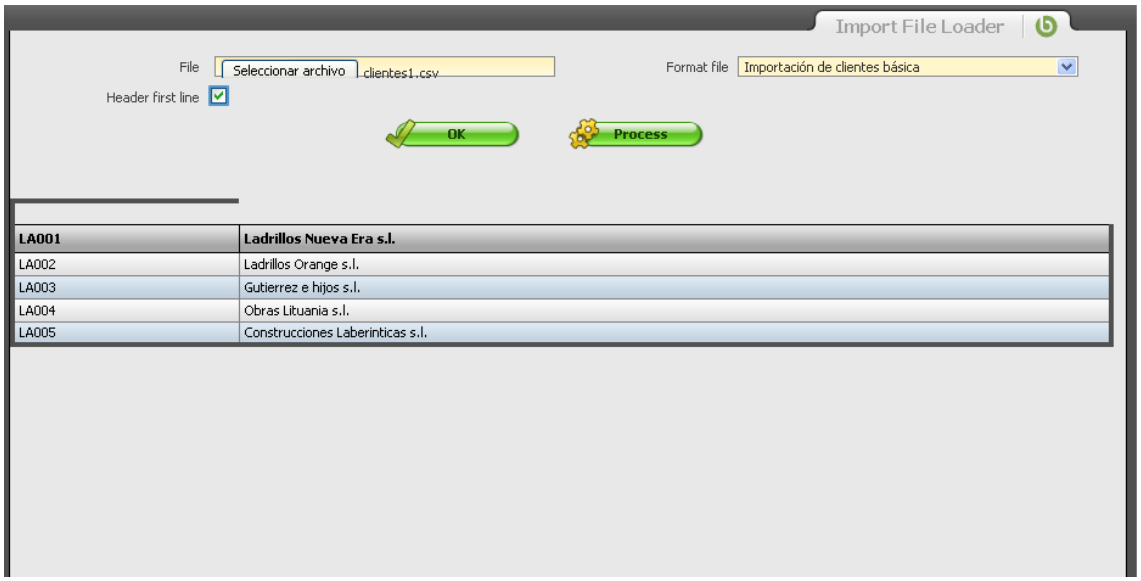



Imagen 7.6: Paso 2, Importación a la tabla temporal

5. En caso de que la información sea correcta haz clic en *Process*. Este paso carga el contenido del fichero CSV en la tabla temporal asociada al formato. Aparecerá un mensaje indicando que el proceso ha importado correctamente 5 registros.

En caso de que se visualizara algún error sería necesario revisar tanto el contenido del fichero CSV como la definición del formato. Los errores comunes son:

- No todos los valores están separados por comas.
- El número de las columnas no coincide con el número de filas definida en el formato.
- Hay una línea vacía al final del fichero CSV.
- No se han indicado todos los valores obligatorios de la tabla en el fichero CSV.

**Paso 3: Volcado de la tabla temporal**

Para finalizar, se nos permite realizar un filtro sobre los registros que se van a insertar antes de que pasen definitivamente a nuestro ERP. Podemos descartar registros, modificar la información que había en el fichero e incluso asignar valores a nuevos campos.

1. Selecciona *Master Data Management* > *Import Data* > *Import Bussines Partner*. Aquí es donde vamos a visualizar los campos importados a la tabla temporal. Uno a uno vamos a ir validándolos.
2. Para empezar, vamos a indicar que la organización a la que pertenezca el cliente sea la de California.
3. En el campo *Business Partner Category* haz clic en “SELLER”.
4. Selecciona el país “United States” y estado “CA”.
5. Navega con los botones  y asigna a todos los registros los mismos valores. Invéntate algunos de los valores como descripción, dirección o código postal.
6. Ves abajo del todo de cualquier registro y haz clic en el botón *Import business partner*. Aparecerá una nueva ventana indicándote que a algunos de los valores vacíos se les asignará valores por defecto y que esto no supone la sobreescritura de la información. Haz clic en *Ok*.
7. Openbravo te indicará que todas las filas han sido correctamente insertadas. Si miras abajo del todo de cualquier registro verás un check que indica que el registro en el que estás se ha procesado correctamente.

**Ejercicio 1: Importación con herramientas propias**

**1.1 – Ves al listado *Master Data Management* > *Business Partners* y visualiza un listado de los clientes que acabas de importar. Haz una captura de pantalla y súbela a PorliformaT.**

### *7.1.4.3.- Implantación con herramientas externas*

En este apartado vamos a estudiar una herramienta llamada Kettle ETL que nos va a permitir realizar la misma funcionalidad pero de una forma más extensible y automatizada.

**Kettle ETL**

Pentaho Data Integration (o también llamado Kettle) es una herramienta libre para las funciones de manejo de datos (ETL). Un ejemplo de su uso sería crear un Servicio Web para que recogiera información de una base de datos y la almacenara directamente en otra base de datos. En nuestro caso vamos a utilizar la herramienta para importar información externa a OpenBravo.

Kettle está compuesto por estos programas:

- **Spoon:** es un entorno gráfico que permite, de forma visual, configurar acciones (transformaciones y trabajos). Los trabajos y las transformaciones son ficheros que contienen los procesos que necesita Kettle para realizar toda la lógica deseada. Una transformación se centra en cómo manejar el flujo de datos (entradas, transformaciones, salidas, etc.) y un trabajo se centra en cuándo ejecutar esas transformaciones (tareas diarias, backups, etc.).
- **Carte:** es un sencillo servidor web que permite acceder remotamente al repositorio de Kettle, y ejecutar, monitorizar, empezar y parar acciones que se ejecutarán en el servidor.
- **Pan:** permite ejecutar acciones diseñadas en Spoon desde la línea de comandos.
- **Kitchen:** permite ejecutar trabajos en *modo batch* (procesamiento por lotes). Es decir, ejecutar trabajos sin la supervisión de ningún usuario.

Esta arquitectura permite reducir el esfuerzo y minimizar el riesgo al modificar o implementar una nueva funcionalidad en las tareas que se desean llevar a cabo.

Para ver el entorno de trabajo ejecuta el programa: C:\kettle\data-integration\Kettle.exe (esta herramienta, al estar basada en java necesita tener instalado el runtime de java JRE5).

Una vez abierto el programa podremos distinguir tres apartados (si aparece una ventana preguntando el catálogo, hacer clic en “Sin Catálogo”):

- **View**: describe todos los elementos usados (steps, hops, jobs, database connection...).
- **Design**: las operaciones que se pueden usar en Kettle.

- **Tabs**: en el area principal se desplegarán unas pestañas que contienen transformaciones o tareas. Cada fichero tiene un menú que permite realizar operaciones como: ejecutar, debug, explorar base de datos

Aparecerá una ventana para seleccionar un catálogo (los catálogos o repositorios almacenan las tareas/tranformaciones en una base de datos). En nuestro vamos a utilizar el programa sin catálogos para que las tareas se guarden en ficheros.

**Ejemplo: Hola Mundo**

En este ejemplo vamos a realizar una transformación (las transformaciones son un conjunto de *Steps* (acción) unidos por *Hops* (conexión) que, todo en conjunto, forman un flujo de datos) cuyo objetivo es procesar un fichero CSV con una lista de nombres para obtener un fichero XML donde cada en cada nodo haya una cadena saludando a esos nombres.

Vamos a dividir este ejercicio en tres fases:

1. Crear la transformación.

2. Construir el esqueleto de la transformación utilizando *Steps* y *Hops*.

3. Configurar los Steps uno a uno indicando la acción deseada.

**Crear la transformación**

1. Crea una nueva transformación: Fichero > Nuevo > Transformación

2. Abre la opción Transformación > Configuración, y pon en el nombre y la descripción “Hola mundo”.

3. Guarda la transformación en la carpeta de prácticas y dale el nombre “hola”. Se creará el fichero “hola.ktr”.

4. Crea un fichero *clientes21.csv* en esa carpeta que contenga la siguiente información:

```
Nombre,apellidos
Maria,Suarez
Juan,Jimenez
Carla,Bush
Beatriz,Ortiz
```

**Construir el esqueleto de la transformación**

Un *Step* es la unidad más pequeña de una transformación. Hay una gran variedad de *Steps* agrupados en categorías como por ejemplo Input y Output. Cada *Step* está diseñado para cumplir una función específica.

Un *Hop* es la representación grafica del flujo de datos entre dos *Steps*. Representa la información saliente de un *Step* y la información entrante del siguiente. Un *Hop* solo tiene un origen y un destino, pero varios Hops pueden surgir del mismo origen y a su vez, varios *Hops* pueden tener el mismo destino.

1. Haz clic el botón de diseño 

y abre la categoría “Entrada”.
2. Arrastra “CSV File Input” a la ventana de diseño.
3. Abre la categoría “Scripting” y arrastra a la ventana de diseño el *Step* “Valor JavaScript Modificado”.

4. Y por último haz lo mismo con el *Step* “Salida XML” de la categoría “Salidas”.

Ahora vamos a unirlos mediante *Hops*:

1. Selecciona el *Step* “CSV file input”.

2. Manteniendo la tecla Shift presionada, traza una recta con el ratón hasta el *Step* “Valor Javascript Modificado”.

3. Crea el último *Hop* para dejar el flujo como en la siguiente imagen:


**Especificar el comportamiento de los *Steps***

Cada *Step* tiene una ventana de configuración y dependiendo de su funcionalidad nos vamos a encontrar con diferentes parámetros.

**- Configuración del CSV file input**

1. Haz doble clic en el *Step* CSV file input.

2. Aparecerá una ventana donde se puede indicar la localización, formato y contenido del fichero de entrada.

3. Ponle al *Step* el nombre: lista de personas. En el campo filename indica el path al fichero. Justo a la derecha del campo hay un botón con el símbolo $ en rojo. Esto sirve para manejar variables dentro del campo de esta forma ${nombre_de_la_variable}. Podemos definir variables de usuario o del sistema. En nuestro caso no va a ser necesario utilizar variables, pero nos podría servir en caso de querer procesar un fichero que vaya modificando su ruta.

4. Haz clic en “Traer campos” y aparecerá en el grid de abajo las columnas nombre y apellidos.

5. Desactiva la opción “Lazy conversion”.

6. Haz clic en “Previsualizar” y podrás ver el contenido del fichero CSV y si está todo correcto haz clic en “Vale” para finalizar el *Step*.

**- Configuración del Valor JavaScript Modificado**

1. Haz doble clic en el *Step*.

2. Esta ventana nos va a permitir escribir una porción de código java. Vamos a utilizar esta ventana para crear el mensaje de bienvenida.

3. Ponle al *Step* el nombre “Hola mundo”.

4. A la izquierda de la ventana podemos ver dos árboles llamados entradas y salidas. En ellos figuran las variables de las que podemos hacer uso en el flujo de datos. Además, tenemos tres apartados más donde podemos hacer uso de funciones para cadenas, números, etc.

5. Copia el código

```
var msg = "Hola " + Nombre + "!";
apellidos = upper(apellidos);
```

6. Abajo en la ventana podemos ver un grid. En este grid tenemos que definir las variables nuevas que creamos, que en nuestro caso es “msg”. Indicamos como nombre “msg” y del tipo String.

7. Haz clic en el botón “Probar script” para comprobar que no haya ningún error.

8. Haz clic en “Vale”.

9. Haz clic derecho en el Step y accede a la opción “mostrar campos entrada” y “mostrar campos salida” para comprobar que la nueva variable msg se ha creado correctamente.

**- Configuración de la salida XML**

1. Haz doble clic en el *Step*.

2. Nómbralo como “Salida hola mundo”.

3. Indica en el campo fichero de salida el nombre holamundo.xml:

```
…/Holamundo.xml
```

4. Ves a la pestaña campos y haz clic en “Obtener campos”. Borra la fila en la que pone “nombre”.

5. Haz clic en “vale” y guarda la transformación.

**- Verificación, Previsualizado y Ejecución**

1. Antes de ejecutar la transformación debemos comprobar que el flujo tiene correcta la sintaxis haciendo clic en el botón de verificación  .

2. Una vez verificada la transformación, podemos previsualizarla pulsando el botón  . Aparecerá una ventana que resumirá la transformación. Pulsar el botón "Quick Launch" y comprobar el resultado.

3. Y finalmente ejecutaremos la transformación pulsando el botón  . Aparecerá abajo una serie de pestañas con la información del resultado. En la pestaña "Logging" podremos comprobar el correcto funcionamiento si aparece al final el siguiente mensaje: "Spoon – The transformation has finished!".

**Ejercicio 2: Hola mundo**

**2.1– Sube a poliformaT el fichero holamundo.ktr.**

**2.2– sube a poliformat el fichero holamundo.xml.**

***Ejemplo: Importación de clientes con Kettle***

Nos vamos a plantear una situación en la que nuestra empresa quiere cambiar su actual ERP por OpenBravoERP. En un principio queremos tan solo dar de alta los clientes en la base de datos de OpenBravo, así que para no tener que crearlos uno a uno vamos a utilizar las transformaciones de Kettle para hacer la portabilidad más cómoda. Partimos de un fichero *clientes3.csv* donde se detalla la siguiente información:

```
Id, Clave, Nombre
2000000, F001, Ferretería Aguilar s.l.
2000001, N001, Neumaticos del Maritimo s.l.
2000002, F002, Ferretería Pallardo s.l.
```

El objetivo es que todas estas filas se inserten correctamente (respetando los sistemas de integridad de la base de datos: campos no nulos, claves ajenas, etc.) en la tabla **c_bpartner** de Openbravo, que es donde se almacenan todos los clientes. Los principales campos de los que vamos a hacer uso de esta tabla son:

| | |
|---|---|
| **c_bpartner_id** character varying(32) NOT NULL | Identificador único del cliente |
| **ad_client_id** character varying(32) NOT NULL | Id. del usuario de OpenBravo |
| **ad_org_id** character varying(32) NOT NULL | Id. de la organización |
| **Createdby** character varying(32) NOT NULL | Id. del usuario que crea el cliente |

| | |
|---|---|
| **Updatedby** character varying(32) NOT NULL | Id. del usuario que actualiza el cliente |
| **Value** character varying(40) NOT NULL | Clave del cliente |
| **Name** character varying(60) NOT NULL | Razón social del cliente |
| **c_bp_group_id** character varying(32) | Id. del grupo al que pertenece el cliente (vendedor, cliente, partner, etc.) |

Ahora solo falta asociar los campos del fichero con los campos de la tabla (proceso conocido como *match*).

- Id → c_bpartner_id
- Clave → value
- Nombre → name

Como podemos observar, hay una serie de campos obligatorios (NOT NULL) que no tienen ninguna asociación y que de alguna forma les tendremos que asignar alguna constante que tenga sentido.

**Configuración en Kettle**

Vamos a seguir los pasos del ejercicio anterior pero con unas cuantas modificaciones.

1. Crea una nueva transformación.
2. Introduce un Step *Entrada->CSV File Input*.
3. Introduce un Step *Transformar->Añadir Constantes*.
4. Introduce un Step *Salida->Salida Tabla*.
5. Guarda la transformación.
6. Crea el flujo de datos desde la entrada hasta la salida, pasando por el step de “añadir constantes”.
7. Configura el *CSV File Input* para que lea del fichero *clientes3.csv*. Al traer los campos, asegúrate de que todos los campos tengan el tipo de dato string. Es necesario que renombres las variables de entrada haciéndolas

coincidir con el nombre del campo de la tabla c_bpartner correspondiente, ya que sino al validar la transformación te dará error (c_bpartner_id, value y name).

***8.*** Haz doble clic en el Step *Añadir Constantes*. Para dar el valor correspondiente a los campos obligatorios de la tabla *c_bpartner* vamos a crear una fila por cada campo, asignándoles a todos el tipo "Integer", de forma que quede la asignación de esta manera:

- ad_client_id = 1000000 (es el id. del administrador de BigBazaar[12]).
- ad_org_id = 0 (es el id. de la organización por defecto).
- createdby = 0 (valor de demostración).
- updatedby = 0 (valor de demostración).
- c_bp_group_id = 1000002 (id. del grupo Clientes).

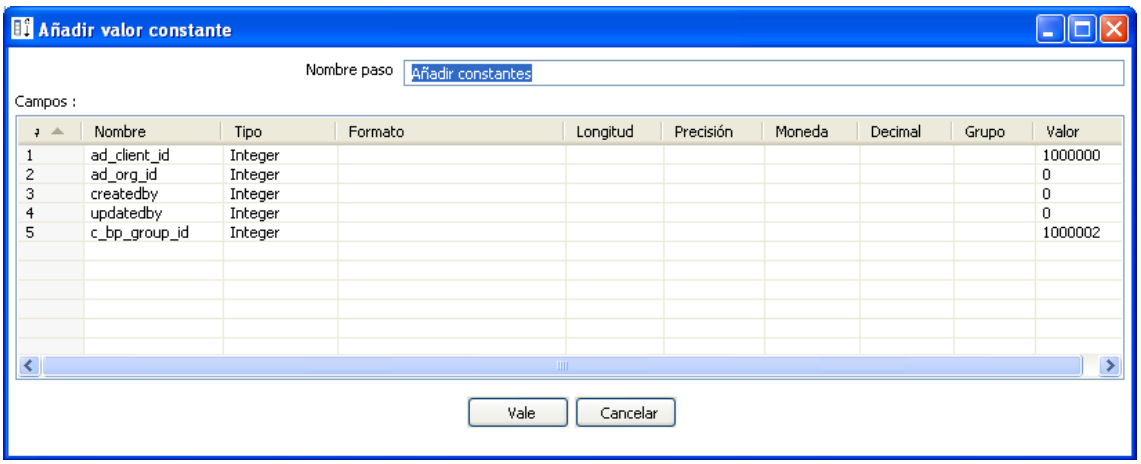



Imagen 7.7: Formulario para añadir constantes al flujo de datos

***9.*** Haz doble clic en el Step *Salida Tabla*. Pulsa el botón Nueva conexión para configurar la conexión a la base de datos. Se abrirá una nueva ventana. Introduce la siguiente información:

[12] BigBazaar es la empresa de demostración de la que dispone OpenBravoERP

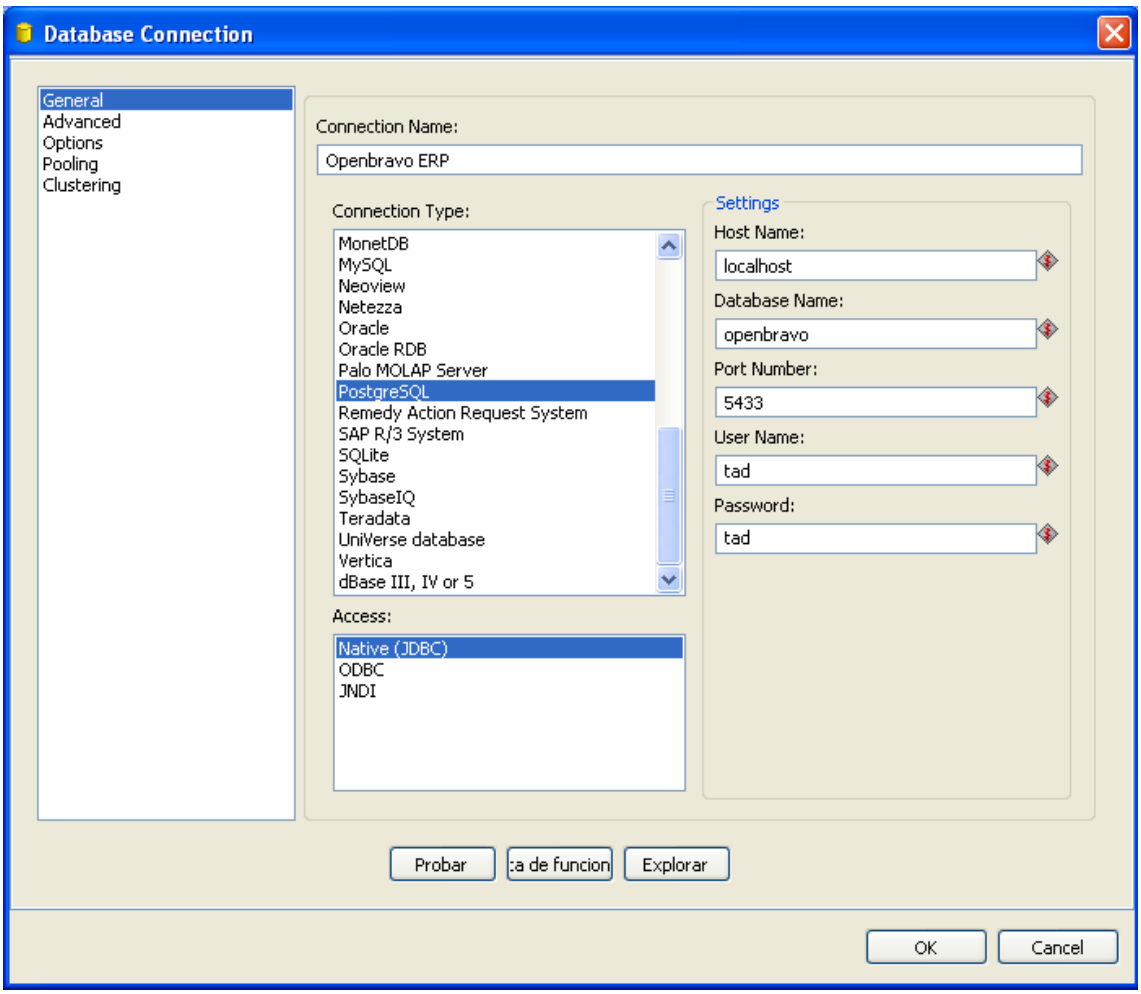



Imagen 7.8: Conexión a la base de datos OpenbravoERP desde Kettle

***10.*** Una vez creada la conexión podrás hacer clic en el botón *Examinar* y ver las tablas que contiene la base de datos. Indica en la tabla destino el valor "c_bpartner".

***11.*** Por último, ya solo necesitaremos asociar los valores de entrada del Step con los campos de la tabla seleccionada. Para ello, haz clic en la pestaña *Database fields*. Pulsa el botón *Enter Field Mapping* y asocia los valores de entrada con los campos (la asociación de los campos de entrada la tienes al principio del ejercicio).

***12.*** Guarda la transformación como ejercicio3.ktr, valídala y si está todo correcto (es posible que salga alguna advertencia, pero se puede omitir), ejecútala.

**Ejercicio 3: Importación con Kettle**

**3.1– Sube a poliformaT el fichero ejercicio3.ktr.**

**3.2– Entra en Openbravo y captura una pantalla donde se pueda ver un listado de los clientes creados. Sube a poliformaT la captura en un fichero .jpg.**

### 7.1.5.- Revisión global

En esta práctica hemos aprendido a instalar correctamente un entorno virtual para poder trabajar con OpenbravoERP. Aunque es una instalación rápida, nos ha permitido trabajar sin la necesidad de tener instaladas todas las herramientas como por ejemplo Apache Tomcat o Apache Ant, necesarias para poder ejecutar el ERP.

Además, hemos aprendido las diferentes formas de implantar OpenbravoERP, cada una con sus ventajas y desventajas, pero siempre facilitándonos en mayor medida las tareas de implantación de un ERP.

## 7.2.- Práctica 2: Ampliando la funcionalidad de OpenbravoERP

En la segunda práctica didáctica vamos a demostrar la potencia del Diccionario de Aplicaciones a la hora de ampliar la funcionalidad del sistema.

### 7.2.1.- Objetivo

El objetivo de esta práctica es conocer las bases de la estructura de OpenbravoERP y aprender a utilizar las herramientas de las que dispone Openbravo ERP para poder ampliar su funcionalidad.

### 7.2.2.- Resumen

Empezaremos la práctica preparando el entorno de trabajo con el que vamos a realizar la práctica. Para ello utilizaremos una máquina virtual con el ERP pre-instalado para poder empezar cuanto antes con el desarrollo. En la segunda parte de esta práctica vamos a analizar una de las principales bazas de OpenbravoERP, la modularidad. Trabajaremos con algunos componentes del Diccionario de la Aplicación, que nos permitirán realizar modificaciones sobre el ERP. Empezaremos creando nuestro propio módulo. Gracias a este módulo podremos agrupar modificaciones sobre el ERP con el objetivo de crear un formulario de entrada donde se gestionará los salarios de los empleados.

### 7.2.3.- Introducción

Una de las mejores bazas de OpenbravoERP es el sistema modular que permite añadir y quitar funcionalidad según convenga. Este sistema, junto con la liberación del código fuente del ERP, ha permitido que una gran comunidad de usuarios desarrollen sus propios módulos y los puedan compartir con el resto de usuarios. Gracias a esto hoy en día podemos instalarnos el ERP básico y añadirle una gran variedad de funcionalidades gracias a los módulos disponibles vía Web.

El objeto de esta práctica va a ser desarrollar nuestro propio módulo. Vamos a añadir una nueva opción en el menú que permita gestionar los salarios de los empleados.

Un aspecto importante es que no nos va a hacer falta desarrollar una sola línea de código. Esto es gracias al Diccionario de la Aplicación, un módulo principal del ERP que gestiona la estructura asociativa entre componentes como: tablas, ventanas, pestañas, etc., permitiendo extender la funcionalidad aunque con ciertos límites.

En resumen, vamos a añadir nueva funcionalidad a Openbravo de una forma rápida y sencilla.

### 7.2.4.- Desarrollo

#### *7.2.4.1.- El entorno de trabajo*

A continuación vamos a describir los pasos necesarios para poner en funcionamiento el entorno de trabajo necesario para poder realizar la práctica.

**Instalación de OpenbravoERP**

Primero de todo vamos a ejecutar en nuestro sistema el software OpenBravo ERP para poder empezar a trabajar. Puesto que la instalación es un proceso que puede llevar unas horas y no es el objetivo de esta práctica vamos a utilizar una máquina virtual en la que viene ya correctamente instalado en el sistema operativo Linux[13]. Para poner en marcha el entorno de trabajo:

1- Ves a la siguiente dirección https://www.vmware.com/products/player/ y descárgate el cliente VMWare Player.
2- Instálatelo haciendo doble clic.

3- Ejecuta VMWare. Al abrirse el programa, haz clic en el botón "Open" y selecciona la máquina virtual[14] "c:\OpenbravoERP\OpenbravoERP-2.50MP6-x86.vmx".

4- Una vez arrancado el SO, se nos pedirá en modo consola el usuario y contraseña de acceso al sistema, a lo que introduciremos "openbravo" en ambos. Para poder salir de la consola y poder controlar el cursor presionar: ctrl. + Alt

Una de las mejores bazas de OpenBravo es que está pensado para que se pueda acceder desde cualquier ordenador conectado a Internet, por lo que se puede ejecutar desde un navegador Web.

Una vez preparada la máquina virtual tan solo tendremos que abrir un navegador desde nuestro SO y acceder a la dirección IP[15] que se indica en la consola virtual de Openbravo (por ejemplo http://192.168.2.101). Aparecerá una página Web con un formulario donde nos pedirá un usuario y su clave, a lo que introduciremos "Openbravo" para el usuario y "openbravo" para la clave.

**Conexión SSH a la Base de Datos**

---

[13] Vamos a utilizar lo que se denomina la aplicación virtual de OpenbravoERP. Podéis encontrar toda la documentación en http://wiki.openbravo.com/wiki/Virtual_appliances#Community_Appliance

[14] La máquina virtual de OpenbravoERP la podéis encontrar en http://sourceforge.net/projects/openbravo/files/01-openbravo-appliances/2.50MP8/

[15] De ahora en adelante, para facilitar la lectura de la práctica vamos a simular que la IP que nos proporciona la consola virtual de OpenBravo es 192.168.2.101 (lo más probable es que el VmWare Player os proporcione otra)

Por razones de seguridad, el SO ejecuta un cortafuegos que bloquea cualquier acceso externo excepto los puertos 22 (SSH), 80 (HTTP) y 443 (HTTPS). Así que para conectarnos a la base de datos con una herramienta externa es necesario antes establecer un túnel SSH.

Para crear ese túnel vamos a utilizar el cliente SSH PuTTY. Empezaremos ejecutando el archivo putty.exe y seguiremos estos pasos [ver imágenes a continuación]:

1. Seleccionar la categoría **Session**.
2. En la caja **Host name**, poner la dirección IP de Openbravo: 192.168.2.101.
3. Seleccionar la categoría **Connection > SSH > Tunnels**.
4. En el campo **Source port** poner 5433.
5. En el campo **Destination** poner localhost:5432.
6. Hacer clic en **Add**.
7. Hacer clic en **Open** para abrir el túnel SSH. Es posible que abra una alerta de seguridad de Putty. Hacer clic en **Si**.
8. Poner el usuario “openbravo” y contraseña “openbravo”.

Para que las herramientas que acceden a la base de datos este túnel debe estar **Abierto** durante toda la sesión.

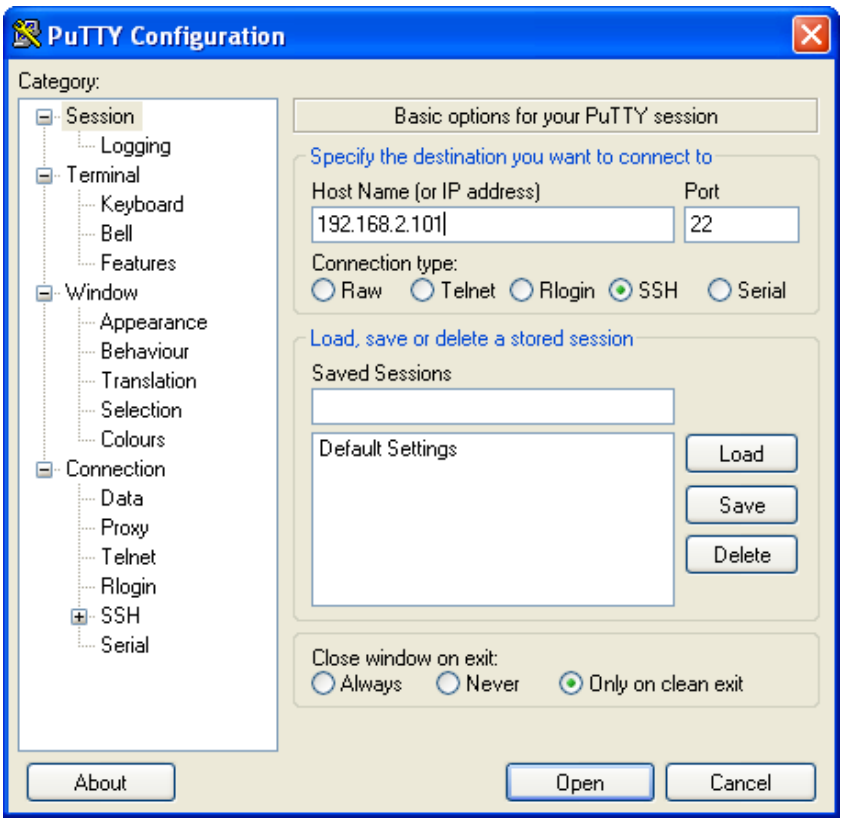



Imagen 7.9: Paso 1 configuración PuTTY

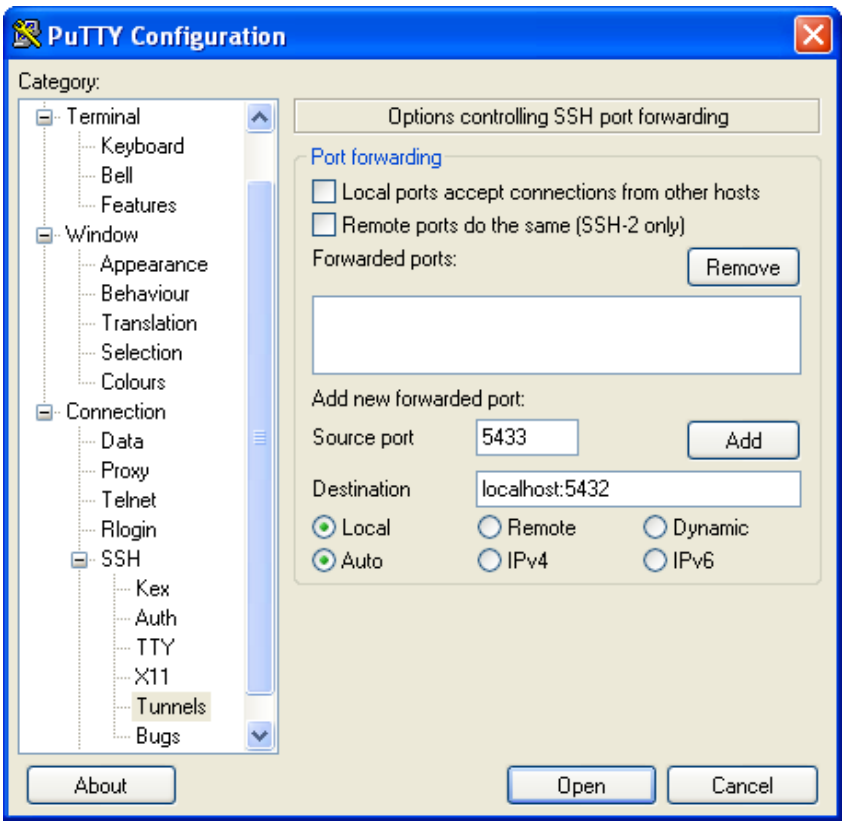



Imagen 7.10: Paso 2 configuración PuTTY

**Configuración de pgAdmin III**

Más adelante se nos va a presentar la necesidad de acceder a la base de datos de OpenBravo para consultar sus tablas. Para poder manipular la base de datos es necesario conectarnos al gestor PostgreSQL con el cliente pgAdmin III. A continuación vamos a configurar la conexión con este programa:

4. Ejecutar Inicio > Programas > pgAdmin III

5. Hacer clic en el botón  para añadir una nueva conexión.

6. Rellenar los siguientes datos:

    a. **Nombre:** OpenbravoERP

    b. **Servidor:** localhost

    c. **Port:** 5433

    d. **Username:** tad

    e. **Password:** tad

Una vez conectados desplegaremos la pestaña openbravo para poder ver todas las tablas (doble clic para conectarse al servidor).

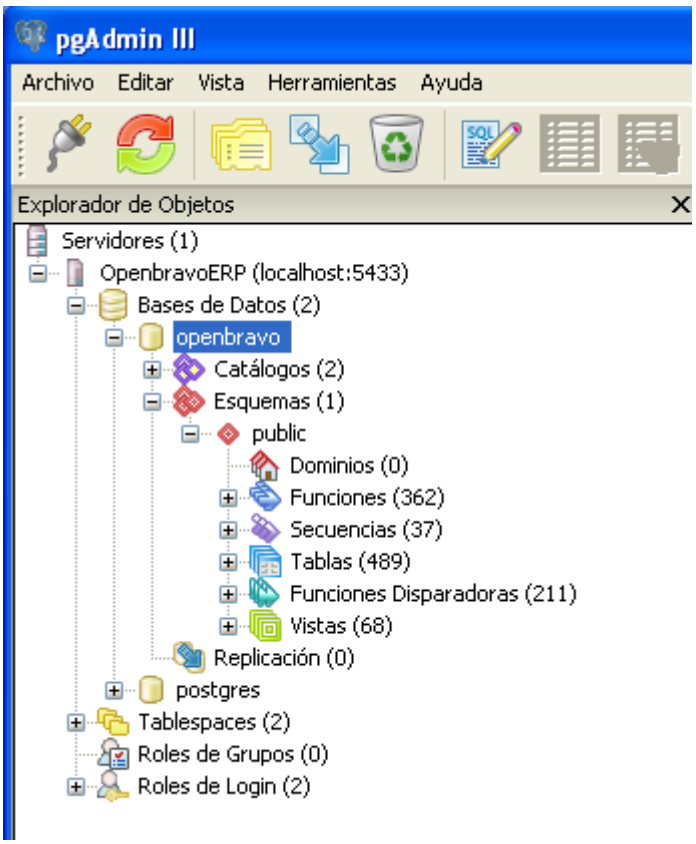



Imagen 7.11: Propiedades de la base de datos

Hasta aquí llega la configuración del entorno de trabajo. Una vez puestas en marcha todas las aplicaciones vamos a empezar con la práctica.

**Guardar el estado de la máquina virtual**

Cuando el PC del laboratorio se reinicia vuelve a un estado “original” y borra todas las modificaciones que haya habido en la anterior sesión. Para poder seguir el trabajo que hagáis en esta práctica para la siguiente sesión será necesario que guardéis el estado de la máquina virtual de OpenBravoERP en una memoria USB. Esto es, copiar el fichero *OpenbravoERP-2.50MP6-x86.vmdk* de 2GB aproximadamente, en un USB y en la próxima práctica restaurar el fichero del PC con el que tengáis en el USB.

### *7.2.4.2.- Proceso de trabajo*

Para implementar el nuevo módulo de salarios vamos a realizar los siguientes pasos:

1. Crear un módulo,
2. Crear la tabla SQL necesaria para almacenar los salarios en la base de datos,
3. Registrar la tabla en la aplicación mediante el Diccionario de la Aplicación,
4. Crear una nueva ventana donde colocar el formulario,

5. Crear un nuevo ítem en el menú para poder acceder a la ventana y

6. Recompilar OpenbravoERP para poder visualizar los cambios.

#### *7.2.4.3.- Creación del módulo*

Un módulo es un contenedor que permite agrupar artefactos que componen la *extensión de módulo*: artefactos del Diccionario de la Aplicación (tablas, ventanas, campos, etc.), recursos software y *scripts*. Definiendo las extensiones como módulos, podremos extraerlas, empaquetarlas y utilizarlas en diferentes entornos. Todas las extensiones y configuraciones de una instancia de OpenbravoERP deben estar definidas obligatoriamente en el contexto de un módulo.

Openbravo soporta tres tipos de extensión de módulos:

- Módulos: es el contenedor básico que define una funcionalidad añadida. Un módulo puede añadir artefactos a una instalación de OpenbravoERP pero no puede modificar artefactos que sean propiedad de otros módulos. Esta restricción permite simplificar la administración de dependencias. Puedes instalar todos los módulos que necesites en Openbravo.

- Paquetes: es un grupo de módulos. Puedes instalar todos los módulos que necesites.

- Plantilla: es un módulo con un tipo específico de artefacto llamado *script de configuración* que permite modificar atributos del artefacto que pertenecen a otros módulos. Puedes usar una plantilla industrial para empaquetar y redistribuir una configuración específica del sistema. En un sistema, solo puede haber una plantilla definida.

En nuestro caso solo vamos a utilizar un módulo básico.

**Pasos a seguir**

Para crear el módulo accede a Openbravo mediante un navegador Web indicando en la URL la IP que proporciona la máquina virtual. Accede con el usuario “Openbravo” y clave “openbravo”. Con este usuario podemos acceder a una serie de roles haciendo clic en el icono  . Los roles que vamos a utilizar son:

- *System administrador*: usuario administrador del sistema. Uno de sus privilegios es poder acceder y realizar modificaciones sobre el Diccionario de la Aplicación.

- *Big Bazaar admin*: usuario administrador de la empresa Big Bazaar. Desde este usuario vamos a poder utilizar la ventana de gestión de salarios.

Para poder ver el Diccionario de la Aplicación y crear el módulo accede como *System administrador*.

1. Entra en *Application Dictionary* > *Module* y crea un nuevo registro.

2. Configura el módulo de la siguiente manera:

    - Marca el módulo como activo.
    - Nómbralo: “ModPrac01”.
    - Asígnale el paquete java “org.openbravo.modprac01”. Esto será el paquete donde introduciremos si es necesario las clases Java.
    - Indica el tipo “Module”.
    - Marca “In Development” y “Default” para indicar que el módulo está en desarrollo y que todos los artefactos que creemos a partir de ahora se asignen automáticamente a este módulo.
    - Marca “Translation required”. Esto indica que el módulo tendrá campos de texto visibles.
    - Haz clic en el botón guardar.

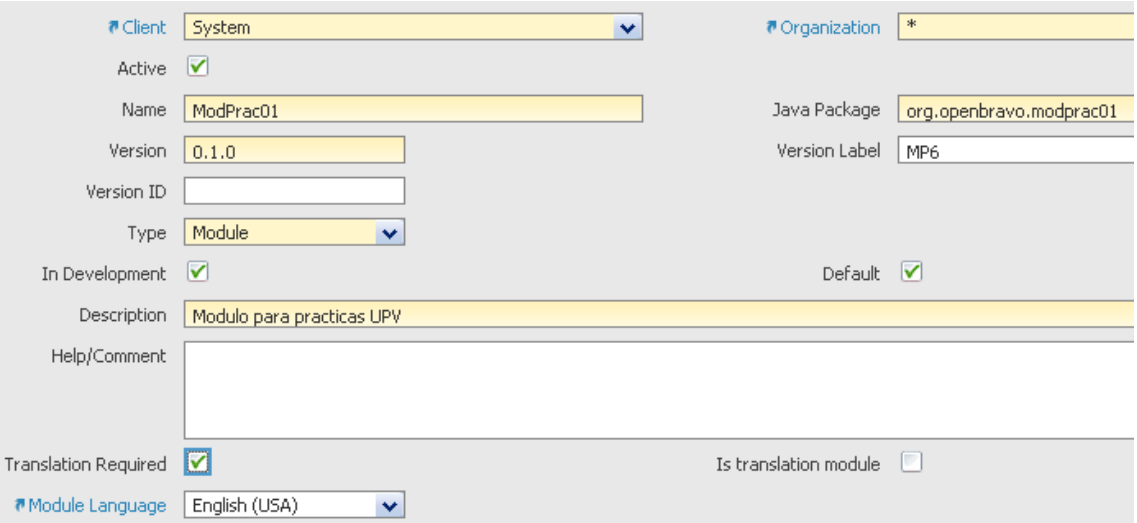



Imagen 7.12: Creación del módulo

3. Haz clic en la pestaña *Dependency*. Crea un nuevo registro, por defecto aparecerá en el desplegable la dependencia *core*. Guarda el registro. Esto nos servirá para poder utilizar en nuestro módulo referencias al módulo *core* que contiene todo el ERP.

4. Haz clic en la pestaña *Data package*. Crea un nuevo registro indicando:

    - Introduce el nombre: “ModPrac01”.
    - Introduce la descripción: “Módulo de prácticas”.

- Introduce el Java Package: “org.openbravo.modprac01.data”. Este paquete contiene las clases relacionadas con la manipulación de datos del módulo.

- Haz clic en el botón guardar.

5. Haz clic en la pestaña *DB prefix*. Crea un nuevo registro indicando el prefijo “MODPR01” y guárdalo. El prefijo es una palabra que identifica nuestro módulo en la base de datos. Hay una serie de prefijos que openbravo tiene reservados para la identificación de sus tablas, por ejemplo AD (Application Dictionary), C (Core), I (Import). Los prefijos deben ser únicos.

Con esto ya hemos creado el módulo sobre el que realizaremos todos los artefactos. Como le hemos asignado el valor “Default” no será necesario preocuparnos por asignar los artefactos a este módulo puesto que se hará automáticamente.

### *7.2.4.4.- Creación de la tabla en BBDD*

El módulo de salarios necesita almacenar la información en base de datos para no perder los cambios realizados sobre los salarios. Para este fin vamos a definir una tabla llamada MODPR01_SALARIOS.

1. Accede a la base de datos Openbravo con el cliente de PostgreSQL pgAdmi3 como indicamos en la anterior práctica.

2. Haz clic en “OpenbravoERP” para conectarte a la base de datos y desplegar sus tablas.

3. Haz clic en el botón  e introduce la siguiente instrucción en la ventana emergente:

```
CREATE TABLE MODPR01_SALARIOS
(
 MODPR01_SALARIOS_ID  CHARACTER VARYING(32)                    NOT NULL,
 AD_CLIENT_ID  CHARACTER VARYING(32)                  NOT NULL,
 AD_ORG_ID     CHARACTER VARYING(32)                  NOT NULL,
 ISACTIVE      CHARACTER(1)              NOT NULL      DEFAULT 'Y',
 CREATED       TIMESTAMP WITHOUT TIME ZONE     NOT NULL            DEFAULT now(),
 CREATEDBY     CHARACTER VARYING(32)                  NOT NULL,
 UPDATED       TIMESTAMP WITHOUT TIME ZONE     NOT NULL            DEFAULT now(),
 UPDATEDBY     CHARACTER VARYING(32)             NOT NULL,
 C_BPARTNER_ID CHARACTER VARYING(32)          NOT NULL,
 AMOUNT  NUMERIC                                NOT NULL,
 C_CURRENCY_ID VARCHAR(32)                    NOT NULL,
 VALIDFROM     TIMESTAMP WITHOUT TIME ZONE    NOT NULL,
  CONSTRAINT       MODPR01SALARIOS_ISACTIVE_CHECK       CHECK       (isactive       =       ANY
(ARRAY['Y'::bpchar, 'N'::bpchar])),
  CONSTRAINT MODPR01_SALARIOS_KEY PRIMARY KEY (MODPR01_SALARIOS_ID),
  CONSTRAINT AD_ORG_CUS_PROJECT FOREIGN KEY (AD_ORG_ID)
   REFERENCES AD_ORG (AD_ORG_ID),
  CONSTRAINT AD_CLIENT_CUS_PROJECT FOREIGN KEY (AD_CLIENT_ID)
   REFERENCES AD_CLIENT (AD_CLIENT_ID),
  CONSTRAINT C_BPARTNER_HR_SALARY FOREIGN KEY (C_BPARTNER_ID)
   REFERENCES C_BPARTNER (C_BPARTNER_ID),
```

```
    CONSTRAINT C_BPARTNER_C_CURRENCY FOREIGN KEY (C_CURRENCY_ID)
     REFERENCES C_CURRENCY (C_CURRENCY_ID)
);
```

4. Haz clic en el botón “play” para ejecutar la instrucción. Comprueba que la consulta se haya ejecutado con éxito.

Los campos AD_CLIENT_ID, AD_ORG_ID, ISACTIVE, CREATED, CREATED_BY, UPDATED y UPDATED_BY son campos obligatorios en todas las tablas de Openbravo. Esto facilita su mantenimiento.

La clave principal de la tabla debe nombrarse siempre como <*Nombre_de_la_tabla + “ID”*>.

#### *7.2.4.5.- Registro de la tabla en Openbravo*

Para que Openbravo pueda empezar a trabajar con la tabla que hemos creado, debemos primero registrarla correctamente mediante el Diccionario de la Aplicación.

1. Accede a *Application Dictionary* > *Tables and Columns* y crea un nuevo registro.

2. Configura la tabla de la siguiente manera:

    a. Introduce el nombre: “MODPR01_SALARIOS”.

    b. Introduce el *DB Table Name*: “MODPR01_SALARIOS”. Hace referencia a la tabla que hemos creado en la base de datos.

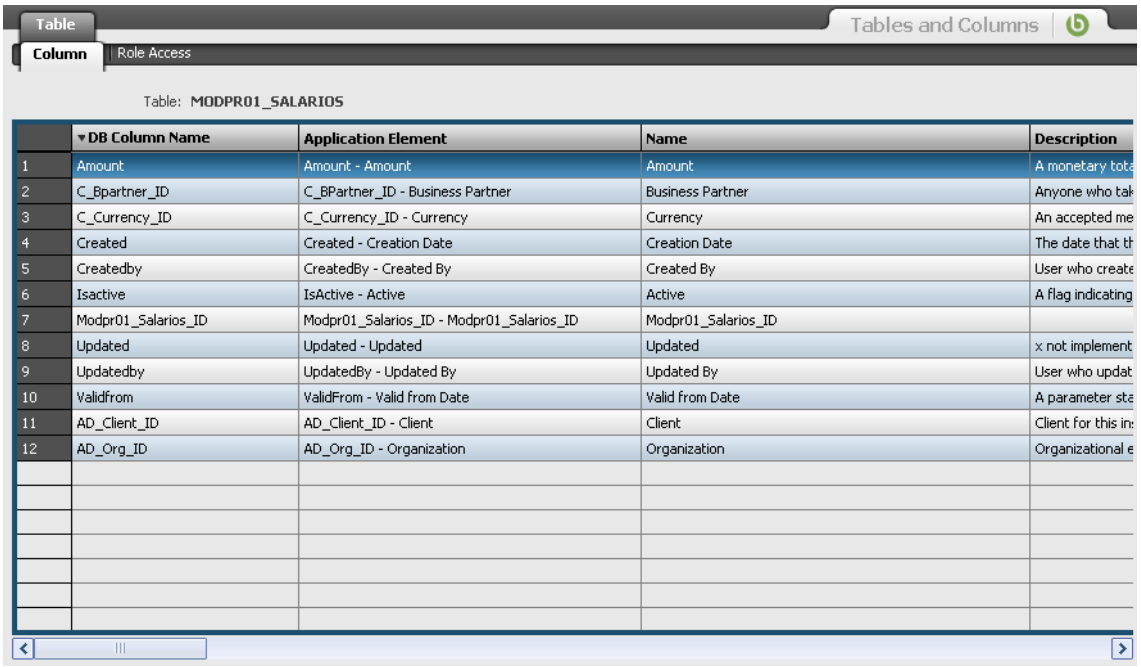
    c. Introduce la *Java Class Name*: “MODPR01Salarios”. Esta será la clase que creará Openbravo para acceder a la información de la tabla MODPR01_SALARIOS.

    d. Selecciona el *Data Acces Level*: “Client/Organization”. Aquí indicamos que nuestro módulo será visible para las empresas, pero no para el administrador.

3. Guarda el registro y haz clic en el botón “Create Columns from DB”. Aparecerá un mensaje indicando que se han añadido 12 columnas nuevas. Haz clic en la pestaña *Column* y podrás ver todos los campos que ha importado Openbravo desde la base de datos. Cada columna tiene una Referencia (que especifica el tipo de dato) dependiendo del tipo de dato que tenga en base de datos.

4. Haz clic en el proceso *Application Dictionary* > *Synchronize Terminology*. Se abrirá una ventana emergente, ha clic en OK. Este proceso comprueba todos los cambios que se hayan realizado en la interfaz del Diccionario de la Aplicación. En caso de existir algún cambio comprueba que tenga asignado algún *Element* (elementos que se visualizan en la interfaz de Openbravo, como por ejemplo: un campo, un desplegable, etc.) y si no lo tiene crea uno automáticamente.

5. Vuelve a entrar en la pestaña *Column* y comprueba que todos los campos tienen un valor en el atributo *Application Element*.




| | DB Column Name | Application Element | Name | Description |
|---|---|---|---|---|
| 1 | Amount | Amount - Amount | Amount | A monetary tota |
| 2 | C_Bpartner_ID | C_BPartner_ID - Business Partner | Business Partner | Anyone who tak |
| 3 | C_Currency_ID | C_Currency_ID - Currency | Currency | An accepted me |
| 4 | Created | Created - Creation Date | Creation Date | The date that th |
| 5 | Createdby | CreatedBy - Created By | Created By | User who create |
| 6 | Isactive | IsActive - Active | Active | A flag indicating |
| 7 | Modpr01_Salarios_ID | Modpr01_Salarios_ID - Modpr01_Salarios_ID | Modpr01_Salarios_ID | |
| 8 | Updated | Updated - Updated | Updated | x not implement |
| 9 | Updatedby | UpdatedBy - Updated By | Updated By | User who updat |
| 10 | Validfrom | ValidFrom - Valid from Date | Valid from Date | A parameter sta |
| 11 | AD_Client_ID | AD_Client_ID - Client | Client | Client for this in |
| 12 | AD_Org_ID | AD_Org_ID - Organization | Organization | Organizational e |

Imagen 7.13: Creación de una Tabla

6. Haz doble clic en el campo *Amount*:

    a. Asígnale al 10 al atributo Lenght.

    b. Selecciona el tipo Amount en el deplegable Reference. Esto es para saber que este campo va a almacenar una cantidad monetaria.

    c. Márcalo como Identifier.

    d. Guarda los cambios.

7. Haz doble clic en el campo Validfrom:

    a. Selecciona el tipo Date en el desplegable Reference.

    b. Márcalo como Identifier.

    c. Guarda los cambios.

Con esto ya tenemos la tabla correctamente configurada en el ERP. Ahora podemos utilizarla en nuestro módulo.

### *7.2.4.6.- Creación de la Ventana*

Vamos a pasar a la configuración de la ventana que visualizará la información que contiene la tabla MODPR01_SALARIOS.

1. Accede a Application *Dictionary* > *Windows, Tabs and Fields* y crea un nuevo registro.

2. Configura la ventana de la siguiente manera:

    a. Introduce el nombre: “Salarios”.

    b. Introduce una descripción y un comentario de ayuda.

    c. Selecciona el *Window Type*: “Maintain”. Esto se usa para ventanas con pocas entradas.

    d. Guarda el registro.

3. Haz clic en la pestaña *Tab* y crea un nuevo registro.

4. Configura la pestaña de la siguiente manera:

    a. Introduce el nombre: “Salarios”.

    b. Introduce una descripción y un comentario de ayuda.

    c. Selecciona la tabla: MODPR01_SALARIOS.

    d. Introduce el valor 0 en el *Tab level*. Esto sirve para saber el orden de las pestañas en el caso de que haya más de una. Si tiene el valor 0, esta pestaña será la primera en visualizarse cuando un usuario entre en la ventana Salarios.

    e. Guarda el registro y haz clic en el botón *Create Fields*. Confirma la ventana emergente.

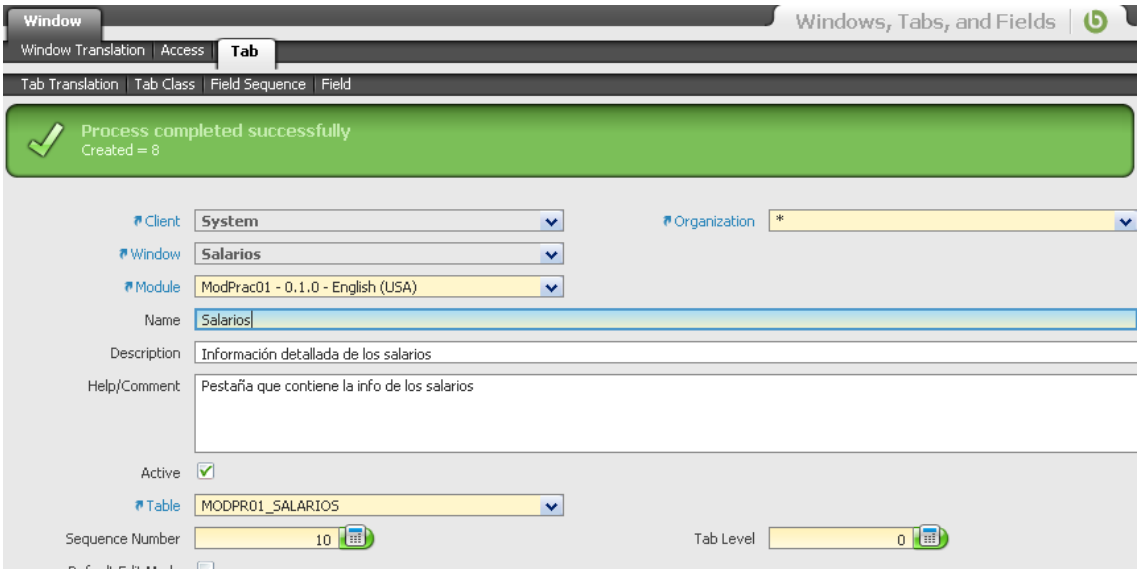



Imagen 7.15: Proceso de Creación de campos

5. Haz clic en la subpestaña *Field* y comprueba que aparezcan todos los campos de la tabla MODPR01_SALARIOS.

6. Haz clic en la subpestaña *Field Sequence*. Aquí podremos elegir los campos que se van a visualizar (tabla derecha) y su orden. Coloca los campos en el siguiente orden: Organization > Client > Business Partner > Active > Amount > Currency > Valid From Date y guarda los cambios.

7. Accede a Application Dictionary > Tables and Columns, busca la tabla MODPR01_SALARIOS y haz doble clic. Busca el desplegable *Window* y selecciona la ventana “Salarios”. Guarda el registro. Con esto crearemos un enlace (link azul) a los elementos de la tabla.

Ya hemos terminado de configurar la ventana y su pestaña principal. Ahora solo falta el último paso, añadir un el elemento en el menú.

#### *7.2.4.7.- Configuración del Menú*

El ítem en el menú es necesario para permitir el acceso a los usuarios a la ventana que hemos desarrollado.

1. Accede a General *Setup* > *Application* > *Menu* y crea un nuevo registro.

2. Configura el menú de la siguiente manera:

    a. Introduce el nombre: “Salarios”.

    b. Introduce una descripción.

    c. Despliega *Action* y selecciona: “Window”. Asociamos el ítem del menú a una ventana. Aparecerá una nuevo desplegable.

    d. Despliega *Window* y selecciona: “Salarios”.

    e. Guarda el registro.

3. Haz clic en el icono  . Aparecerá una ventana nueva donde se visualiza un árbol que contiene todos los ítems del menú de OpenbravoERP. Nuestro objetivo es que aparezca nuestra ventana dentro de las opciones relacionadas con los empleados.

4. Al final de la ventana aparece el ítem del árbol “Salarios”. Arrastra el ítem hasta *Menu* > *Master Data Management* > *Business Partner Setup*. El cambio se guarda automáticamente.

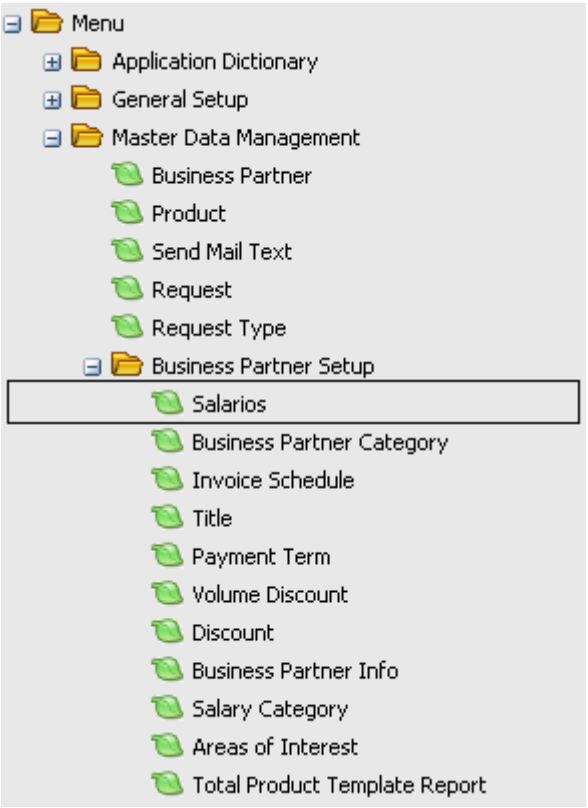



Imagen 7.16: Ventana del árbol de menu

### *7.2.4.9.- Recompilación de Openbravo*

Llegados a este punto, ya hemos creado todos los artefactos del Diccionario de la Aplicación necesarios para el módulo de salarios. Ahora es necesario recompilar OpenbravoERP para que se generen todas las clases java y los ficheros XML de configuración asociados al módulo.

Puesto que compilar todo el ERP puede ser una tarea larga, Openbravo está pensado para que este tipo de tareas podamos compilarlas de una forma "inteligente", es decir, compilando solo los ficheros necesarios.

1. Accede a la consola de comandos de la máquina virtual (vmWare Player). Si todavía no has introducido el usuario, escribe "openbravo" y la clave "openbravo".

2. Escribe el comando "cd /opt/AppsOpenbravo/". Aquí reside todo el proyecto de Openbravo.

3. Escribe el comando "ant compile.development –Dtab='Salarios' ". Esta instrucción invoca al fichero build.xml del directorio actual y ejecuta el procedimiento compile.development, que se encarga de compilar el código fuente deseado.

NOTA: el teclado de la consola virtual no es el Español. Utiliza la siguiente referencia:
/ ⇔ Tecla -
- ⇔ Tecla ‘
= ⇔ Tecla ¡
‘ ⇔ Tecla ¨

4. Si todo está correcto, al finalizar la compilación saldrá el siguiente mensaje: “BUILD SUCCESSFUL”.

```
compile:

copy.files:
     [copy] Copying 1468 files to /var/lib/tomcat/webapps/openbravo
     [copy] Copying 554 files to /var/lib/tomcat/webapps/openbravo/WEB-INF/classes

compile.development:

BUILD SUCCESSFUL
Total time: 2 minutes 22 seconds
[openbravo@localhost AppsOpenbravo]$
```

Imagen 7.17: Resultado de la compilación

Una vez compilado el código fuente vamos a reiniciar la máquina virtual de vmWare para que los cambios surjan efecto.

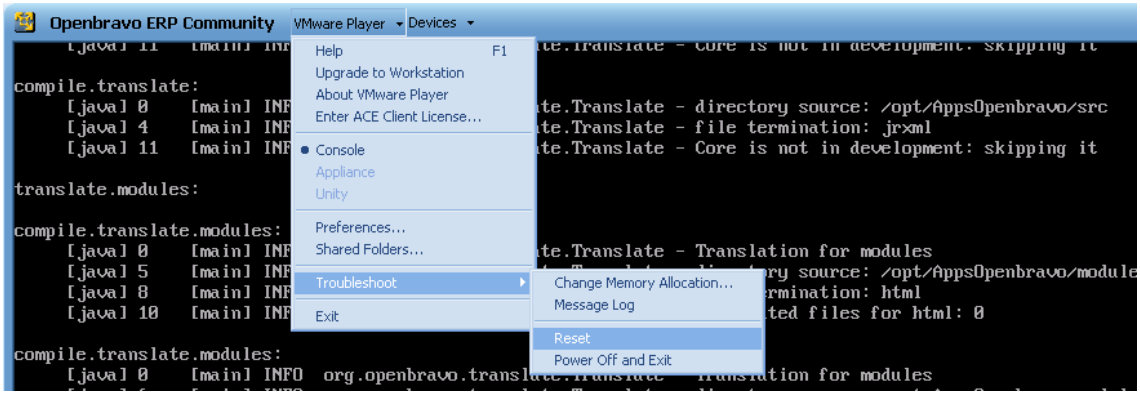



Imagen 7.18: Reset de la máquina virtual

**Ejercicio: Módulo de salarios**

**1- Accede al ERP con el rol “Big Bazaar admin” y busca la ventana de Salarios. Haz una captura de pantalla del menú con la opción “Salarios” visible.**

**2- Crea un nuevo registro inventándote los datos del salario. Haz una captura de pantalla del formulario de creación del salario.**

**Sube a poliformaT las capturas de pantalla**

### 7.2.5.- Revisión global

En esta práctica hemos indagado en la estructura modular de Openbravo y sus ventajas. Los módulos permiten a esta herramienta amoldarse a las necesidades de los usuarios y configurar de una forma muy sencilla que quiere y lo que no. Además, hemos aprendido a realizar un sencillo módulo que se integra a la perfección con el ERP, sin la necesidad de tocar una línea de código fuente.

# 8.- Conclusiones

## 8.1.- Acercamiento a los objetivos planteados

En este proyecto se ha conseguido completar satisfactoriamente los objetivos planteados al inicio. Se ha conseguido analizar la arquitectura con la que se sostiene el sistema OpenbravoERP para, posteriormente, poder realizar ampliaciones en su funcionalidad de una forma estable, rápida y eficaz.

El objetivo didáctico se ha abarcado satisfactoriamente gracias al último capítulo, que pone en práctica gran parte de lo visto en la memoria. Además, uno de los boletines de prácticas ha sido probado con éxito en un entorno real.

## 8.2.- Conclusión a nivel de funcionalidad

En esta memoria se han asentado las bases de la arquitectura de OpenbravoERP. Esto significa que se ha analizado gran parte de lo necesario para entender cómo está organizado el sistema por dentro.

A nivel funcional la memoria va a servir para documentarse en muchos aspectos, pero no para todos los posibles. Un ERP abarca muchas tecnologías y conocer todas sus funcionalidades puede llevar mucho tiempo. Así, en caso de que un usuario quiera realizar un desarrollo en OpenbravoERP, esta memoria le servirá como base para documentarse, pero deberá emplear los conocimientos adquiridos y documentarse más para poder alcanzar cualquier desarrollo que se proponga.

## 8.3.- Conclusión del trabajo propio

A lo largo de todo este proyecto he adquirido muchos conocimientos sobre las tecnologías que utiliza OpenbravoERP, como por ejemplo Ant, los servicios REST o Hibernate. He aprendido que un sistema grande como OpenbravoERP requiere el soporte de muchas tecnologías y sobretodo, herramientas que ayuden a la simbiosis entre ellas.

Me he quedado sorprendido por la cantidad de cuidados que Openbravo dedica a sus desarrolladores facilitándoles la tarea de implementar mediante los módulos, el Diccionario de Aplicaciones, o con las mismas herramientas de desarrollo comunitario, como por ejemplo Selenium.

Aún así, una de las experiencias más duras sufridas durante este proyecto ha sido la búsqueda de documentación relacionada con el ERP. Openbravo ha dispuesto a sus usuarios una página Web al estilo Wikipedia para recoger toda la documentación. Este sistema es bueno, pero la información que almacena no está actualizada. OpenbravoERP avanza tan rápido de versión que lo más probable es que la documentación de la Wikipedia se quede anticuada rápidamente.

En conclusión, ha sido una experiencia muy favorable personalmente el estudiar un ERP de código fuente abierto.

## 8.4.- Líneas de trabajo futuro

Existen muchas posibilidades para extender este proyecto. Para empezar, tan sólo se han realizado dos boletines de prácticas. Estos solo ponen en práctica dos pequeñas partes del Diccionario de Aplicaciones. Se podría hacer, por ejemplo, un boletín para diseñar informes con JasperReports y registrarlo en el sistema para que se puedan visualizar desde la aplicación.

Otra línea de trabajo futuro sería actualizar la documentación ya que OpenbravoERP libera periódicamente nuevas actualizaciones de su sistema.

# Bibliografía


*Blog de Ricardo Galli*, Ricardo Galli, 2007. Disponible en http://mnm.uib.es/gallir/posts/2007/08/15/1146/ (06/02/2010)

*ERP*, Deister, 2009. Disponible en http://www.deister.es/es/products/e-erp/icon/pa.html

James Senn, *Análisis y diseño de Sistemas de Información*, Mc Graw Hill, 1990

*OpenbravoERP*, Enrique Dans, 2006. Disponible en http://www.enriquedans.com/2006/05/openbravo-un-erp-de-codigo-abierto.html (02/02/2010)

*Planificación de Recursos Empresariales*, Jose Luís Figueroa. Disponible en http://www.gestiopolis.com/recursos4/docs/ger/planerp.htm (10/10/2009)

*Sitio Web de Compiere*, Compiere, 1999. Disponible en http://www.compiere.com/ (04/02/2010)

*Sitio Web de FacturaLux*, FacturaLux, 2008. Disponible en http://www.facturalux.org/ (04/02/2010)

*Sitio Web de Fisterra*, Fisterra, 2003. Disponible en http://community.igalia.com/twiki/bin/view/Fisterra (04/02/2010)

*Sitio Web de Gartner*, Gartner co. Disponible en http://www.gartner.com (11/10/2009)

*Sitio Web de OpenBlueLab*, OpenBlueLab 2007. Disponible en http://www.openbluelab.org (05/02/2010)

*Sitio Web de Openbravo*, Openbravo, 2001. Disponible en http://www.openbravo.com/es/ (03/02/2010)

*Sitio Web de OpenERP*, OpenERP, 2005. Disponible en http://www.openerp.com/ (04/02/2010)

*Sitio Web de OpenXpertia*, OpenXpertia, 2007. Disponible en http://www.openxpertya.org/ (04/02/2010)

*Sitio Web de Project Open*, Project Open, 2009. Disponible en http://www.project-open.org/ (05/02/2010)

*Sitio Web de Sage*, Sage Division, 2009. Disponible en http://www.sageerpx3.com/sp/page/home (02/02/2010)

*Sitio Web de SAP*, SAP, 2005. Disponible en http://www.sap.com/spain/solutions/business-suite/erp/index.epx (02/02/2010)

*Sitio Web de Solmicro*, Somicro, 2010. Disponible en http://www.solmicro.com/ (03/02/2010)

**Enlaces utilizados para el capítulo 3**

*Configuración del Stack de desarrollo*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Development_Stack_Setup

*Instalación del entorno de Openbravo*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Openbravo_environment_installation

*Manual de instalación de OpenbravoERP*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Openbravo_ERP_Installation/Custom_Installation

*Manual de Usuario de Apache Ant*, Apache, 2004. Disponible en http://ant.apache.org/manual/index.html (10/10/2009)

**Enlaces utilizados para el capítulo 4**

*Arquitectura dirigida por modelos*, Wiki Openbravo. Disponible en http://es.wikipedia.org/wiki/Model_Driven_Architecture

*Conceptos básicos en el desarrollo de Openbravo*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Developers_Guide/Openbravo_Main_Development_Concepts

**Enlaces utilizados para el capítulo 5**

*Capa de Acceso a Datos*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Developers_Guide/Concepts/Data_Access_Layer

*Diagrama entidad – relación*, SourceForge. Disponible en http://sourceforge.net/projects/openbravo/files/07-openbravo-other/

*Modelo de base de datos, Wiki Openbravo*. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Developers_Guide/Database_Model

*Modelo entidad*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Developers_Guide/Reference/Entity_Model

**Enlaces utilizados para el capítulo 6**

*Guía para los desarrolladores*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.50/Developers_Guide/

*Manual de usuario de Openbravo*, Wiki Openbravo. Dispoble en http://wiki.openbravo.com/wiki/ERP/2.50/User_Manual/

*Manual para los desarrolladores*, Wiki Openbravo. Disponible en http://wiki.openbravo.com/wiki/ERP/2.40/Developers_Manual