FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

"RE-DISEÑO DE LA ARQUITECTURA DEL ERP DE FUENTE ABIERTA ADEMPIERE, UTILIZANDO EL MARCO DE REFERENCIA SOA, PERSONALIZACIÓN CON SPRING FRAMEWORKS E INYECCIÓN DE DEPENDENCIAS."

"Trabajo de titulación presentado en conformidad a los requisitos establecidos para optar por el título de Ingeniero de Sistemas de Computación e Informática"

Profesora Guía:

Ing. Mayrita Valle Avendaño

Autor:

Antonio Alejandro Cañaveral Gil

Año

2012

DECLARACIÓN PROFESOR GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante, orientando su conocimiento y competencias para un eficiente desarrollo del tema y tomando en cuenta la Guía de Trabajos de Titulación correspondiente”.

_____

Mayrita Valle Avendaño
Ingeniera
60258302-3

DECLARACIÓN DE AUTORIA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen mis derechos de autor vigentes”.

_____

Antonio Alejandro Cañaveral Gil
171067996-8

DEDICATORIA

Esta tesis es la culminación de varios años de esfuerzo compartido. Mis compañeros y socios, por impulsarme a seguir adelante en este proyecto.
A mi esposa Paula, por el apoyo y las malas noches.
Y a mi hija Zara. No dejes nunca de intentar tocar la luna.

¡Gracias a Ustedes !

Antonio Cañaveral Gil

RESUMEN

En este trabajo de titulación se estudia al ERP de fuente abierta Adempiere, planteando el problema de la obsoleta arquitectura, frente a su potente funcionalidad y lógica de negocio.

Se analiza la arquitectura actual del sistema, mediante la técnica de ingeniería inversa, identificando todos los componentes del sistema. Funcionalidad, código fuente, diccionario de datos, persistencia, interfaces de comunicación, interfaces de usuario y servicios; elaborando artefactos propios de esta técnica.

Se aborda el análisis mediante un estudio de virtudes y falencias encontradas en la ingeniería inversa y proponiendo soluciones viables, con tecnología adecuada, moderna y extensible. Se usa de base Spring Framework con inyección de dependencias, grooby y un API de persistencia para la base de datos, además de protocolos de conexión como REST y Web Services. Se incluye en la propuesta la integración de un BPM y un CMS al ERP.

Parte del estudio trata sobre la forma y el orden de implementación de las mejoras.

Se concluye que el desarrollo del proyecto es viable, seguro y posible, identificando y resolviendo primeramente los elementos críticos del sistema, para que una vez resuelto se implementen el resto de mejoras. Se demostró la potencia de la lógica de negocio actual del sistema, justificando mantenerla en el nuevo modelo.

ABSTRACT

This study examines the open source ERP Adempiere, posing the problem of outdated architecture, compared to its powerful functionality and business logic.
It analyzes the current architecture, using the technique of reverse engineering, identifying all system components. Functionality, source code, data dictionary, persistence, communication interfaces, user interfaces and services developed artifacts typical of this technique.
Analysing through a study of strengths and weaknesses found in reverse engineering and proposing viable solutions, with appropriate technology, modern and extensible. Using Spring Framework dependency injection, Grooby and a persistence API for the database, and connection protocols such as REST and Web Services. Included in the proposal to integrate a CMS and BPM to ERP.
Part of the study deals with the form and order of implementation of the improvements.
We conclude that the project is viable, safe and feasible, first identifying and resolving critical system elements, so that once resolutely implement the remaining improvements. It demonstrated the power of the actual business logic of the system, justifying keeping it in the new model.

ÍNDICE

INTRODUCCIÓN

Adempiere es el ERP (Enterprise Resource Planning, Planeación de Recursos Emplesariales) de fuente abierta más potente y distribuido del mundo. Su núcleo (core) ha sido usado en varios proyectos nuevos de ERP como referencia de operación, algunos ejemplos son OpenBravo y Open ERP.

El sistema objeto de este estudio es Adempiere versión 3.6.0 LTS. Cuya licencia pública es GNU 2.0 con versión 1.2 de Junio de 1991.

La problemática de la arquitectura de Adempiere últimamente ha sido una encrucijada frente a su funcionalidad. Su persistencia obsoleta, la falta de integración con nuevas aplicaciones orientadas a servicios como BI (Business Intelligence, Inteligencia de Negocios) o BPM (Business Process Model, Modelamiento de Procesos de Negocio) y el requerimiento excesivo de esfuerzo en el desarrollo para realizar personalizaciones, han provocado plantearse una reestructuración de su framework y diccionario de datos; todo esto frente a la potente estructura financiera y contable que mantiene el sistema.

La tecnología SOA (Service Oriented Architecture, Arquitectura Orientada a Servicios) permite la creación de sistemas de información altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.

Spring framework ofrece libertad a los desarrolladores en Java, soluciones muy bien documentadas y fáciles de usar para las prácticas comunes de la industria, el manejo de inyección de dependencias facilitan la integración y el rediseño de la persistencia, característica deseable en Adempiere.

Tema del trabajo.

Re-Diseño de la Arquitectura del ERP de fuente abierta Adempiere, utilizando

el marco de referencia SOA, personalización con Spring Framewoks e inyección de dependencias.

Justificación de la idea.

En una implementación regular del ERP y principalmente en el desarrollo de verticalizaciones, es común diseñar una estructura propia y personalizada tanto de objetos de la base de datos, como clases y paquetes fuentes. Por ello que la problemática de mantener un código “trunk” estándar y varios paquetes customizados se convierte en un proceso engorroso, además si no se sigue una estricta metodología de implementación, en un momento ambos dejan de ser compatibles y el sistema dejará de actualizarse, volviéndose obsoleto. (En control de revisiones de software, el proyecto trunk es aquel que se implementa en tiempo real, donde se acometen los últimos cambios realizados por los desarrolladores).

El diseñar un framework en el cual la base de datos almacene incluso el código customizado, hacen que la potencia del ERP se simplifique a los módulos core, los cuales son parte del mismo framework, de tal manera que toda vertizalización se “inyecte” desde la propia personalización, programada con scripts, agregando dependencias.

Objetivos de la investigación

Objetivo general

Rediseñar el sistema de fuente abierta Adempiere ERP, orientándolo a SOA e incluyendo una capa de modelo que permita la inyección de dependencias mediante Spring Framework.

Objetivos específicos

- Realizar Ingeniería inversa del Sistema Adempiere ERP, para estudiar y comprender su arquitectura actual y funcionamiento, separando los

módulos de lógica de negocio del sistema.

- Diseñar un nuevo diccionario de datos, que admita el modelo de servicios y una capa para inyección de dependencias.
- Modificar y modernizar la persistencia de datos.
- Estructurar la arquitectura según los parámetros anteriores.

Referencia de la Tecnología a utilizar

Spring Framework

Spring Framework, conocido como Spring, es una plataforma de desarrollo de aplicaciones para java. Por su diseño ofrece mucha libertad para crear aplicaciones de alto performance, fáciles de probar, bien documentadas, reusables y sin ningún bloqueo en el código.
Spring Incluye:

- Inyección de dependencias con estilos de configuración por XML (Extensible Markup Language, Lenguaje de Marcas Extensible) y anotaciones.
- Soporte para programación orientada a aspectos para las variantes proxy-based y AspectJ-based.
- Apoyo a las transacciones declarativas, almacenamiento declarativo en caché y validación declarativa.
- Abstracciones de gran alcance para trabajar con especificaciones de Java EE comunes tales como JDBC (Java Database Connectivity, Conexión Jaba para Base de Datos), JPA (Java Persistence API, API de Persistencia Java), JTA (Java Transaction API, API de Transacciones Java) y JMS (Java Message Service, Servicio de Mensajes Java).
- Soporte para frameworks como Hibernate y Quartz.
- Un framework web flexible para la creación de aplicaciones RESTfull.
- Facilidades para pruebas enriquecidas e integradas.

Spring tiene un diseño modular permitiendo la adopción individual de los contenedores y servicios.

Soporte el despliegue de aplicaciones en un amplio rango de plataformas como Tomcat, Jboss e incluso WebSphere, además de plataformas “on cloud” (en la nube), como Heroku, Google App Engine y VMWare.

Spring Framework está liberado bajo la versión 2.0, Enero 2004, de la Licencia Apache. (Spring Source Community, 2012).

Persistencia (Hibernate)

Hibernate es una solución open source desarrollada por jboss para “mapear” bases de datos relacionales con objetos java. Hibernate es un ORM (Object Relational Mapping, Mapeo Objeto-Relacional) ya que no solo se encarga de referenciar objetos con la base de datos, sino que también proporciona consulta de datos y sistemas de recuperación de información.

Hibernate no es una buena solución cuando se trata de aplicaciones que se centran en el manejo de altos volúmenes de datos y cuando las aplicaciones utilizan procedimientos almacenados para implementar la lógica de negocio. Es más útil con modelos orientados a objetos y lógica de negocio basada en una capa intermedia de java. Hibernate ayuda a encapsular código de un proveedor específico SQL (Structured Query Language, Lenguaje Estructurado de Consulta), traduciéndolo mediante un leguaje común denominado HQL (Hibernate Query Languge, Lenguaje Hibernate de Consulta). (Hibernate Community, 2012).

El funcionamiento de Hibernate es sencillo, consiste en asociar a cada tabla de la base de datos un POJO (Plain Old Java Object, Objeto Ordinario Plano de Java). Un POJO es un objeto primario que tiene solamente tiene atributos, setter y getters, por ejemplo:

```
public class Cat {
    private String id;
    private String name;
    public Cat() {
    }
    public String getId() {
        return id;
    }
    private void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Para poder asociar el POJO a su tabla correspondiente, Hibernate usa ficheros XML, para “mapear” cada campo con un atributo de la tabla.

Desde la versión 3 de Hibernate, este funcionamiento evolucionó permitiendo el mapeo directo mediante anotaciones, lo que simplifica enormemente el desarrollo de un sistema con persistencia.

El uso de HQL permite, mediante un lenguaje (dialecto) intermedio, acceder y hacer consultas a la base de datos y de forma automática y transparente HQL transformará la sentencia en SQL, según el motor al que se quiera acceder como muestra la tabla 1 Dialectos de Hibernate.

TABLA 1. Dialectos de Hibernate

Fabricante	Dialecto
DB2	org.hibernate.dialect.DB2Dialect
DB2 AS/400	org.hibernate.dialect.DB2400Dialect
DB2 OS390	org.hibernate.dialect.DB2390Dialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
MySQL	org.hibernate.dialect.MySQLDialect
MySQL with InnoDB	org.hibernate.dialect.MySQLInnoDBDialect
MySQL with MyISAM	org.hibernate.dialect.MySQLMyISAMDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect

Fabricante	Dialecto
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Informix	org.hibernate.dialect.InformixDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Ingres	org.hibernate.dialect.IngresDialect
Progress	org.hibernate.dialect.ProgressDialect
Mckoi SQL	org.hibernate.dialect.MckoiDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Pointbase	org.hibernate.dialect.PointbaseDialect
FrontBase	org.hibernate.dialect.FrontbaseDialect
Firebird	org.hibernate.dialect.FirebirdDialect

Nota: Tomado de http://www.javabeat.net/qna/163-list-of-hibernate-sql-dialects

Web Services (Marco SOA)

Los Servicios Web son aplicaciones que residen en servidores centralizados, para intercambiar datos entre sistemas independientemente del lenguaje de programación en el que estén desarrolladas y de la plataforma en dónde se ejecuten.

Los Servicios Web disponen de una interfaz pública definida mediante un lenguaje común WSDL (Web Service Definition Language, Lenguage de Definición de Servicios Web) basado en XML que contiene los requisitos funcionales para establecer una comunicación con el Web Service. Esta comunicación se realiza por medio del protocolo SOAP (Simple Object Access Protocol, Protocolo Simple de Acceso a Objetos). (vaxMan, 2011).

El conjunto de servicios vinculados a tareas empresariales, es un estilo arquitectónico de TI denominado SOA (Service Oriented Architecture, Arquitectura Orientada a Servicios).

SOA permite la reutilización de los activos de TI existentes en una empresa,

interoperabilidad entre aplicaciones heterogéneas y tecnologías diversas. SOA proporciona un nivel de flexibilidad que antes no era posible en el sentido que:

- Los servicios son componentes de software con interfaces bien definidas que son independientes de la implementación, separando el qué (interfaz de servicio) con el cómo (implementación).
- Los servicios son autónomos, realizan tareas predeterminadas y ligeramente acopladas, por la independencia.
- Los servicios pueden ser dinámicamente resueltos, en cuanto a su estructura.
- Se pueden crear servicios compuestos a partir de otros servicios.

SOA utiliza el paradigma find-bind-execute (encontrar-unir-ejecutar), en este paradigma, los proveedores registran sus servicios públicamente. Este registro es utilizado por los consumidores para encontrar servicios que cumplan ciertos criterios. Es indispensable realizar aplicaciones que transformen cualquier funcionalidad en servicio para una correcta integración a la arquitectura SOA. (Qusay H. Mahmoud, 2005).

Groovy

Groovy es un lenguaje dinámico orientado a objetos para la Máquina Virtual de Java. Ofrece la posibilidad de revisar, escribir y compilar estáticamente el código, es decir, en tiempo de ejecución.

Proporciona un dialecto de scripting muy similar a Java, menos restrictivo y con un único archivo en el cual se especifica toda la estructura de una clase o un método, en el cual main(String[]args) (Método de entrada para una aplicación Java) está embebido, permitiendo obtener el resultado directamente en el mismo script.
Usando el siguiente código, se obtiene el resultado de Hello World.

```
def name='World'; println "Hello $name!"
```

Para implementar una clase y obtener el resultado de su ejecución, se realiza de la misma forma.

```
import static org.apache.commons.lang.WordUtils.*
class Greeter extends Greet {
  Greeter(who) { name = capitalize(who) }
 }
new Greeter('world').salute()
```

(Spring Source & Groovy Community, 2012)

Adempiere ERP+CRM

Compiere es el primer ERP de software libre de primera clase. Con historia de diseño desde 1982, El desarrollo de Compiere fue patrocinado en el año 2000 por GoodYear en Alemania. Su creador Jorg Janke tiene un fuerte background en ERP, SAP, Oracle, Diccionario Activo, java, etc. Compiere es un nombre italiano y significa “hacer” Después de 5 meses sin liberar versiones, en privado anunciaron cambios en contratos de soporte, Venture Capital invirtió en la compañía, en Agosto de 2006 la dispersa comunidad de Compiere preocupada porque el proyecto se cerrara, inicio una caliente discusión pública sobre hacer una bifurcación del proyecto y finalmente lo lograron, constituyendo el proyecto Adempiere, del italiano “cumplir”. (Comunidad Adempiere, 2009).

La evolución del proyecto fue rápida, durante 100 días Adempiere fue el número 1 en Souceforge (repositorio de más 173.000 programas).

La meta inicial de Adempiere fue liberar una versión estable, sin mucha funcionalidad adicionada, esta meta se cumplió el 1 de mayo de 2007, con nuevas características grandes como soporte PostgresSQL, soporte Oracle XE, nuevos arreglos, herramientas de migración, muchas mejoras de desempeño, muchos bugs arreglados, herramientas 2 pack, documentación de Wiki, integración Jasper y flexibilidad en el validador del modelo. (Comunidad

Adempiere, 2006).

Adempiere es un proyecto orientado a la comunidad. No tiene ninguna compañía propietaria. Todas las decisiones sobre el proyecto, su estructura, marketing y código, se definen en asambleas en el marco de la comunidad Adempiere, definida con la siguiente estructura:

Consejo: Grupo de 9 fundadores.

Líder: Los fundadores del consejo votaron por el líder, Redhuan Daniel Oon de Malasia. Participante activo del foro de Compiere desde julio de 2004.

Business Development Committee (BDC): tiene 4 miembros, el propósito de ellos es dar la bienvenida a las empresas interesadas en el proyecto.

Project Management Committee (PMC): este es un comité de 8 miembros que proporcionan una guía y dirección al proyecto.

Comunidad: en muchos casos la comunidad es consultada para votar por medio de foros para la toma de decisiones. (Comunidad Adempiere, 2009).

La arquitectura de Adempiere (heredada de Compiere), está diseñada con la intención de seguir los cambios en la medida en que evolucionan los negocios. Al mismo tiempo los clientes pueden cambiar la estructura de la información, ajustándose a las nuevas necesidades de información.

La estructura de Adempiere difiere de los sistemas tradicionales dado que su diseño está basado en los procesos del negocio en lugar de utilizar la base tradicional de departamentos o módulos. Este enfoque refleja las condiciones del negocio en empresas más pequeñas, donde los procesos relacionados son manejados frecuentemente por la misma persona. Este enfoque se muestra en la tabla 2, Módulos del Sistema Adempiere:

TABLA 2. Módulos del Sistema Adempiere

Módulos Tradicionales	**Cotización a Efectivo**	**Requisición a Pago**	**Manejo de Clientes**	**Manejo de Socios**	**Cadena Abastecimiento**	**Análisis de Desempeño**
Contabilidad General						X
Cuentas por Pagar		X			X	X
Cuentas por Cobrar	X		X			X
Órdenes de Compra		X	X		X	X
Órdenes de Venta	X		X		X	X
Inventarios	X	X	X		X	X
Activos Fijos						X
CRM	X				X	

Adempiere es un sistema multifuncional, que propone a los datos en catálogos muy bien definidos, para garantizar su extensibilidad. Adempiere es:

- multi-compañía (multi-cliente)
- multi-organización
- multi-costeo
- multi-moneda
- multi-lenguaje
- multi-impuestos
- multi-contabilidad.

CAPÍTULO 1

1. INGENIERIA INVERSA DEL SOFTWARE ADEMPIERE

1.1. Introducción

Se analiza la arquitectura actual del software Adempiere, identificando sus componentes, desde la estructura de datos hasta el código que implementa la funcionalidad de los módulos del sistema.

En el proyecto Adempiere existen aplicaciones y módulos que han sido identificados como obsoletos, así como otras aplicaciones que tienen un funcionamiento personalizado para implementaciones específicas, las cuales no serán objeto de estudio en esta tesis.

La técnica para este análisis se denomina ingeniería inversa, la cual consiste en realizar un estudio profundo de un sistema, para obtener información, diagramas y documentación, a partir de un producto existente y en funcionamiento.

Existen tres tipos básicos de ingeniería inversa:

- Ingeniería inversa de datos.
- Ingeniería inversa de lógica o de proceso.
- Ingeniería inversa de interfaces de usuario.

Para todas ellas se utilizan herramientas de software, que facilitan la tarea de análisis, como son, depuradores, desensambladores, compiladores inversos y sistemas de documentación para generar diagramas de clases y diagramas Entidad Relación (ER). (Miguel-Angel Sicilia, 2012).

1.1.1. Esquema del Sistema y Funcionamiento

El esquema actual del sistema Adempiere se presenta en la figura 2.1 Esquema del Sistema Adempiere.

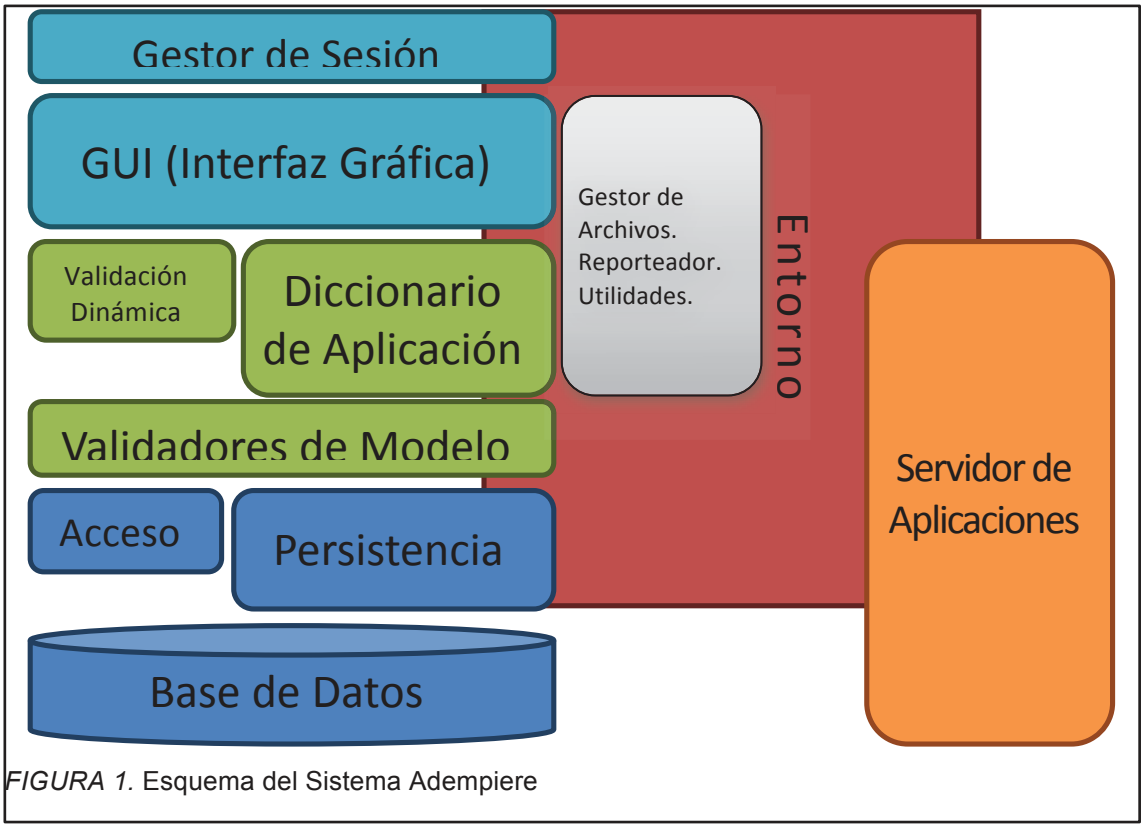


FIGURA 1. Esquema del Sistema Adempiere

En la figura 2.1 Esquema del Sistema Adempiere, se observa que es un sistema Modelo Vista Controlador (MVC) desarrollado en 3 capas, con un diccionario de aplicación que fusiona la capa modelo con la de controlador y algunas peculiaridades que se detallan:

- La primera capa es la de datos o Modelo, que cuenta con un motor de persistencia, herramientas de consulta y acceso directo a la data. El modelo se define en el diccionario de aplicación.
- La capa controlador se define parcialmente en el diccionario de aplicación, pero la mayor funcionalidad está establecida claramente en la lógica de negocio del sistema.
- La capa Vista o GUI (Graphical User Interface, Interfaz Gráfica de

Usuario), está fuertemente vinculada al diccionario de aplicación, ya que en este define la forma de presentación de las ventanas.

Resumiendo, se tiene un modelo MVC que funciona transversalmente con el diccionario de aplicación, como muestra la figura 2.2 Esquema MVC.

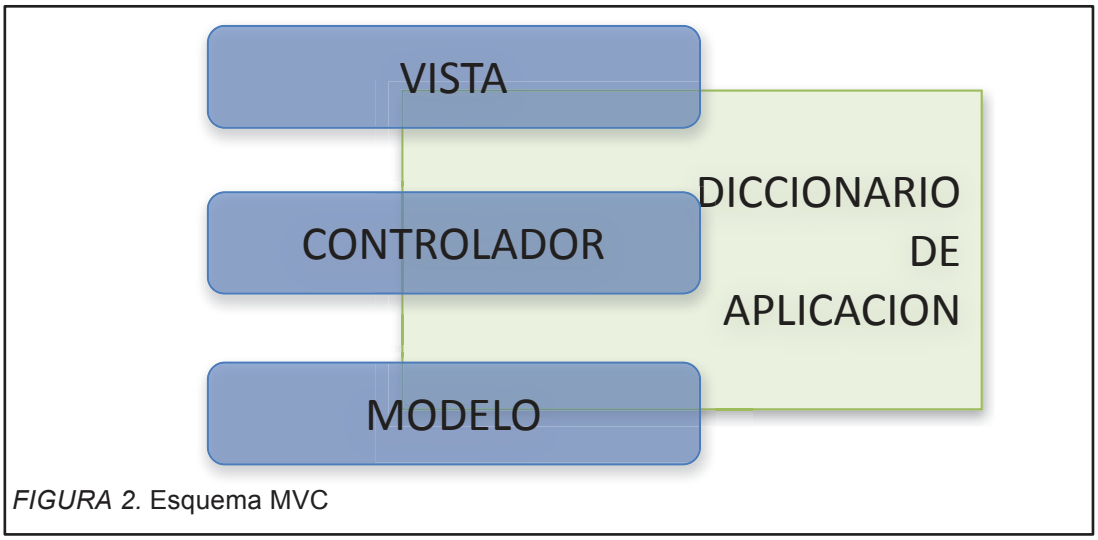


FIGURA 2. Esquema MVC

La sesión del usuario se gestiona adecuadamente, aunque existen falencias de seguridad detectadas en el análisis. La principal es que cuando no se usa SSO (Single Sign On, Sistema de Autenticación Reducida) el sistema despliega las contraseñas de los usuarios en texto plano, tanto en el log, como en la base de datos. Esto no permitiría que el sistema funcione Stand Alone y siempre dependa de un servidor LDAP (Lightweight Directory Access Protocol, Protocolo Ligero de Acceso a Directorios).

El sistema cuenta con dos interfaces gráficas, una interfaz desarrollada en Java Swing, la cual fue creada en conjunto con el resto del sistema y una nueva interfaz RIA (Rich Internet Application, Aplicaciones de Internet Enriquecidas) con el toolkit (Conjunto de Herramientas que facilitan el desarrollo de una aplicación) ZK, liberada desde la versión 3.41 de Adempiere.

Una de las limitaciones a estudiar es la fuerte dependencia que mantiene la interfaz Swing con el sistema, a diferencia de la RIA, que está completamente

desacoplada.

Sin embargo desde el inicio del proyecto Compiere, se ha estandarizado el funcionamiento de la interfaz, proporcionando una barra de herramientas que accede a todas las funcionalidades CRUD (Create,Read,Update&Delete Crear,Leer,Actualizar y Borrar), utilidades de impresión, reportería, archivo digital y correo.

Desde la interfaz se accede a todas las ventanas del sistema mediante el menú que se genera a partir del diccionario de aplicación. Las ventanas, que también son generadas desde el diccionario de aplicación, muestran la información validada tal como se parametrizó.

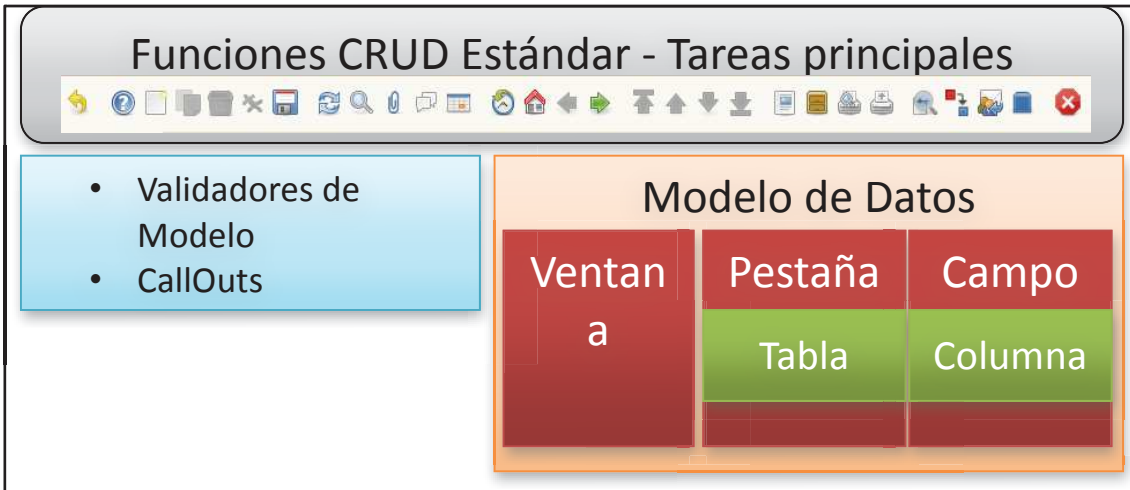


FIGURA 3. Esquema GUI.
(Comunidad Adempiere, 2012)

Uno de los componentes clave de Adempiere, es el servidor de aplicaciones. Este puede ser GlassFish o JBoss. En él, el sistema despliega los servidores de correo, de autenticación LDAP, el servidor web y los servicios de ejecución asíncronos, que son usados para contabilizar, enviar alertas, procesar solicitudes y correr cualquier proceso que se haya desarrollado en el API de Adempiere y se necesite ejecutar bajo un esquema de tarea programada.

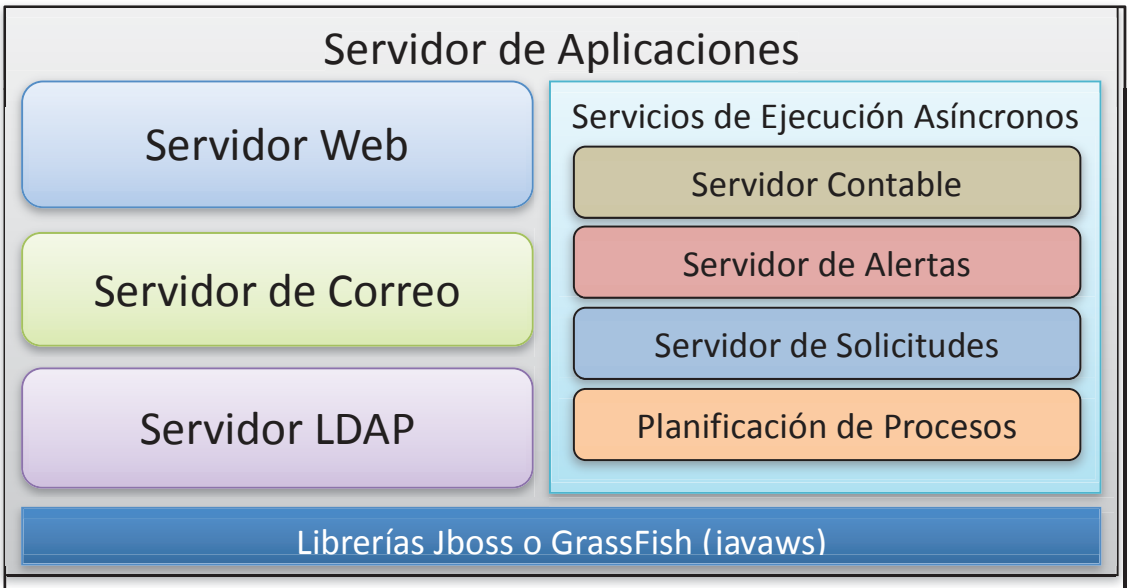


FIGURA 4. Esquema Servidor de Aplicaciones

1.1.2. Análisis General de la Base de Datos Adempiere

Adempiere soporta dos marcas comerciales para el sistema gestor de base de datos, estas son PostgreSQL y Oracle XE. En ambas la estructura es idéntica en cuanto a objetos de la base de datos, 790 tablas, 145 vistas y 70 funciones, (Es un dato generalizado, depende de la versión de Adempiere instalada, los parches aplicados y los módulos).

Existe una buena cantidad de funciones de base de datos implementadas para la solución, esto es un problema que se analizará en el punto 3.1.2.2 Persistencia y Base de Datos, ya que limita fuertemente la portabilidad del sistema.

Analizando las tablas de la base de datos claramente se observa que la nomenclatura usada es la siguiente:

MODULO_NombreTabla
MODULO_Nombre_ExtensoTabla
MODULO_NombreTabla_trl

Así obtenemos el cuadro de definiciones de módulos de la base de datos presentado a continuación:

TABLA 3. Módulos de la Base de Datos

Prefijo	Módulo	Descripción
A	Asset	Activos Fijos
AD	Application Dictionary	Diccionario de Aplicación
C	Core	Tablas del Core de Adempiere, entre las más importantes, clientes, facturas, impuestos,etc.
FACT	Contabilidad	Datos de Contabilidad (Cubo Contable)
GL	General Ledger	Libro Mayor
HR	Human Resources	Recursos Humanos
I	Import	Importación de Datos
M	Material Management	Administración de Materiales, Productos y Stock
T	Temporary	Tablas Temporales

No hay que confundir los módulos de la base de datos con los módulos de la aplicación, en la base de datos se dividen los módulos desde el punto de vista funcional y en la aplicación, desde el operativo. Como parte de la refactorización de la aplicación, se propondrá una mejor estructura de la base de datos.

En el análisis de las tablas también se observa que todas, excepto las de traducción, tienen 7 campos que se tornan obligatorios en el entorno del sistema.

TABLA 4. Campos Obligatorios de las Tablas del Sistema

Campo	Tipo	Descripción
AD_Client_ID	Numeric(10)	Define el cliente al que el registro afecta.
AD_Org_ID	Numeric(10)	Organización al que el registro

		pertenece, 0 para la organización * (asterisco).
Created	Timestamp	Fecha de creación del registro.
CreatedBy	Numeric(10)	Usuario que creó el registro.
Updated	Timestamp	Última fecha de actualización del registro.
UpdatedBy	Numeric(10)	Último usuario que actualizo el registro.
isActive	Char(1)	El Registro está o no activo. Valores (Y/N)

Estos campos se utilizan para elaborar la auditoría de los registros.

Además de estos 7 campos, el sistema espera que cada tabla tenga un identificador único que se llame exactamente igual que la tabla, pero con la terminación _ID. Por ejemplo si se crea la tabla UDLA_Ejemplo, debe tener un campo Primary Key llamado UDLA_Ejemplo_ID.

1.1.3. Análisis general del código fuente

Adempiere es una aplicación desarrollada totalmente en JAVA. Actualmente funciona bajo la versión 1.6 de la máquina virtual de java.

Según el código fuente el sistema se divide de la siguiente manera:

- Módulo base
- Módulo client
- Módulo zkwebui
- Módulo Glassfish & JBoss facet
- Módulo serverRoot

1.1.3.1. Módulo base

Este módulo es el encargado de implementar la funcionalidad completa del Diccionario de Aplicación, persistencia, modelo de datos, definición de la lógica de negocio contable, reporteadores, utilerías y visores de archivos.

Como Adempiere hereda su código fuente del proyecto Compiere, la mayoría de paquetes en el módulo base heredan de org.compiere.

TABLA 5. Paquetes del módulo base

Paquete	Subpaquete	Definición
org.compiere		La clase Adempiere carga el sistema desde la interfaz swing.
	Acct	Contiene la lógica de negocio del motor contable y los tipos de documentos.
	Db	Maneja las conexiones de la aplicación a la base de datos y proporciona utilerías.
	Dbport	Utilerías de homogenización entre el motor de PostgreSQL y Oracle.
	FA	Utilerías para el libro mayor y depreciación
	Interfaces	Proporciona una interfaz entre el servidor de aplicaciones y el core.
	Model	Contiene todas las clases del modelo de datos y los objetos de persistencia.
	Plaf	Proporciona los temas (skins) para el cliente swing de Adempiere.
	Print	Funcionalidad para el reporteador nativo de Adempiere
	Process	Definición del motor de procesos y clases con todos los procesos que ejecuta el core y el diccionario de aplicación.
	Report	Utilerías adicionales para el reporteador nativo de Adempiere. Y funcionalidad para los reportes financieros.
	Swing	Extensiones de la librería nativa Swing de java para los componentes de la aplicación.

	Útil	Utilerías de log, mail, sincronización de operaciones, cargadores de archivos, evaluadores, conversores, etc.
org.adempiere	Exceptions	Clases para manejar las excepciones de Adempiere
	Model	Clases adicionales para manejar el diccionario de datos, se implementa la Clase GenericPO para manejar la persistencia de una forma abstracta.
	Pipo	Utilería para importar y exportar definiciones y paquetes del diccionario de aplicación.
	Plaf	Nuevo tema para la interfaz swing de Adempiere.
	Process	Procesos Adicionales para el framework de Adempiere

Es evidente que desde el proyecto Compiere, se estructuró inadecuadamente el código del sistema, además es evidente que el crecimiento descontrolado y sin estándares en la aplicación influyen en la desorganización.

1.1.3.2. Módulo Client

Dentro de este módulo se implementa la mayoría de la interfaz Swing del sistema. Esto ratifica la desorganización existente en el código del sistema.

TABLA 6. Paquetes del módulo client

Paquete	Subpaquete	Definición
org.compiere	Acct	Visores de contables
	Apps	Ventanas auxiliares, login, consultas, manejador de vistas de componentes
	Grid	Motor de ventanas estándar del sistema
	Minigrid	Renderizadores de vistas de tabla.

	Print	Visores de reportería
org.adempiere.apps	Graph	Herramientas para manejar gráficos.

1.1.3.3. Módulo zkwebui

Módulo que contiene toda la interfaz Web RIA del sistema.

Todos los paquetes org.adempiere.webui implementan una interfaz independiente del sistema Adempiere.

El proyecto zkwebui es un claro ejemplo de independencia, ya que mantiene la ejecución y funcionalidad del core separada de la interfaz, que funciona como una aplicación web independiente.

1.1.3.4. Módulo glassfish & jboss - facet

Inicia los contextos de los entornos de servidor glassfish y jboss, implementa clases de autenticación y login.

1.1.3.5. Módulo serverRoot

El paquete serverRoot gestiona los procesadores asíncronos, en la tabla 7 Paquetes del módulo serverRoot, se describen los componentes.

TABLA 7. Paquetes del módulo serverRoot

Paquete	Subpaquete	Definición
org.compiere	Session	Gestiona los Bean de estado y de servidor.
	Ldap	Conectores, procesadores de mensajes y resultados.
	Server	Procesador contable, de

		mensajería, replicación, planificador y de flujos de trabajo.
	Web	Monitor web de Adempiere

Para facilitar el estudio del Sistema y agilizar la integración de los nuevos componentes. Se decide dividir al ERP en 2 grandes grupos de funcionalidad:

- Diccionario de Aplicaciones o Framework.
- Estructura Operativa Financiera.

1.2. Diccionario de Aplicaciones (Framework)

1.2.1. Función del Diccionario de Aplicaciones

Adempiere cuenta con un framework, integrado al código del sistema en el cual está implementado el 90% de los elementos del ERP, esto es, listas, campos, referencias, tablas, columnas, ventanas y menú, al cual Adempiere lo denomina **Diccionario de Aplicaciones.**

El diccionario de la aplicación es una de las características más importantes de Adempiere. Prácticamente, todas las nuevas funcionalidades se pueden manejar mediante cambios en el diccionario, en la mayoría de los casos las personalizaciones pueden ser configuradas directamente en el diccionario sin requerir compilación o desarrollo. Desde aquí se puede definir que campos van a aparecer en una ventana, añadir columnas adicionales a una tabla para registrar información contemplada en una personalización, definir como va a estar organizado el menú, filtros dependientes en los distintos combos o búsquedas del sistema, llamadas a CallOuts o Validadores desarrollados en alguna librería integrada al sistema, además del propio modelo de negocio que se personalice en la implementación de un proyecto ERP.

1.2.2. Estructura del Diccionario

En la figura 5 Estructura del Diccionario, se esquematiza la configuración del diccionario de aplicación en su estado actual.

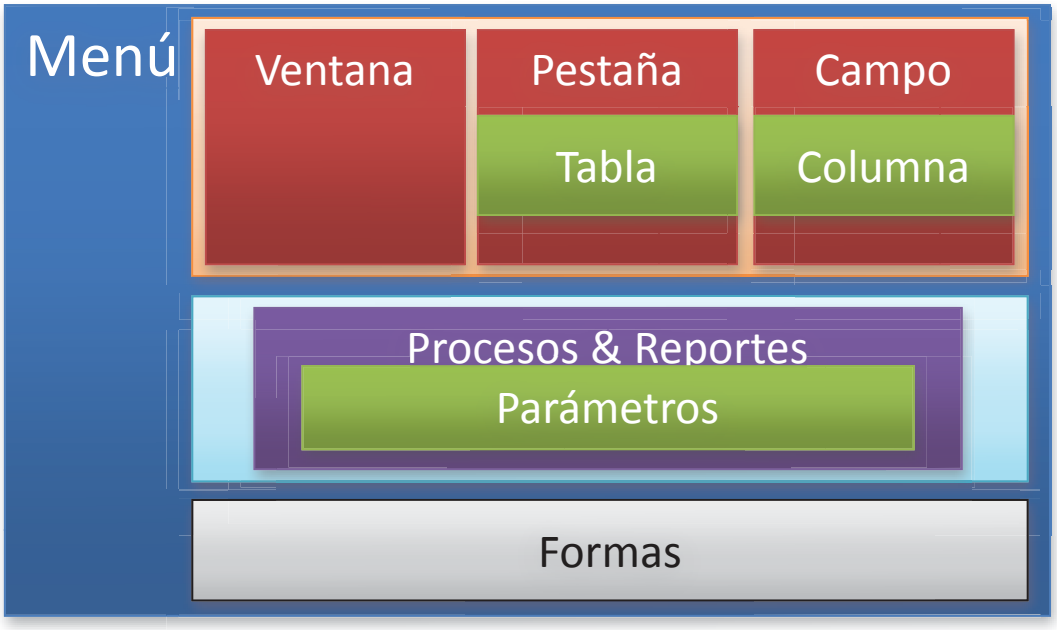


FIGURA 5. Estructura del Diccionario.
(Comunidad Adempiere, 2012)

Se analiza el Diccionario de Aplicación de Adempiere, desde el objeto más atómico (Elemento del diccionario), hasta el más global (Menú).

1.2.2.1. Tipo de Entidad

Determina la propiedad de los Objetos del Diccionario de Aplicación, esto quiere decir que cuando se cree algún elemento dentro del diccionario, siempre se debe relacionar con el tipo de entidad al que ese objeto pertenece. Más adelante esto servirá para definir las clases del modelo relacionadas con los objetos de Adempiere. Existen dos entidades que Adempiere las gestiona y mantiene:

- Dictionary
- Adempiere

Estas entidades definen los elementos y clases bajo el paquete

org.compiere.model.

Para Definir un Tipo de Entidad, se agrega un registro en la tabla AD_EntityType, proporcionando al menos los valores de los campos EntityType y Name.

TABLA 8. Tipo de Entidad

AD_EntityType		
Columna	**Tipo de Dato**	**Descripción**
EntityType	Cadena (40)	Identificador alfanumérico del tipo de entidad.
Name	Cadena (120)	Nombre unívoco del tipo de entidad.
Description	Cadena (255)	Descripción del tipo de entidad
Version	Cadena (20)	Descripción de la versión del tipo de entidad
ModelPackage	Cadena (255)	Paquete java en el cual se crearán las clases del modelo de datos (org.compiere.*)

Nota: Tomado del Diccionario de Datos Adempiere.

1.2.2.2. Elemento

El Elemento es el objeto más atómico dentro del diccionario de aplicación. Se utiliza para nombrar a las columnas, darles una descripción, traducción y definir textos de ayuda.

Existen alrededor de 3000 elementos en Adempiere donde constan identificadores de columnas y campos. No es óptimo que la traducción esté a tan bajo nivel, mas adelante se propondrá un nuevo esquema de traducción.

1.2.2.3. Referencia

Adempiere define como referencia a cada tipo de dato, listas de validación y sus parámetros y validaciones de tablas.

TABLA 9. Tipos de Datos

Tipo de Dato Adempiere	Tipo de Dato Base de Datos	Descripción del campo y la interfaz
Account	Entero	Almacena un entero en la base de datos. Muestra un componente con la información de la dimensión contable.
Ammount	Número (10,4)	Almacena un BigDecimal, Presenta un componente tipo moneda.
Assignment		
Binary	Binario	Almacena Datos Binarios en la Base de Datos, presenta un componente para cargar un archivo.
Button	Texto	No almacena valores en la base de datos, presenta un botón para ejecutar una acción.
Date	Date	Almacena una Fecha, presenta un componente de fecha
Date+Time	Timestamp	Almacena un timestamp sin zona horaria, presenta un componente fecha hora
ID	Entero	Almacena un entero en la Base de Datos, presenta un Campo de Texto con el valor entero.
Image	Binario	Almacena un binario, presenta una imágen
Integer	Entero	Almacena un Entero, presenta un componente de manejo de enteros
List	Cadena	Almacena una Cadena, presenta un componente de lista.
Location (Address)	Entero	Almacena un entero, Despliega un componente para definir una dirección
Locator (WH)	Entero	Almacena un entero, Despliega un componente para definir una localización de almacén.
Memo	Texto	Almacena un texto, Despliega un componente de texto ampliado
Number	Numero	Almacena un BigDecimal, Presenta un componente para editar números
Product Attribute	Numero	Almacena un entero, presenta un componente para definir atributos a un producto.
Quantity	Numero	Almacena un BigDecimal, Presenta un componente para editar números
String	Cadena	Almacena una cadena, Presenta un componente de texto
Table	Entero	Almacena un entero, Presenta un buscador de la tabla a buscar.
Table Direct	Entero	Almacena un entero, Presenta un combo con los valores de la tabla
Text	Texto	Almacena y presenta un texto extendido.
Text Long	Texto	Almacena un texto, presenta un componente de edición y visualización HTML
Time	Timestamp	Almacena un timestamp, presenta un componente para visualizar horas.
URL	Cadena	Almacena una cadena, presenta un componente para acceder directamente a la URL
Yes-No	Carácter	Almacena un carácter (Y/N), presenta un check box.

Nota: Adaptado del Diccionario de Datos Adempiere.

1.2.2.4. Tabla y Columna

Es donde se enlaza el modelo de negocio de Adempiere, con la base de datos.

Aquí se definen todas las tablas y vistas, con sus respectivas columnas, tipos de datos y comportamiento de los mismos.

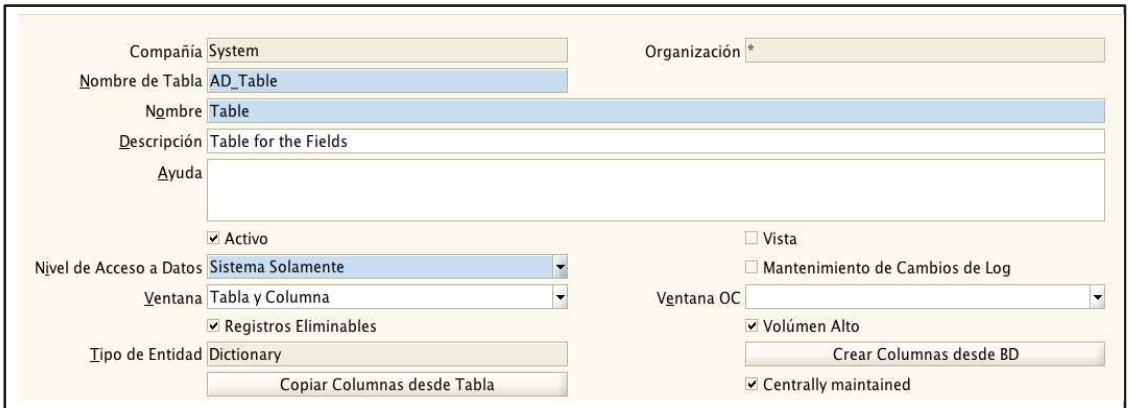


Figura 6. Ventana Tablas.

Los nombres son sensibles a las mayúsculas. Si la tabla tiene un ID , este debe ser incluido en el nombre,(por ejemplo : para la tabla MODULO_NombreTabla la columna ID debe ser llamada exactamente MODULO_NombreTabla_ID).

Vista: indica que la tabla que se está definiendo es una vista, las vistas no son sincronizadas en la base de datos (es decir no se puede crear la vista a partir del botón “crear columnas de la BD”, la vista debe ser creada directamente desde la BD). Pueden ser usadas además para hacer una tabla de solo lectura (Read only).

Nivel de acceso a los datos es usado para definir el acceso default para los roles (Maneja: sistema, compañía y organización).

Mantenimiento de cambios de Log: cuando está marcada todos los cambios realizados sobre la tabla son guardados en AD_ChangeLog. Para la adecuada auditoría de la tabla, se debe activar también el campo Mantenimiento de cambios de log.

Ventana: define la ventana habilitada para la funcionalidad de acercar, además se puede definir otra ventana de acercar diferente para procesos de compra

(ventana PO).

Registros Eliminables: permite habilitar y deshabilitar el borrado de registros de la base de datos.

Volumen Alto: indica si una pantalla mostrara la ventana de búsqueda al abrir la ventana. (Cuando se tiene muchos registros en una tabla en vez de mostrar la tabla con toda la información, muestra un filtro previo).

Crear columnas de la BD si se hacen cambios en la base de datos se puede obtener los cambios en Adempiere, con este botón.

Copiar columnas desde tabla. Mediante este botón se logra crear una tabla de la forma más rápida, se selecciona una tabla similar y este proceso creará todas las columnas con nombres exactos a la original (el ID será renombrado para que concuerde con el nombre de la tabla). (Cañaveral, 2009.)

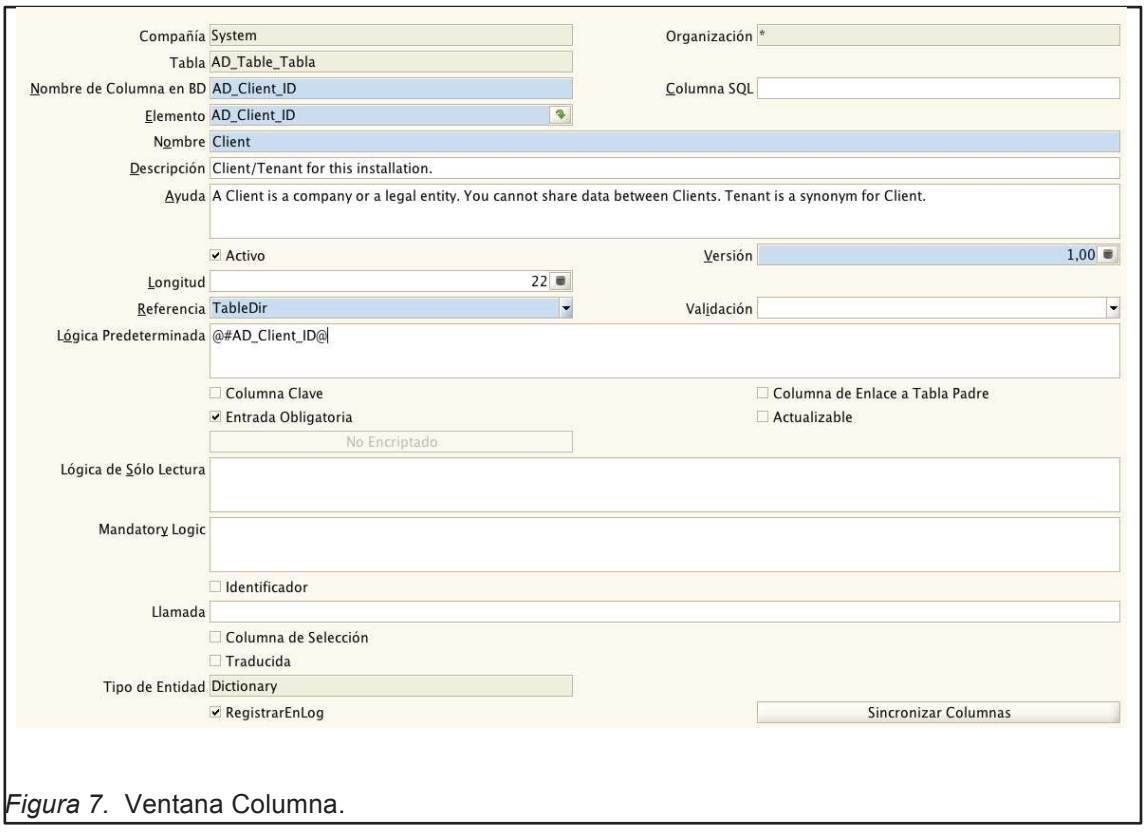


Figura 7. Ventana Columna.

La relación de Columna con Tabla es de 1:N, permitiéndonos crean N columnas por cada tabla que exista en el sistema.

Elementos del sistema: el nombre de la columna de BD, nombre, descripción y traducción se heredaran del elemento.

Nombre de la columna de la BD, es el nombre exacto de la columna de la BD.

ColumnaSQL es utilizado para definir columnas virtuales, pude mostrar información resumida, o información de otras tablas sin necesidad de adicionar una columna real a la bd. es construido por un select que tiene un join con algún campo de la tabla principal.

Referencia: define el tipo de dato de una columna, cada una de las referencias indica un comportamiento diferente en la interfaz GUI.

Validación: sirve para cambiar dinámicamente las listas y búsquedas.

Valor de referencia: lista estática para tablas y listas.

Formato de Valores: para los campos de tipo strings, poder definir un formato para el campo.

Lógica Predeterminada- variables de contexto –sentencias sql. Se pueden definir varios default separados por ";" la primera encontrada no nula será la por defecto
Columna llave. Solo una por tabla (primary key – normalmente ID generado internamente no se muestra a los usuarios)

Columna de enlace a tabla padre: define una relación de hijo con una o mas tablas (pueden ser tablas sin ID pero con uno o mas padres-como tablas de

acceso) es cuando la columna es un ID de otra tabla.
Siempre actualizable: hace al campo actualizable incluso si este es procesado.

Encriptación es solo para campos de tipo cadena, se puede elegir datos, asegurándose que el campo tenga una longitud suficiente para mantenerse ajustado.

Lógica de solo lectura. Condición para hacer el registro de solo lectura. (por default las columnas isactive y proceseed marcan las columnas como de solo lectura, sin necesidad de llenar este campo).

Lógica de obligatorio. Condición para hacer el campo obligatorio

Identificador: Una o más columnas (normalmente valores y/o nombres) son mostradas en lista y para ser usados en reportes, los identificadores son mostrados en el orden definido en el campo **secuencia**.

Callout, pieza de código (personalización) utilizada para llenar otros campos, o simples validaciones (no recomendada para validaciones, es necesario validar para guardar) en los campos cadena el callout se dispara cada vez que se ingresa una letra con el teclado.

Columnas de selección cuando este campo es marcado, la columna se muestra en la ventana de búsqueda como default para realizar las búsquedas.

Traducido Para definir traducciones para una columna, en este caso se necesita definir una tabla y un tab con el mismo nombre de la tabla original y el sufijo (_Trl), y crear la tabla con la misma clave que el padre, columna del lenguaje, y la columna traducida. (Cañaveral, 2009.)

1.2.2.5. Ventana Pestaña y Campo

La ventana “ventana, pestaña y campos” define la presentación GUI de tablas y columnas dentro de cada ventana. La ventana de tipo transacción es para ingresar documentos. La de mantenimiento es para tablas parametrizables. La de solo consulta presenta la ventana sin posibilidad de editar los datos.

Las ventanas de Transacción solo muestran los registros pendientes o los creados en las últimas 24 horas (el usuario puede ver todos los registros usando el icono de historial)

Adempiere presenta de forma estándar los datos dividiendo el contenido en pestañas. Dependiendo del GUI, se presentarán a la izquierda, derecha o en la parte inferior.

Los campos son los homónimos visuales de las columnas, presentándose en la ventana bajo un constructor genérico el cual usa el campo orden y el campo misma línea para construir la ventana, como en el ejemplo de la tabla 10 Ejemplo de Construcción de Ventana.

TABLA 10. Ejemplo de Construcción de Ventana

Campo	Orden	Misma Línea
Campo1	10	No
Campo2	20	Si
Campo3	30	No
Campo4	40	Si
Campo5	50	No
Campo6	60	Si
Campo7	70	Si

El resultado de la ejecución del ejemplo anterior se mostrará en la interfaz tal como en la figura 8 Ejemplo de Ventana

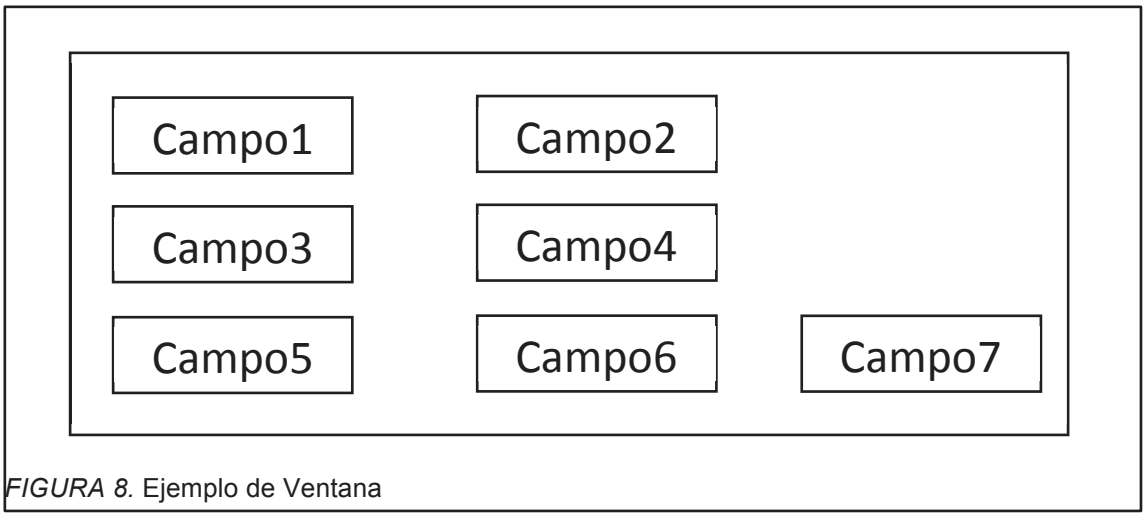


FIGURA 8. Ejemplo de Ventana

Además en la definición del diccionario de los campos, se pueden sobre escribir propiedades de las columnas, como son, tipo de dato (siempre y cuando se utilice otro que sea compatible con el origen), lógica predeterminada, definir si el campo es obligatorio y formato de salida.

1.2.2.6. Procesos y Reportes

Los procesos se utilizan para ejecutar acciones en ventanas de Adempiere o invocándolos desde el menú directamente.

En las ventanas se llaman directamente desde los tipos de datos botón, al cual se le define que proceso debe ejecutar.

Tanto los procesos como los reportes se asemejan en su funcionamiento:

- Un reporte puede tener un pre-proceso.
- Un proceso puede mostrar información cuando finalice.
- Ambos utilizan parámetros, que son parecidos a las columnas y los campos, se puede definir valores por default y rangos.

Los procesos del Sistema implementan la clase abstracta SvrProcess, objeto de estudio en el siguiente capítulo.

1.2.2.7. Formas

En algunas ocasiones, la funcionalidad requerida no se puede satisfacer con las ventanas estándar de Adempiere para lo cual se requiere definir ventanas personalizadas que manejen una estructura especial. Para ello se utilizan las formas.

Las formas se definen en el diccionario de aplicación especificando la clase java que ejecuta la ventana personalizada, de esta manera se puede agregar al menú el elemento forma en donde se requiera.

Una de las falencias de Adempiere es precisamente las herramientas requeridas para desarrollar formas. Actualmente las formas hay que programarlas definiendo cada elemento sin que exista una interfaz para hacerlo.

1.2.2.8. Menú

El menú de Adempiere es parte del diccionario de aplicación. Es una ventana especial con disposición de árbol, en la cual se estructura el menú del usuario. Para crear el árbol se utiliza drag&drop sobre los elementos.

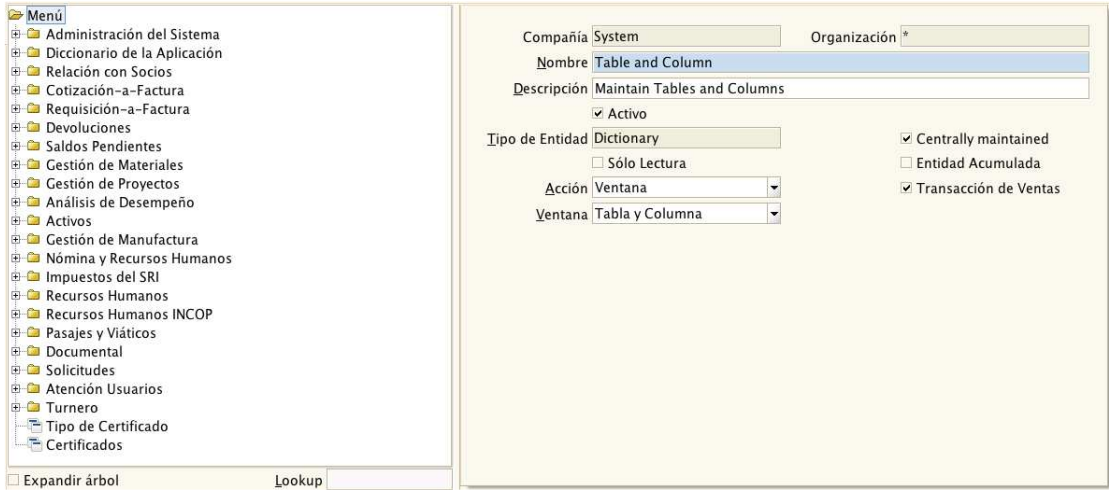


Figura 9. Ventana Menú.
Tomado de Adempiere ERP.

Solo Lectura: Muestra un elemento del menú en modo solo lectura.

Entidad Acumulada crea un menú contenedor, para añadir ítems.

Acción: Permite elegir el componente que se creará en el menú (ventana, reporte, proceso o forma).

Ventana-Proceso-Forma: Despliega la lista de estos elementos, para elegirlos al crear el menú**.**

Transacción de Ventas despliega el objeto en modo venta.

Para asignar un elemento del menú a un rol determinado, este solamente debe tener permisos de acceso al elemento. No se define un menú para cada rol, sino uno genérico para todo el sistema. (Cañaveral, 2009)

1.2.2.9. Elementos Adicionales del Diccionario

Grupos de Campos: Permite definir subsecciones en una pestaña. Se pueden agrupar los campos que tengan relación entre sí y colapsar o extender la sección.

Mensajes: Maneja todos los mensajes multi-idioma del sistema. Estos son cargados en caché al iniciar.

Vistas del Reporte: Formatos preestablecidos y vinculados con pestañas del sistema para desplegar reportes.

Reglas de Validación: Filtros SQL o Java, definidos para los tipos de dato TableDirect, List y Table.

Regla: Nueva funcionalidad de Adempiere acogida y liberada desde la versión

3.5 en la cual se generan reglas desarrolladas en groovy para extender el sistema.

1.2.3. Principales Tablas del Diccionario de Aplicaciones

Existen alrededor de 170 tablas y más de 2500 campos que estructuran el diccionario de aplicaciones de Adempiere.

Se centrará en aquellas que son el core del diccionario y en las que agregan valor al estudio realizado.

1.2.3.1. Diagrama Entidad Relación

Usando el programa de fuente abierta SQL Power Architect (http://www.sqlpower.ca/page/architect) se realizó ingeniería inversa a las principales tablas del diccionario de la aplicación, para tener una perspectiva visual de su estructura, como en la figura 10 Diagrama Entidad Relación del Diccionario de Aplicación.

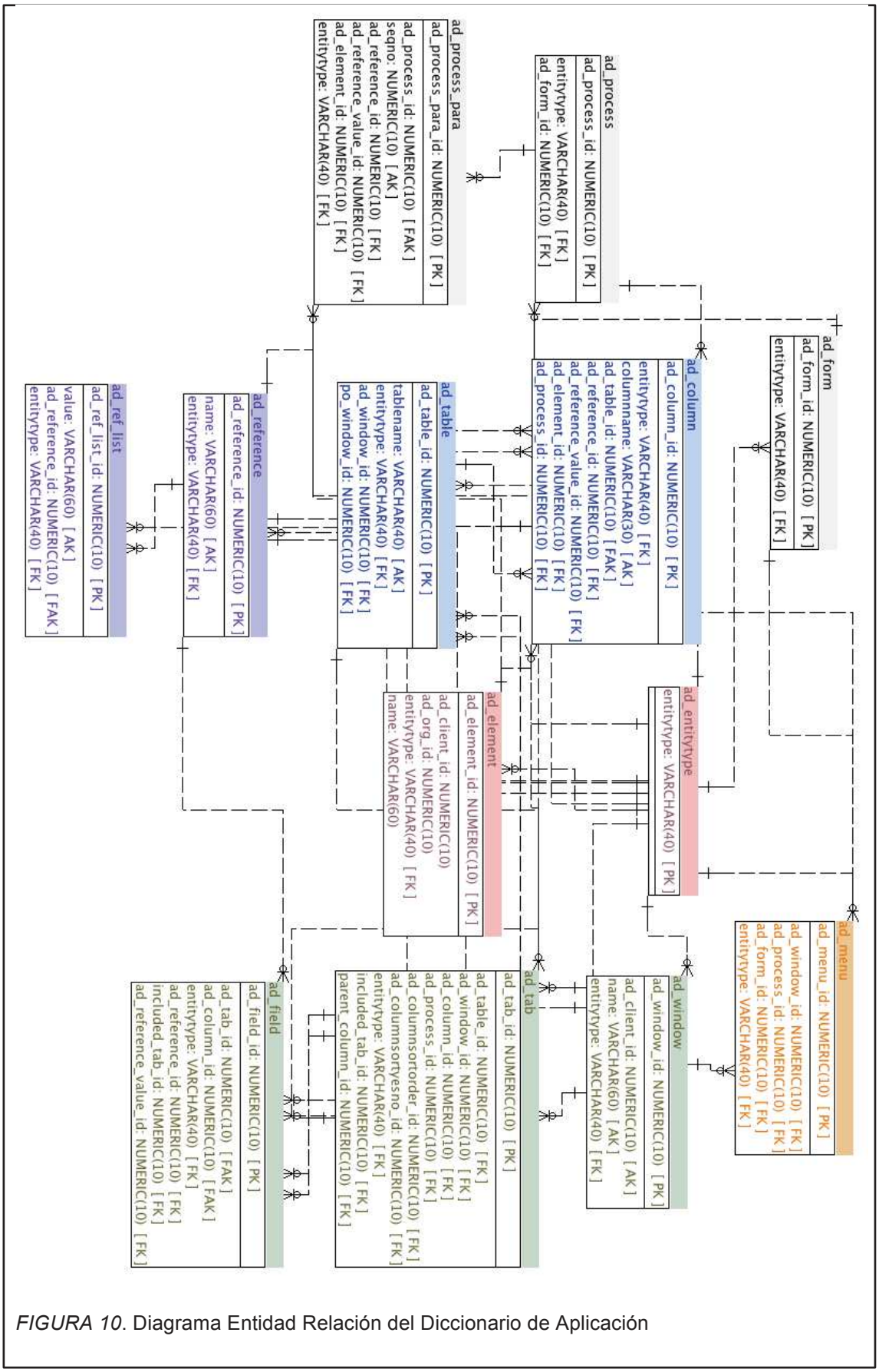


FIGURA 10. Diagrama Entidad Relación del Diccionario de Aplicación

1.2.4. Estructura de la Aplicación

Adempiere es un macro proyecto java estructurado en varias aplicaciones y paquetes. La comunidad Adempiere a adoptado a la plataforma eclipse como entorno de desarrollo para el producto, además de SVN y Mercurial como repositorios de sincronización de código fuente.

Los prerrequisitos para trabajar con el código fuente de Adempiere son los siguientes:

- Java JDK 1.6
- Base de Datos Oracle o PostgreSQL (preferiblemente PostgreSQL)
- Entorno de desarrollo Eclipse
- Plug-in SVN o Mercurial (preferiblemente Mercurial)

La forma de agregar el proyecto Adempiere al entorno de desarrollo, es importándolo directamente desde el repositorio de código fuente. La url en la que se encuentra el proyecto es: **https://adempiere.svn.sourceforge.net/svnroot/adempiere/trunk**

Una vez realizados todos los pasos indicados, se obtendrá el sistema Adempiere y todos sus sub-proyectos, como en la figura 2.9 Proyecto adempiereTrunk. (Comunidad Adempiere 2012)

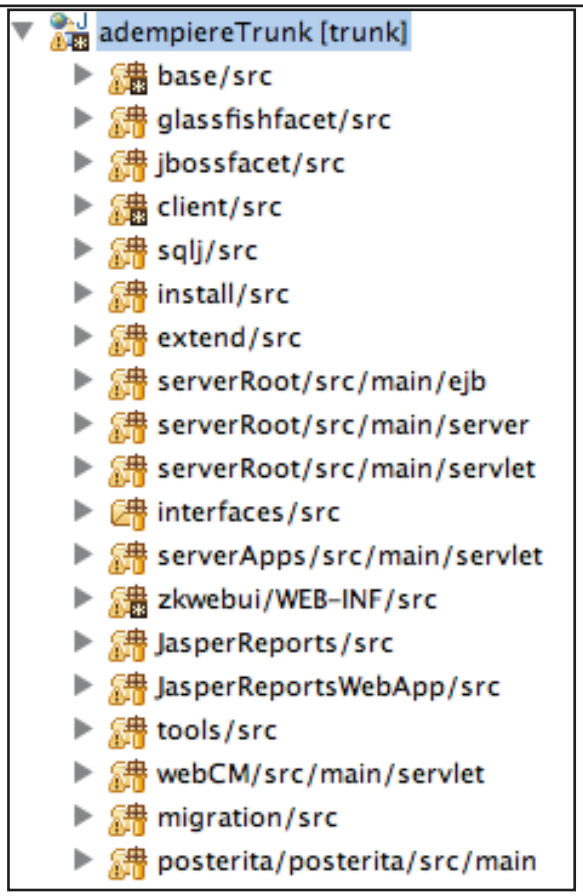


FIGURA 11. Proyecto adempiereTrunk

1.2.4.1. Objeto Entorno (Env)

Para manejar el entorno de Ejecución, Adempiere utiliza una clase Estática llamada Env, del inglés Environment.

En esta clase existen métodos como:

- isWindows()
- isMac()
- setContext()
- getCtx()
- exitEnv()

Las cuales ayudan radicalmente a la programación de nueva funcionalidad del

sistema.

En el contexto, que es también parte del entorno de Adempiere, se recrean los valores obligatorios y necesarios para trabajar con el sistema. El contexto es una clase Properties, que almacena valores como: Usuario que inició sesión, Compañía, Organización en la que se logeó, ventanas abiertas, valores de campos en caché, etc.

1.2.4.2. Persistencia y Manejo de Entidades

La persistencia del sistema se maneja en el paquete del proyecto base org.compiere.model.

Las entidades de Adempiere están compuestas por:

- **Clase X**: en la cual se tiene los constructores por defecto, setters, getters y relaciones con otros objetos.
- **Clase I**: interfaz que será implementada por X donde se definen las columnas y enumeradores de cada tabla o entidad.
- **Clase PO**: clase abstracta que implementa toda la funcionalidad de persistencia.
- **Clase M**: opcionalmente se puede crear una clase denominada M, la cual puede sobrescribir la lógica predeterminada de PO. También se la usa para extender funcionalidad.

Al crear tablas en el diccionario de aplicación se obtendrá automáticamente estas clases, corriendo las utilerías org.adempiere.util.ModelClassGenerator y org.adempiere.util.ModelInterfaceGenerator

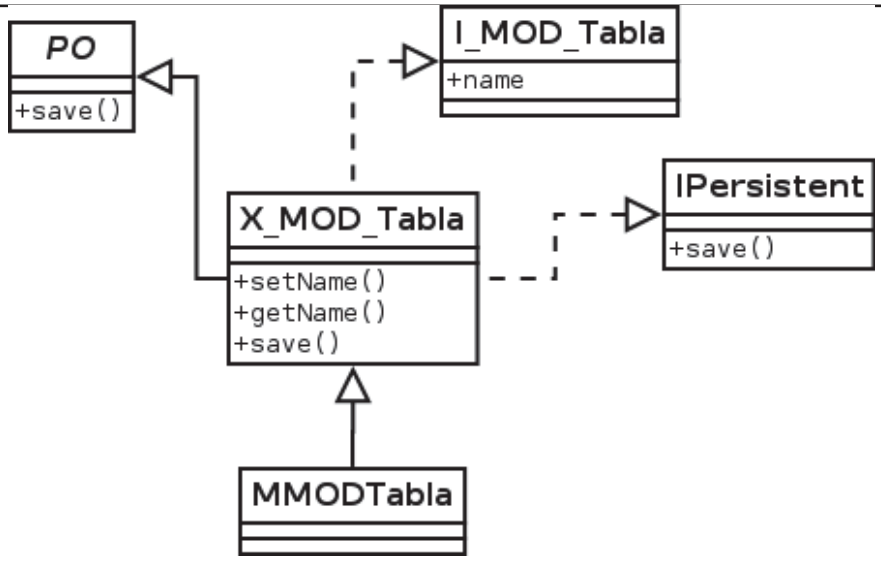


FIGURA 12, Esquema Clases Persistencia

En el ejemplo se ve el modelo de la entidad MOD_Tabla, con una única columna llamada name, previamente definida en el diccionario de aplicación.

Al crear MOD_Tabla como entidad, se puede llamar en cualquier parte del fuente de Adempiere de la siguiente manera:

```
//Creamos un objeto mtabla
MMOD_Tabla mtabla = new MMOD_Tabla(Env.getCtx(), 0, null);
//Ponemos valor a nombre
mtabla.setName("nombre ejemplo");
//Guardamos el registro
mtabla.save();
```

Si no se tiene la clase M, se puede hacer la llamada directamente desde X:

```
//Creamos un objeto xtabla
X_MOD_Tabla xtabla = new X_MOD_Tabla(Env.getCtx(),0,null);
//Ponemos valor a nombre
xtabla.setName("nombre ejemplo");
//Guardamos el registro
xtabla.save();
```

El resultado de ambas ejecuciones es la misma, se crea un registro nuevo en la tabla, se coloca un valor en el campo name y se almacena el registro. La clase

M entonces se utilizaría para implementar métodos únicos de MMOD_Tabla que se requieran en el sistema.

1.2.4.2.1. Clase Abstracta PO

La clase PO la utiliza Adempiere como motor de persistencia en todas las entidades del sistema. Se detallan los principales métodos de la clase:

TABLA 11. Métodos de PO

Método	Descripción
PO(Properties,int, String)	Constructor de un Objeto PO a partir de un ID
PO(Properties,Resulset,String)	Constructor de un Objeto PO a partir de un ResulSet
copyValues(PO,PO)	Crea un duplicado de un objeto PO
delete(boolean)	Borra un registro.
getColumnIndex(String)	Obtiene el ID de una columna.
getColumnName(int)	Obtiene el nombre de una columna a partir de un ID
getID()	Obtiene el ID del registro
get_KeyColumns()	Obtiene las columnas marcadas como clave en el diccionario de aplicación.
get_Table_ID()	Obtiene el ID de la Tabla.
get_TableName()	Obtiene el Nombre de la Tabla.
getValue(int) – getValue(String)	Obtiene el valor de una columna.
save()	Almacena el registro
afterDelete()	Métodos que se pueden sobrescribir para ejecutar un trigger
afterSave()	Métodos que se pueden sobrescribir para ejecutar un trigger
beforeDelete()	Métodos que se pueden sobrescribir para ejecutar un trigger
beforeSave()	Métodos que se pueden sobrescribir para ejecutar un trigger

La clase funciona de la siguiente manera:

- Una Entidad que hereda de PO ejecuta el constructor, pasándole por parámetro el ID del registro que se va a cargar o 0 en el caso de uno nuevo.
- Según la información de esa entidad, obtenida de la clase I, el

constructor PO inicializa un objeto POInfo, el cual carga toda la información definida en el diccionario de aplicación para esa tabla. Como resultado se tienen las propiedades de los campos, si son editables o no, longitudes, validadores, etc.

- Se establece un sistema de caché por medio de Array de Objetos, para definir valores viejos y valores nuevos.
- Se ejecuta el método load(). Que carga los valores del registro en el array de objeros viejos. Se entiende que valores nuevos está a null, hasta que se hagan modificaciones.

La problemática encontrada en este modelo nace con el hecho de tener que generar las clases I y X, ya que en todos los casos estas clases son las que se define desde el diccionario de aplicación.

Sería deseable que el motor de persistencia fuera más dinámico, que tomara la información del diccionario de aplicación y construyera el objeto de forma abstracta sin importar que tabla es la que se está utilizando. Por otro lado, la clase M es indispensable, ya que ésta implementa la lógica de negocio del sistema y si no heredara de I y X, no se podría utilizar los getter y setters de los atributos de la entidad.

Validación de Entidades

Adempiere establece que para extender la validación de una tabla, se debe sobre escribir los métodos before y after.

Sería deseable implementar un validador genérico en el cual se establezcan todas las validaciones de todas las tablas. Que sea cargado en el contexto y utilizado en el momento en el que se va a ejecutar cada acción, de esta forma se quita la dependencia de los validadores en las clases M.

1.2.4.3. Extender el Sistema Adempiere

Adempiere cuenta con un API, para extender la funcionalidad del modelo de negocio.

1.2.4.3.1. Callouts

Los CallOuts son llamadas que se ejecutan con el evento onChange de un componente de una ventana estándar de Adempiere. Previamente en el diccionario de aplicación se indica como parametrizarlo.

La clase de la cual se extiende el API para implementar CallOuts es org.compiere.model.CalloutEngine y se construyen de la siguiente forma:

```
package org.compiere.model;

import java.util.Properties;

public class testCallOut extends CalloutEngine {
  /**     Implementamos el método que hace la llamada
   *      test
   *      @param ctx -> Contexto del Sistema
   *      @param WindowNo -> Ventana que ejecuta el Callout
   *      @param mTab -> Pestaña que ejecuta el Callout
   *      @param mField -> Campo que ejecuta el Callout
   *      @param value -> Valor actuál del campo
   *      @return mensaje de error o ""
   */
  public String test(Properties ctx, int WindowNo, GridTab
mTab, GridField mField, Object value) {
    //Escribimos la lógica del callout,
    //que puede ser una validación
    //o la asignación dinámica de un dato
    //a un campo de la ventana.
    return null;
  }
}
```

1.2.4.3.2. Validadores del Modelo

Se puede modificar la lógica de negocio de un documento sobre escribiendo

los métodos extendidos de la clase DocAction en las clases M.

```
package org.compiere.model;

import java.io.File;
import java.math.BigDecimal;
import java.sql.ResultSet;
import java.util.Properties;

import org.compiere.process.DocAction;
import org.eevolution.model.X_HR_Employee;

public class MMOD_Tabla extends X_HR_Employee implements
DocAction{
  public MMOD_Tabla(Properties ctx, int MOD_Tabla_ID, String
trxName) {
    super(ctx, MOD_Tabla_ID, trxName);
  }
  public MMOD_Tabla(Properties ctx, ResultSet rs, String
trxName) {
    super(ctx, rs, trxName);
   }
  @Override
  protected boolean beforeSave(boolean newRecord) {
    //Implementamos El Trigger para beforeSave
    return super.beforeSave(newRecord);
  }
  @Override
  protected boolean afterDelete(boolean success) {
  //Implementamos El Trigger para afterDelete
    return super.afterDelete(success);
  }
  @Override
  public String completeIt() {
    // Implementamos la lógica para el documento
    return null;
  }
}
```

1.2.4.3.3. Java Triggers

Los triggers de Java se implementan en las clases M sobre escribiendo los métodos before y after.

```
package org.compiere.model;

import java.sql.ResultSet;
import java.util.Properties;
import org.eevolution.model.X_HR_Employee;

public class MMOD_Tabla extends X_HR_Employee {
  public MMOD_Tabla(Properties ctx, int MOD_Tabla_ID, String
trxName) {
    super(ctx, MOD_Tabla_ID, trxName);
  }
  public MMOD_Tabla(Properties ctx, ResultSet rs, String
trxName) {
    super(ctx, rs, trxName);
   }
  @Override
  protected boolean beforeSave(boolean newRecord) {
    //Implementamos El Trigger para beforeSave
    return super.beforeSave(newRecord);
  }
  @Override
  protected boolean afterDelete(boolean success) {
  //Implementamos El Trigger para afterDelete
    return super.afterDelete(success);
  }
}
```

1.2.4.3.4. Procesos

Los procesos son una parte muy importante en el sistema Adempiere, ya que estos implementan las acciones que ejecutan los botones y las que se ejecutan desde una tarea programada.

Los procesos reciben parámetros de ejecución que pueden ser parte de la ventana en la que el botón se ejecuta o parámetros introducidos por el ususario.

Para implementar un proceso se debe extender la clase abstracta org.compiere.process.SvrProcess, de la siguiente manera:

```
package org.compiere.process;

public class TestProcess extends SvrProcess {
  @Override
  /**
   * @return Mensaje de Salida o ""
   * */
  protected String doIt() throws Exception {
    // Ejecutamos una acción o proceso en el sistema
    return null;
  }
  @Override
  protected void prepare() {
    // Obtenemos los parámetros introducidos en la ejecución del
proceso
  }
}
```

1.2.4.3.5. Formas

La creación de formas es importante ya que desde aquí se pueden realizar ventanas personalizadas, saliéndonos del estándar de las ventanas de Adempiere.
Desde aquí se pueden implementar desde ventanas de acción hasta puntos de venta completos.

Para extender la funcionalidad de Adempiere creando un formulario personalizado, dependiendo del resultado esperado se hace de 2 maneras:

1. Creando un formulario con menú y barra de herramientas

```
package org.compiere.apps.form;

public class testFormToolBar extends FormFrame implements
FormPanel {
  @Override
  /**
   * @param WindowNo -> ventana que inicia la forma
   * @param frame -> jFrame que inician la forma
   * */
  public void init(int WindowNo, FormFrame frame) {
```

```
    getMenu(); // Obtiene el panel menú
    getToolkit(); // Obtiene la barra de herramientas
  }
  @Override
  public void dispose() {

  }
}
```

2. Creando un formulario sin menú ni barra de herramientas

```
package org.compiere.apps.form;

public class testForm implements FormPanel {
  @Override
  /**
   * @param WindowNo -> ventana que inicia la forma
   * @param frame -> jFrame que inician la forma
   * */
  public void init(int WindowNo, FormFrame frame) {

  }
  @Override
  public void dispose() {

  }
}
```

1.2.4.4. Extender Adempiere mediante Groovy

Desde la versión 3.5 de Adempiere, es posible extender la Aplicación sin necesidad de compilar el código, mediante una máquina de ejecución groovy que se integra al diccionario de aplicación.

Para ello el motor pone a disposición variables que se ejecutan en el entorno de groovy.

- **CallOut**
  - Variables de Entorno:
    - A_WindowNo
    - A_Tab

        - A_Field
        - A_Value
        - A_OldValue
        - A_Ctx
    - Ejemplo:

```
if (A_Tab.getValue("Name") != null) {
A_Tab.setValue("Description", A_Tab.getValue("Name"));
}
result = "";
```

- **Model Validator & Table Triggers**
    - Variables de Entorno:
        - A_Ctx
        - A_PO
        - A_Type
        - A_Event
    - Ejemplo:

```
A_PO.setDescription(A_PO.getName());
result = "";
```

- **Process**
    - Variables de Entorno:
        - A_Ctx
        - A_Trx
        - A_TrxName
        - A_Record_ID
        - A_AD_Client_ID
        - A_AD_User_ID
        - A_AD_PInstance_ID
        - A_Table_ID
    - Ejemplo:

```
import org.compiere.model.MTable;
import org.compiere.util.DB;
import org.compiere.util.Msg;

MTable table = new MTable (A_Ctx, P_AD_Table_ID, A_TrxName);
if (table.get_ID() == 0)
      throw new IllegalArgumentException ("No AD_Table_ID=" +
      P_AD_Table_ID);
```

```
String tableName = table.getTableName();
if (!tableName.startsWith("I"))
        throw new IllegalArgumentException ("Not an import table = " +
      tableName);
String sql = "DELETE FROM " + tableName + " WHERE AD_Client_ID=" +
A_AD_Client_ID;
int no = DB.executeUpdate(sql, A_TrxName);
A_ProcessInfo.addLog (0, null, null, "Deleted "+no+" rows from table
"+tableName);
result = "OK";
```

Aunque esto es un gran avance en Adempiere, permitiendo la personalización del sistema y ejecución de código “en caliente”, las herramientas de edición de groovy, ejecución y verificación, son muy precarias. (Comunidad Adempiere 2012)

1.2.4.5. Presentación de la Información

El motor de presentación también está vinculado al diccionario de aplicación, ya que en este se definen las ventanas y la forma en la que los campos aparecen. En el caso de Adempiere, el motor está fuertemente vinculado al GUI que el usuario ejecuta, existe uno para la interfaz swing y otro para la RIA.

Para cada una de estas interfaces, se han sobrecargado los componentes comunes de fecha, texto, combos, etcétera. Ofreciendo funcionalidad adicional a la predeterminada.

1.2.4.6. Servicios y Aplicaciones Adicionales

Reporteador:
Adempiere cuenta con un reporteador nativo del sistema elaborado en java que genera un reporte a partir de la información de la tabla obtenida del diccionario de aplicación.

Este reporteador tiene herramientas de agrupamiento, sumarizacion, organización de la información, posibilidad de crear reportes como formas, útil sobre todo para crear formatos de facturas o cheques.

Gestor de Documentos:

Adempiere también cuenta con un gestor de documentos o almacén de documentos, que permite guardar el estado de un reporte, indexándolo por los campos claves definidos en la tabla y dando la posibilidad de buscarlo.

Adjuntos:

En cada uno de los registros de Adempiere se pueden adjuntar archivos, los cuales pueden ser visualizados y descargados más adelante.

Foro:

Se cuenta con un foro para agregar comentarios adicionales a cualquier registro, sabiendo que usuario y en que fecha se realizó el texto.

1.3. Estructura Financiera y de Documentos

1.3.1. Funcionalidad Core Financiera

Adempiere maneja la parte financiera como un cubo OLAP, en el cual están definidos los lados del cubo como una dimensión que luego se utilizará para registrar una transacción contablemente. En el cubo constan todos los campos de auditoría obligatorios y aquellos que registran la información contable:

TABLA 12. Detalle de la Tabla Fact_Acct

Campo	Descripción
C_Acctschema_ID	Relación al catálogo esquema contable, para garantizar el principio de multicontabilidad.
Account_id	Relación al catalogo de cuentas contables.
dateTrx	Fecha de la transacción
dateAcct	Fecha de contabilización
C_Period_ID	Relación al catálogo de periodos contables
AD_Table_ID	Relación al catálogo de tablas, para identificar desde donde se origino la transacción
Record_ID	Identificador del registro que realizó

	la transacción
PostingType	Tipo de aplicación contable realizada. (actual o presupuesto)
C_Currency_ID	Referencia al catálogo de monedas, para garantizar el principio de multimoneda.
AmtSourceDr	Monto de origen para el débito
AmtSourceCr	Monto de origen para el crédito
AmtAcctDr	Monto debitado contablemente
AmtAcctCr	Monto acreditado contablemente
C_Uom_ID	Referencia al catálogo de unidades de medida.
Qty	Cantidad
M_Product_ID	Dimensión Producto
C_BPartner_ID	Dimensión Socio de Negocio
AD_OrgTrx_ID	Dimensión Organización de la Transacción
C_SalesRegion_ID	Dimensión Región de Ventas
C_Project_ID	Dimensión Proyecto
C_Campaing_ID	Dimensión Campaña de Mercadeo
A_Asset_ID	Dimensión Activo Fijo
User1_ID	Dimensión Personalizada 1
User2_ID	Dimensión Personalizada 2

El esquema gráfico de esta tabla muestra como un punto espacial en el cubo representa un registro contable.
Financieramente esto facilita el análisis de la información, ya que se puede hacer drill down directamente sobre los propios registros, además que simplifica la contabilidad, porque evita la creación de cuentas únicas.

El ejemplo claro de esto es Cuentas por Cobrar, según la nueva normativa del SRI, compras superiores a determinado monto no se facturan a consumidor final, sino que obligatoriamente se hace una factura a una persona natural o jurídica.

Normalmente para registrar el evento contable de esta transacción, se creaba una cuenta bajo Cuentas por Cobrar con el Nombre de la persona:

1.1.2 Cuentas por Cobrar

1.1.2.1 Antonio Cañaveral

Si el cliente no vuelve a ser atendido, esa cuenta tendrá una sola transacción formará parte del catálogo contable hasta que se realice una auditoría de cuentas, esto puede ser por varios años. Complicando el manejo Administrativo y dificultando el mantenimiento técnico.

En el caso del cubo financiero de Adempiere, este registro se realizaría directamente sobre la cuenta 1.1.2 Cuentas por Cobrar y simplemente se añadiría la dimensión Socio de Negocio (Cliente) Antonio Cañaveral al registro. Incluso se podría agregar la dimensión producto, para más adelante analizar cantidades o porcentajes de ventas, se agregaría campaña de marketing para evaluar su efectividad e incluso región de ventas para analizar qué local es más rentable. De este modo Adempiere simplifica el catálogo contable y además agrega más valor al análisis financiero.

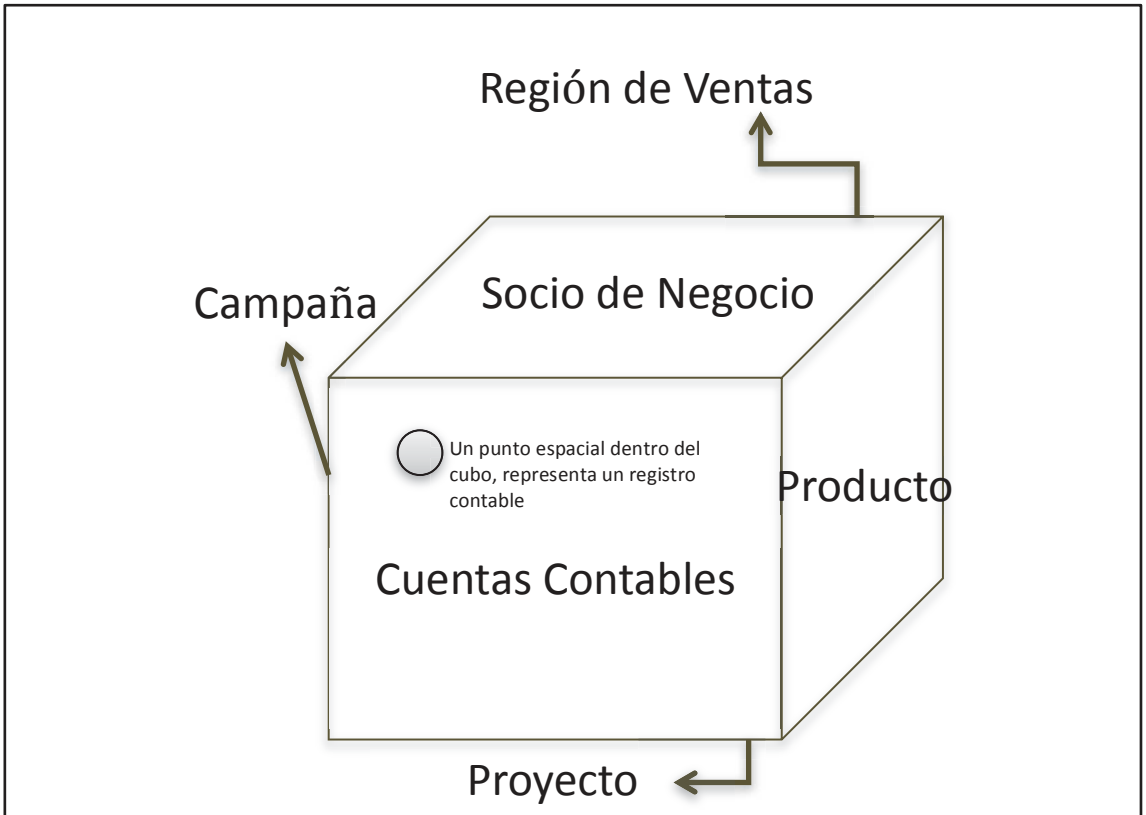


FIGURA 13, Esquema Cubo Financiero

1.3.2. Funcionalidad Core de Documentos

Parte del proceso financiero de una empresa es la gestión de los documentos que generan información financiera:

- Facturas
- Notas de Crédito
- Pagos
- Vales de Caja Chica
- Cobros
- Recibos y Entregas de Material

Para administrar estos documentos Adempiere maneja un motor de flujos para determinar los niveles de autorizaciones de los documentos. Además cada documento tiene un estado del documento y dependiendo de esto una acción disponible, a esto Adempiere lo denomina motor de documentos.
Los documentos creados en el Core de Adempiere y las clases que intervienen en su contabilización son:

TABLA 13. Documentos de Adempiere

Clase de Modelo	Descripción	Clase Financiera
MInvoice	Factura de Compra o Venta	Doc_Invoice
MPayment	Pago o Cobro	Doc_Payment
MAllocationHdr	Asignación de Pago	Doc_Allocation
MBankStatement	Cuenta Bancaria	Doc_Bank
MCash	Caja	Doc_Cash
MJournal	Diario	Doc_GLJournal
MInventory	Inventario	Doc_Inventory
MMovement	Movimiento de Inventario	Doc_Movement
MInOut	Entrada o Salida de Material	Doc_InOut
MOrder	Orden de Compra o Venta	Doc_Order
MRMA	Manufactura	Doc_Order

Entre las acciones más importantes de los documentos constan:

- **Prepare:** Procesa el documento reservando los recursos necesarios.
- **Complete:** Completa el documento agregando registros de impuestos y stock.
- **Post:** Deja el documento listo para que el motor contable lo registre.
- **Reverse_Correct:** Reversa el documento generando un transacción contable inversa.

De estas acciones se derivan los estados en los que un documento se encuentra:

- **Drafted:** Borrador
- **Processed:** Procesado
- **Completed:** Completo
- **Reversed:** Revertido

Estas definiciones se encuentran escritas en la interfaz org.compiere.process.DocAction

Con este antecedente, se puede completar el modelo de la lógica de negocio del sistema añadiendo esta interfaz a las clases M de las entidades que forman parte de un documento. Ejemplo de MInvoice:

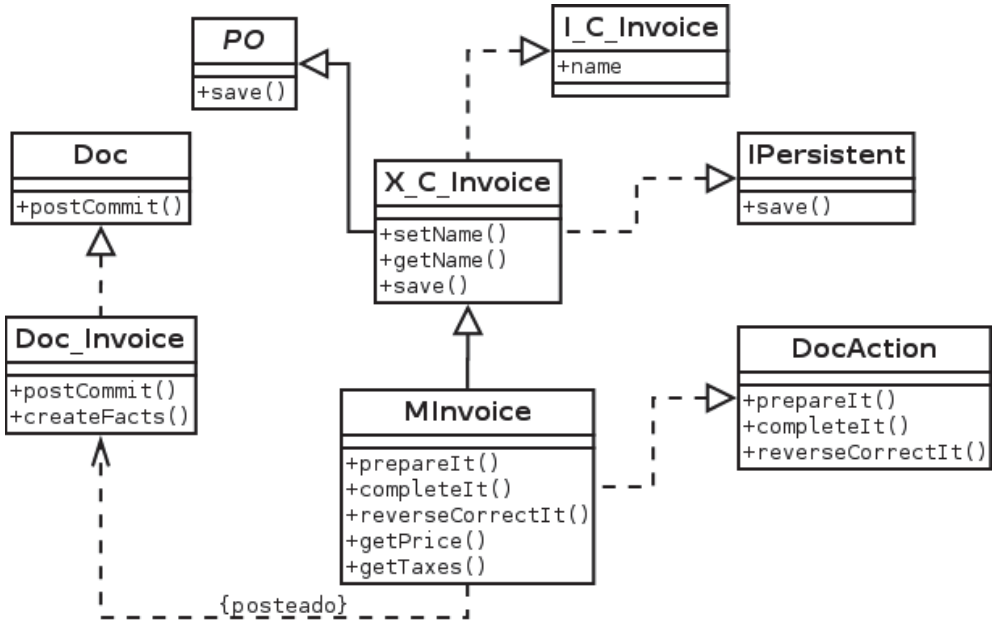


FIGURA 14, Diagrama de Clases Modelo & Documentos

CAPÍTULO 2

2. CONSTRUCCION DE LA NUEVA ARQUITECTURA

2.1. Virtudes y Falencias del Sistema Adempiere

En el estudio realizado en el capítulo 2, se encontraron más falencias que virtudes en el sistema Adempiere. Se analizará cada elemento, revisando si es una falencia, en tal caso se propondrá una mejora, sin desestimar la potencia y funcionalidad actual del sistema.

Existen un sinnúmero de implementaciones de Adempiere en todo el planeta, incluyendo varias muy importantes en Ecuador y sobre todo en el sector público. Estas implementaciones funcionan adecuadamente y no experimentan errores ni bajo performance. El único inconveniente es el duro mantenimiento.

2.1.1. Virtudes del Sistema

Es difícil establecer si el sistema nació como un Framework o directamente como un ERP. Lo que se pudo observar en el análisis de la aplicación y consultando con varios expertos financieros, fue que como ERP es una potente herramienta que permite cualquier tropicalización minimizando el desarrollo.

Desde el punto de vista técnico se encontraron estas virtudes:

- **Nomenclatura Adecuada en la Estructura de la Base de Datos:** La forma en la cual se definen las tablas en Adempiere, la nomenclatura de los índices, claves primarias, etc. Facilitan mucho el manejo de la base de datos, identificando de una forma práctica las relaciones de una entidad.
- **Diccionario de Aplicación Robusto:** La clave de la facilidad de extensibilidad de Adempiere es el robusto diccionario de aplicaciones.

Aunque tiene algunos detalles que se van a proponer en la modificación, se coloca como virtud porque en general se usará como base para el nuevo modelo.

- **Reutilización de Elementos:** Adempiere maneja la buena práctica de reutilizar elementos en el diccionario de aplicación, lo que permite minimizar las entidades a mantener.
- **Entorno :** El entorno de Adempiere facilita el desarrollo de funcionalidad, permitiendo acceder a variables desde cualquier parte del sistema.
- **Estructura Financiera Robusta:** El manejo de la contabilidad, el cubo financiero y la minimización de registros contables, hacen que este método se reutilice en la nueva arquitectura del sistema.
- **Manejo de Documentos:** Con el manejo de documentos de adempiere podemos implementar cualquier cambio que una nueva ley disponga sobre la operatividad financiera de una empresa.

2.1.2. Falencias del Sistema y Soluciones Planteadas

Para organizar mejor las falencias encontradas en el sistema, se han dividido en cuatro partes. Las de persistencia y base de datos, arquitectura de la aplicación, interfaz de usuario y generalidades que deben ser solucionadas.

2.1.2.1. Generalidades

2.1.2.1.1. Gestión de Credenciales, Log-in

Como se vio en el capítulo anterior, uno de los problemas más grandes del sistema es el manejo de credenciales y log-in de usuario.

El sistema maneja el log-in mediante una consulta a la base de datos, en la cual la contraseña se encuentra en texto plano. Esto es peligroso ya que un

ataque de inyección SQL sería posible y permitiría el acceso a un usuario mal intencionado.

Se propone reemplazar el servicio existente de LDAP, por un servicio de autorización y autenticación, JAAS (Java Authentication & Authorization Service) + Kerberos (Protocolo de Autenticación). De esta manera proporcionamos un servicio de log-in además de mantener la compatibilidad con servidores LDAP y Exchange.

Más adelante en la sección 3.1.2.3.2 Servidor de Aplicaciones Obsoleto, se estudiará la posibilidad de actualizar el servidor de aplicaciones, independientemente de eso la forma de implementar JAAS en el sistema sería la siguiente:

- Cargar la librería correspondiente al servicio de autenticación en el servidor de aplicaciones. com.sun.identity.policy.
- Crear los archivos de configuración que JAAS necesita para iniciarse. jaas-standard.cfg, jaas-tomcat.cfg, jaas-jboss.cfg.
- Implementar una interfaz que podría llamarse AuthenticationService.java, en la cual se establezcan los servicios que van a ser usados por Adempiere o por aplicaciones externas.
- Puesto que se mantendrá el manejo de los Roles tal como los hace actualmente Adempiere, los cambios a hacer en el fuente serían los siguientes:

    Después de analizar el fuente, se verifico que solamente el método getRoles de la clase org.compiere.util.Login.java es el que hace la autenticación de Adempiere. Si se cambia esta clase implementando la interfaz AuthenticationService y re-escribiendo el método para ejecutar el servicio jaas, la autenticación se gestionará por esta vía y el impacto en el cambio del código será mínimo.

```
/**
    *  Actual DB login procedure.
    *  @param app_user user
    *  @param app_pwd pwd
    *  @param force ignore pwd
    *  @return role array or null if in error.
    *  The error (NoDatabase, UserPwdError, DBLogin) is saved
    in the log
    */
   private KeyNamePair[] getRoles (String app_user, String
app_pwd, boolean force)
```

2.1.2.1.2. Manejo del Multilenguaje

Como se revisó anteriormente Adempiere complica el manejo de multi-lenguaje, creando tablas con la terminación _trl.

Se propone una capa de traducción sobre el diccionario de aplicación, para que éste presente la interfaz ya traducida, construyendo un modelo relacionado con una tabla única de traducciones que contenga la ventana, pestaña y campo que se quiere traducir, al igual que el elemento del menú.

2.1.2.1.3. WorkFlow de Documentos

Aunque las herramientas de gestión de documentos que cuenta Adempiere son óptimas para el manejo financiero, hay problemas en la parte de flujo, ya que no existe un modelador que permita dibujar el proceso, complicando las autorizaciones, validaciones y el poder adicionar información en distintas instancias del documento.

Se plantea integrar un motor BPM a la Arquitectura de Adempiere para así manejar con facilidad los procesos de negocio.

No es objeto de esta tesis evaluar la mejor herramienta BPM para que forme parte del sistema, se ha elegido a BonitaSoft, por la compatibilidad, extensibilidad y conexiones que permite.

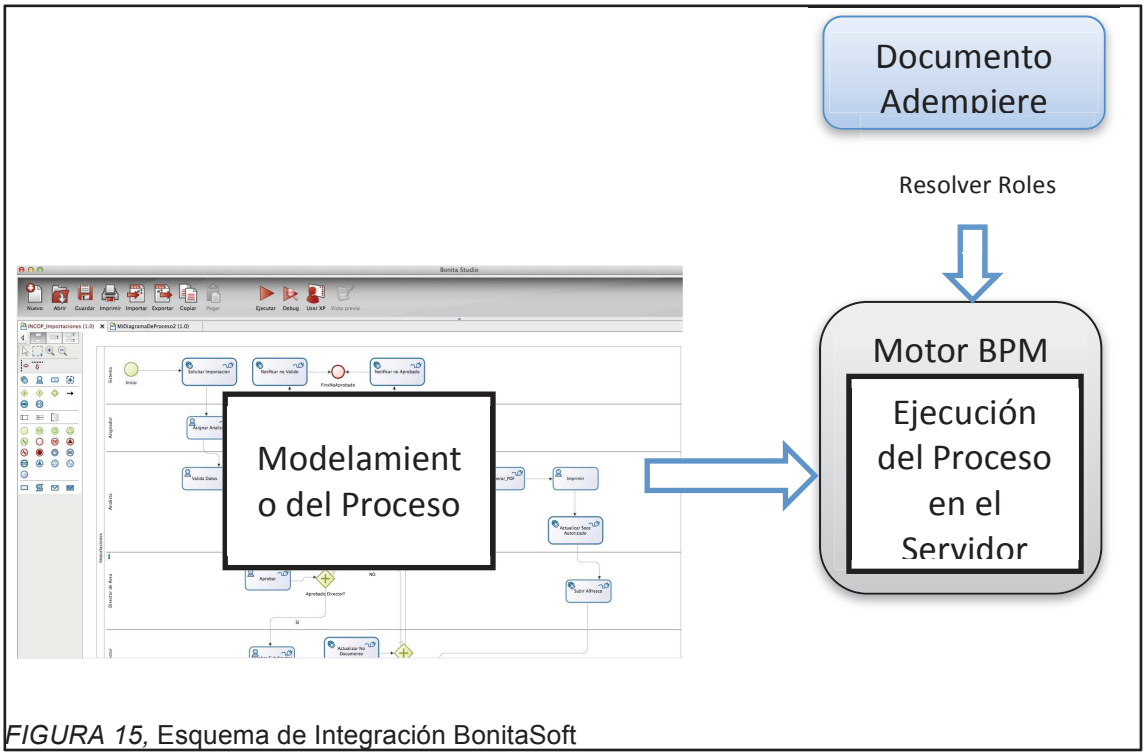


FIGURA 15, Esquema de Integración BonitaSoft

La forma de integrar la herramienta BPM es elaborar el proceso de una forma visual desde la herramienta Designer de BonitaSoft para obtener el archivo .bar del proceso, que será interpretado por el motor.

Adempiere ejecuta el flujo trasladando el documento a los aprobadores durante el proceso. El último aprobador completa el documento para su contabilización.

2.1.2.1.4. Gestión Documental

Es indispensable mantener la documentación asociada al proceso o incluso a un registro en particular. Adempiere cuenta con 2 herramientas obsoletas para ello, la posibilidad de agregar uno o varios adjuntos a un registro y almacenar el estado de un reporte en un primitivo gestor de archivos.

Estas características tienen grandes falencias, ya que no permiten buscar documentos por un índice determinado, mantener versiones de los documentos, estructurar y unificar la documentación en un archivo digital. Además que tecnológicamente sobrecargan la base de datos de Adempiere, con objetos bytea que derivan en una administración compleja de la base de

datos y altos requerimientos de disco duro.

La propuesta es conectar nativamente con un sistema de gestión documental. Para ello se debe desarrollar una estructura de índices configurable desde el diccionario de aplicación y adecuada para los documentos que se van a publicar en el sistema documental, que en principio son:

- Documentos financieros:

  Facturas, cheques, estados de cuenta, retenciones, etc.

- Archivos relacionados a un registro.

  Curriculum, fotos, contratos, etc.

El estudio se centrará en la referencia de la herramienta de gestión documental de fuente abierta más extendida y premiada en el mundo, Alfresco.

Se creará en el diccionario de aplicación la siguiente estructura de tablas para implementar la mejora:

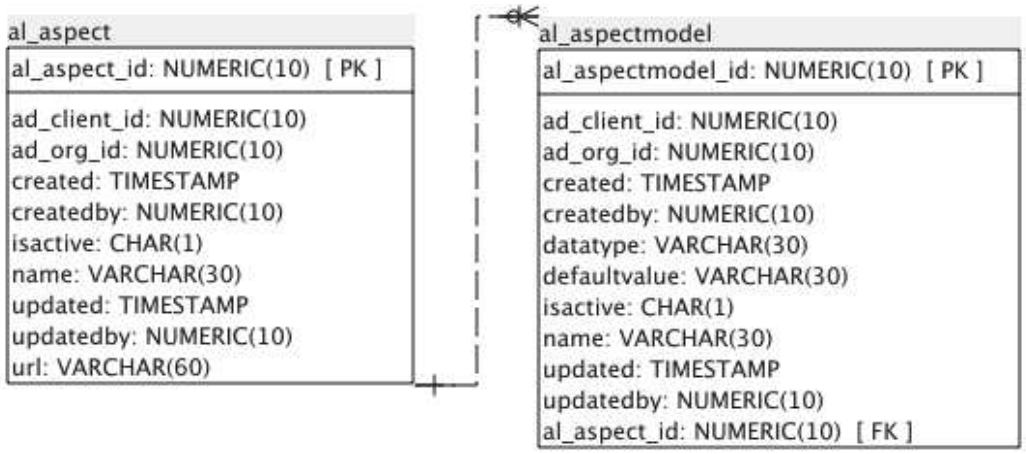


FIGURA 16, Modelo ER Conector Documental

El campo DefaultValue permitirá asignar valores a los índices a partir de las variables de contexto ubicadas en la ventana desde donde se invoca la conexión al repositorio documental.

Esta estructura permitirá crear índices personalizados para los tipos de documentos. Además es necesario implementar un proceso que genere el

modelo XML para que el servidor documental pueda replicar la estructura. Generando un archivo que será instalado en el repositorio, parecido a este:

```
<model name="al:adempieremodel" xmlns="http://www.alfresco.org/model/dictionary/1.0">
  <description>Adempiere Model</description>
  <author>Adempiere</author>
  <version>1.0</version>
 <imports>
       <!-- Import Alfresco Dictionary Definitions -->
       <import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d"/>
       <!-- Import Alfresco Content Domain Model Definitions -->
       <import uri="http://www.alfresco.org/model/content/1.0" prefix="cm"/>
 </imports>
  <namespaces>
    <namespace uri="al.adempiere.model" prefix="bs"/>
  </namespaces>
   <aspects>
    <aspect name="al:documentModel">
      <title>Creado desde Adempiere </title>
      <properties>
        <property name="al:documentNo">
          <type>d:text</type>
        </property>
        <property name="al:c_bpartner_id">
          <type>d:int</type>
        </property>
     </properties>
    </aspect>
  </aspects>
</model>
```

Se necesita desarrollar un conector para que inicie sesión a Alfresco, ingrese el documento y cree los índices asociados. Esto se desarrollará manteniendo la extensibilidad y la compatibilidad con otros motores documentales, como se muestra en la figura 17.

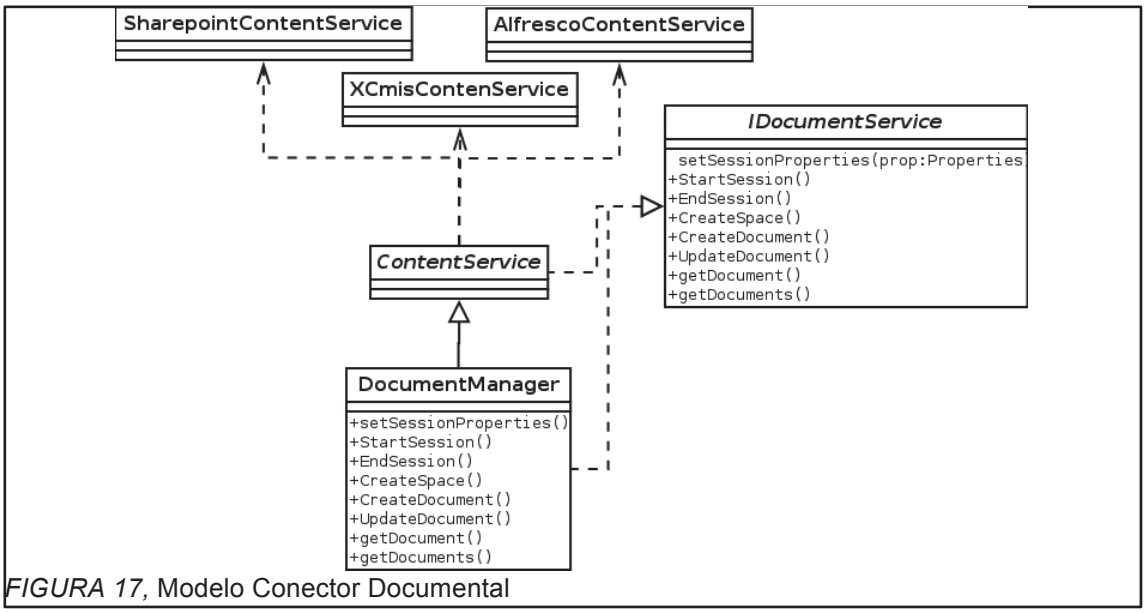


FIGURA 17, Modelo Conector Documental

La clase Content Service Implementará la funcionalidad del conector dependiendo del repositorio usado. En primera instancia se implementará la funcionalidad con Alfresco.

DocumentManager será la clase que utilizará Adempiere para gestionar el contenido en el repositorio documental, independientemente de cual sea. De esta manera simplificamos el código obteniendo líneas como las siguietes:

```
startSession();
    byte[] data = fileToByte(pathToFile);
    try {
      doc = getContent(spaceName, name);
      if (doc == null)
        doc = createDocument(spaceName, name, data);
      else
        doc = updateDocument(doc, name, data);

    } catch (Exception e) {
      e.printStackTrace();
    }
    endSession();
```

En el código, se abre una sesión al gestor documental, se crean los bytes del archivo que va a subir, se obtiene el archivo remotamente, si existe lo actualiza

en una nueva versión, si no existe lo crea. Finaliz cerrando la sesión del gestor documental.

2.1.2.2. Persistencia y Base de Datos

Como Adempiere fue un proyecto que inició a mediados del 2006, aún mantiene prácticas de persistencia comunes en esa época, que actualmente son obsoletas.

Uno de los principales contras encontrados es la dependencia a 2 marcas comerciales (PostgreSQL y Oracle). También se observa que el manejo de las sesiones de la base de datos causa problemas y requiere que el motor mantenga abiertas varias conexiones por cliente de Adempiere que esté activo.

La solución propuesta es integrar un motor de persistencia Hibernate. Se utiliza muy ampliamente y se desarrolla activamente, Hibernate también soporta una de las mayores comunidades de Open Source. Las ventajas que ofrece son claras:

- Usa el mecanismo de reflexión de java, que permite manipular un objeto en tiempo de ejecución
- Intercambiar motores de base de datos es transparente a la aplicación, simplemente modificando un archivo de configuración. Esto soluciona la portabilidad que se necesita para el sistema.
- Es más fácil de manejar que JDBC, ya que su programación es natural y orientada a objetos.

Hay 2 formas de crear persistencia con Hibernate, de manera estática, mapeando las entidades por medio de un archivo xml y dinámicamente por medio de Anotaciónes, funcionalidad implementada desde java 1.6.

Para implementar este cambio, se deben reconstruir todas las Entidades y utilizar Anotaciónes, ya que al crearse de forma dinámica las entidades, no es viable mapear cientos de tablas.

Pasos para implementar Hibernate:

Entidades:

```
@Entity
public class C_Invoice{
   @Id
   @GeneratedValue
   private Integer C_Invoice_ID;
```

Se observa que se trata de una clase normal, donde el la primera línea se anota que se trata de una entidad y luego se indican las propiedades de los atributos mapeados directamente con la base de datos.

DAO:

```
public interface Dao {
          public void persist(Object entity);
          public void persist(Object[] entities);
          public <T> List<T> find(Class<T> entityClass);
          public <T> T load(Class<T> entityClass, Serializable id);
          public <T> List<T> find(String hql);
}
```

El Data Access Object será la interfaz que resida la lógica de manejo de Hibernate esto facilitará la independencia del propio Hibernate, por si más adelante se quiere implementar otro motor de persistencia.

```
@Repository
   public class SpringHibernateDao extends HibernateDaoSupport implements Dao {

      @Autowired
      public SpringHibernateDao(SessionFactory sessionFactory) {
         super.setSessionFactory(sessionFactory);
      }

      @Transactional
      public void persist(Object entity) {
         getHibernateTemplate().saveOrUpdate(entity);
      }

      @Transactional
      public void persist(Object[] entities) {
         for (int i = 0; i < entities.length; i++) {
            persist(entities[i]);
         }
      }

      @Transactional(readOnly = true)
      public <T> List<T> find(Class<T> entityClass) {
         final List<T> entities = getHibernateTemplate().loadAll(entityClass);
```

```
        return entities;
    }

    @Transactional(readOnly = true)
    public <T> T load(Class<T> entityClass, Serializable id) {
        final T entity = (T)getHibernateTemplate().load(entityClass, id);
        return entity;
    }

    @Transactional(readOnly = true)
    public <T> List<T> find(String hql) {
        final List<T> entities = getHibernateTemplate().find(hql);
        return entities;
    }

}
```

Este ejemplo de código de la clase de SpringHibernateDao, indica cómo se puede refactorizar la clase PO estudiada de Adempiere, para que todas las Entidades hereden funcionalidad de persistencia.

La anotación @Repository indicará a Spring que la clase es de persistencia, convirtiéndola en un Singleton. Es decir a diferencia de la clase PO que se creaba en cada llamada de una entidad del modelo de Adempiere, esta nueva clase se instanciará una sola vez creando tantos Threads como llamadas tenga. Optimizando fuertemente el rendimiento, las conexiones y el uso de memoria.

La etiqueta @Autowired le indica a Spring que cuando vaya a crear la instancia de HibernateDaoSupport, le inyecte la dependecia al SessionFactory que se configura mediante un XML en Hibernate.
La anotación @Transactional Indica a Spring que el método es transaccional. Spring verificará si existe una transacción abierta, si existe une la nueva transacción al hilo en ejecución, si no existe creará una nueva transacción. De esta manera se asegura que toda operación dentro de base de datos se realiza dentro de una transacción y el respectivo RollBack si ocurriera algún error de Runtime.

Capa de Negocio:

Más adelante cuando se integre toda la nueva arquitectura propuesta, se indicará que esta capa es la más importante para elevar Adempiere a los estándares SOA. El homónimo actual de Adempiere con esta capa, son las clases M del modelo.

```
@Service
public class MInvoice {

   @Resource
   private Dao dao;

   public C_Invoice newMInvoice() {
      return new C_Invoice();
   }

   public void persist(C_Invoice invoice) {
      dao.persist(invoice);
   }

   public List< C_Invoice > getInvoices() {
      final List< C_Invoice > list = dao.find(C_Invoice.class);
      return list;
   }
}
```

@Service es una anotación de Spring que indica que es una clase de servicio y que debe ser un Singleton.

La anotación @Resource indica que se debe inyectar la dependencia de la instancia Dao creada anteriormente con @Repository.

García, A. Spring + Hibernate + Anotaciones = Desarrollo Rápido en Java. Adictosaltrabajo (e.f.)

2.1.2.2.1. Funciones de Base de Datos

Otro de los inconvenientes reportados en el estudio de Adempiere del capítulo anterior es la cantidad de funciones de base de datos que maneja el sistema.

Se listan las funciones de la base de datos y la viabilidad de migrarlas a lenguaje java junto con hibernate y su nivel de complejidad:

TABLA 14 Funciones de la Base de Datos

Función	Descripción	Migrable	Complejidad
adddays	Añadir Días	Si	Baja
acctbalance	Contabilizar Balance	Si	Media
accttotalbalance	Totalizar Balance	Si	Media
add_missing_translations	Corrección de Traducciones	No	--
add_months	Añade meses a una fecha	Si	Baja
altercolumn	Alter column para el diccionario de datos	Si	Alta
bompricelimit	Modifica el precio límite al actualizar una factura	Si	Media
bompricelist	Modifica el precio de lista al actualizar una factura	Si	Media
bompricestd	Modifica el precio actual al actualizar una factura	Si	Media
bomqtyavailable	Modifica el disponible al actualizar una factura	Si	Baja
bomqtyavailableasi	Modifica el disponible al actualizar una factura	Si	Baja
daysbetween	Calcula los días entre dos fechas	Si	Baja
bomqtyonhand	Calcula Stock de un producto	Si	Alta
bpartnerremitlocation	Devuelve al ID de la dirección por defecto de un Socio de Negocio.	Si	Baja
bomqtyordered	Calcula el Stock Ordenado	Si	Baja
charat	Devuelve el carácter de una posición dada.	Si	Baja
bomqtyreserved	Calcula el Stock reservado	Si	Media
count_full_weekend_days	Devuelve el número de fines de semana entre 2 fechas.	Si	Media
currencybase	Devuelve el ID de la moneda definida como Base	Si	Baja
currencyrate	Devuelve la taza de cambio de la moneda base	Si	Baja
currencyconvert	Convertibilidad de Cambio de Monedas	Si	Baja
currencyround	Redondeo de convertibilidad	Si	Baja
documentno	Devuelve el siguiente	Si	Media

	número de documento.		
firstof	Devuelve el primer registro.	Si	Media
getdate	Devuelve la fecha del periodo	Si	Baja
infowithseparator	Devuelve un conjunto de registros en Modo lista	No	--
instr	Función para operar con Cadenas	Si	Baja
invoicediscount	Calcula el descuento de una factura	Si	Alta
invoiceopen	Calcula la contabilidad abierta de una factura	Si	Alta
invoiceopentodate	Calcula la fecha de la apertura de una factura	Si	Alta
invoicepaid	Paga una factura	Si	Alta
invoicepaidtodate	Comprueba la fecha de pago de una factura.	Si	Media
isoncomission	Verifica si una transacción está en comisión	Si	Media
isonvacation	Verifica si un recurso está de vacaciones.	Si	Media
nextidfunc	Devuelve el siguiente ID de una función	Si	Baja
migr_fix_payment_cashline	Concilia los pagos con el flujo de caja.	Si	Baja
nextbusinessdate	Siguiente fecha laborable.	Si	Baja
nextbusinessday	Siguiente día laborable.	Si	Baja
trunc	Trunca una cadena	Si	Baja
nextid	Devuelve el siguiente ID para un registro.	Si	Media
nextidbyyear	Siguiente ID anual para un registro contable.	Si	Alta
paymentallocated	Reserva un pago	Si	Media
paymentavailable	Libera un pago	Si	Media
paymenttermdiscount	Obtiene los métodos de descuento de un pago.	Si	Media
paymenttermduedate	Obtiene la fecha de un término de pago.	Si	Media
paymenttermduedays	Obtiene los días restantes de un término de pago.	Si	Media
productattribute	Obtiene los atributos de un producto.	Si	Media
returndate	Obtiene la fecha de	Si	Baja

	devolución de un producto.		
role_access_update	Actualiza los permisos de un Rol.	Si	Media
round	Redondea un valor.	Si	Baja
subtractdays	Extrae días.	Si	Media
update_sequences	Actualiza las secuencias de los documentos.	No	--

Nota: Adaptado del Modelo de Adempiere.

Teniendo un total de 57 funciones, 3 no son migrables principalmente porque no son compatibles con el nuevo modelo estructurado con Hibernate, 8 tienen complejidad alta y hay 23 medias y 23 bajas.

Si se asignan 8 horas para resolver las altas, 6 horas las medias y 3 horas las bajas. Se llega a un total de 271 horas de trabajo para migrar las 54 funciones.

Cabe destacar que el 50 % de las funciones analizadas se realizaron en Adempiere para suplir necesidades que el entorno java no solventaba en esa época.

2.1.2.2.2. Estructura de Tablas y Catálogos

Aunque se ha mostrado como ventaja la buena práctica de Adempiere para nombrar a tablas y columnas, se han identificado un buen número de tablas repetitivas o que al ser de catálogos tienen no más de diez o quince registros.

La propuesta es crear una sola tabla de catálogos, indexada y filtrada, para disminuir considerablemente el número de tablas en la base de datos. Además crear un servicio en el core que devuelva fácilmente los valores de estos catálogos.

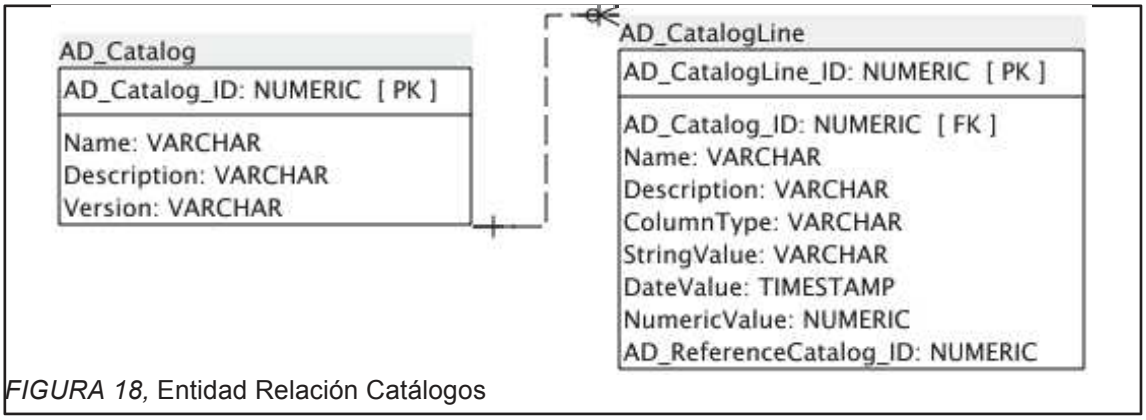


FIGURA 18, Entidad Relación Catálogos

TABLA 15. Catálogo

Campo	Descripción
AD_Catalog_ID	Identificador de la tabla
Name	Nombre del catálogo
Description	Descripción del catálogo
Version	Versión del Catálogo

TABLA 16. ítem del Catálogo

Campo	Descripción
AD_CatalogLine_ID	Identificador de la tabla
AD_Catalog_ID	Referencia a Catálogo
Name	Nombre del Valor
Description	Descripción del Valor
ColumnType	Tipo de Columna, String, Date o Number
StringValue	Valor cuando es texto
DateValue	Valor cuando es fecha
NumericValue	Valor cuando es número
AD_ReferenceCatalog_ID	Referencia opcional a un subcatálogo cuando se establece un valor.

2.1.2.2.3. Auditoría Desacoplada a las Tablas

Como se analizó en el punto 1.1.2 existen siete campos obligatorios en todas las tablas del sistema para auditar el usuario que creó el registro, la fecha de creación, el último usuario que actualizo el registro y la última fecha de actualización.

Esta estructura complica el manejo de las tablas, además que crea problemas de ejecución si se crea una entidad sin estos campos.

Se propone englobar la auditoría del sistema en un módulo independiente desacoplado de las tablas de Adempiere. Para ello se crea una estructura de datos que manejará la auditoría de cambio de cada campo, almacenando información indexada por tabla de Adempiere y fecha del evento. También se añadirá en la persistencia el manejo automático de la auditoría en las clases que ejecuten las operaciones CRUD.

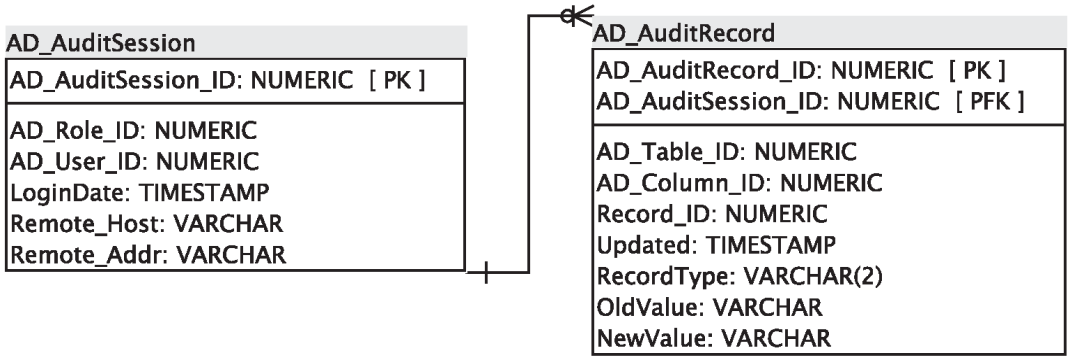


FIGURA 19, Entidad Relación Módulo Auditoría

TABLA 17. Auditoría de Sesión

Campo	Descripción
AD_AuditSession_ID	Identificador de la Tabla
AD_Role_ID	Rol con el cual el usuario inició sesión
AD_User_ID	Usuario que inició sesión
LoginDate	Fecha & Hora de Ingreso
Remote_Host	Nombre de la máquina remota
Remote_Addr	IP de la máquina remota

TABLA 18. Auditoría de Tabla y Registro

Campo	Descripción
AD_AuditRecord_ID	Identificador de la Tabla
AD_AuditSession_ID	Clave foránea a la tabla AD_AuditSession
AD_Table_ID	Tabla que se hizo el cambio
AD_Column_ID	Columna a la que se hizo el cambio
Record_ID	Registro del evento
Updated	Fecha del evento
RecordType	Típo de evento, insert, update, delete
OldValue	Valor anterior
NewValue	Nuevo valor

2.1.2.3. Código y Arquitectura

2.1.2.3.1. Reorganización del Código Fuente

La construcción del sistema se realizó de una forma muy acelerada y desorganizada, por ello es indispensable refactorizar el código fuente creando paquetes y directorios coherentes que faciliten el futuro mantenimiento del sistema.

2.1.2.3.2. Servidor de Aplicaciones Obsoleto

Adempiere depende de un servidor de aplicaciones embebido en el sistema Jboss 4.5 el cual tiene varios problemas de rendimiento y seguridad que las nuevas versiones han solventado. Este motivo ha sido uno de principales factores por los que no se han implementado herramientas más avanzadas para actualizar la arquitectura de Adempiere. Actualmente la última versión de Jboss es 7.1

La propuesta es crear aplicaciones estándar y stand-alone, para que no dependan directamente de las librerías del servidor. De esta forma el sistema se podrá desplegar en las últimas versiones de servidores Jboss o Tomcat.

2.1.2.3.3. Manejo de Entidades con Inyección de Dependencias

El propósito de la inyección de dependencias es que los módulos de software sean independientes comunicándose únicamente a través de una interface(clase).
El uso de implementaciones y la eliminación de instanciaciones de objetos mediante la instrucción new facilitan el manejo de las entidades del sistema, además que permiten la integración de nuevas tecnologías como groovy o WebServices

En el capítulo 3.1.2.2 se describen las clases de persistencia con inyección de

dependencia. En la figura 3.5 Modelo de Entidades y Persistencia con Inyección de Dependencias se propone la estructura de clases para el nuevo modelo de manejo de entidades.

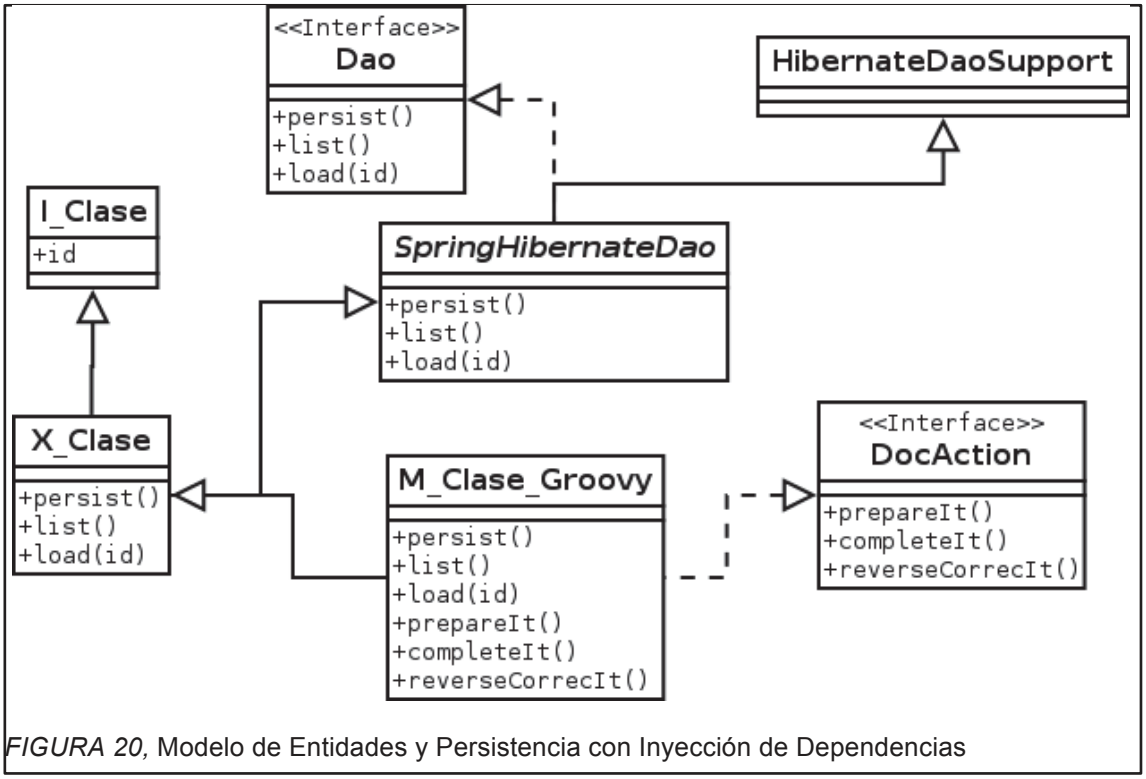


FIGURA 20, Modelo de Entidades y Persistencia con Inyección de Dependencias

El objetivo de este modelo es crear una arquitectura en la cual la clase SpringHibernateDao sirve de motor de persistencia, se instancia una sola vez optimizando recursos, ya que es un Singleton @Repository.

También es una propuesta del modelo migrar las clases M de Adempiere a Groovy, para poder modificar la lógica de negocio en tiempo de ejecución, entonces si se quisiera cambiar las operaciones que hace una factura al momento de completarse (Operación que realiza un documento de Adempiere previo a su contabilización), simplemente se sobrescriben los métodos heredados de DocAction. Esto se analizará en el punto 2.1.2.3.4

2.1.2.3.4. Integrar en la Arquitectura el Motor Groovy

Para lograr un sistema en el cual el mantenimiento del mismo sea fácil, rápido y

sin tener la necesidad de suspender el servicio, es imprescindible la utilización de lenguajes de scripting que compilen los cambios en tiempo de ejecución.

Groovy proporciona herramientas para que el testeo, desarrollo y compilación del código se hagan de una manera óptima y estática.

Se propone una fuerte integración de Groovy en el sistema, principalmente para la validación del modelo. Además proporcionar herramientas para que mediante groovy se realicen validaciones en el diccionario de aplicaciones y en la interfaz gráfica.

Una de las características a implementar será un repositorio groovy para acceder a las clases M, desde las que se realizarán cambios a la lógica de negocio directamente desde el fuente y sin necesidad de compilar.

Las clases M implementadas en groovy tendrán una estructura como la siguiente:

```
import org.bmlaurus.document.DocAction;
import org.bmlaurus.model.SpringHibernateDao;
import org.bmlaurus.webservice.WebServiceModelResolver;

@Service
public class M_Clase_Groovy extends X_Clase implements DocAction,WebServiceModelResolver
{
	@Resource
	private ClaseCustom recurso;
	//Recurso para inyección en otras clases.

	public M_Clase_Groovy(int Record_ID){
	//Constructor clase M;
	}

	public void prepareIt(){
	//Implementación del documento de Adempiere
	}

	public Object getObject(int clientID, String username, String password){
	//Implementación del Servicio.
	}

	public void metodoPersonalizado(){
	//Metodo Personalizado desde Groovy
	}
```

}

En cuanto a las validaciones del modelo de negocio se propone una herramienta visual para la escritura de código en un ambiente amigable y con el entorno necesario para realizar testeo de métodos. Se propone además integrar nativamente las herramientas groovy para extender Adempiere, como se revisó en el punto 1.2.4.4.

2.1.2.3.5. Implementación de Servicios, API y Conectores

Además de los servicios nativos descritos anteriormente, se propone generar un API REST (Representational State Transfer), que es una técnica de arquitectura de software para sistemas distribuidos, basado en los siguientes diseños fundamentales:

Protocolo Cliente/Servidor sin estado:
Cada mensaje HTTP contiene toda la información necesaria para comprender la petición, ni el cliente ni el servidor necesitan recordad ningún estado de las comunicaciones entre mensajes.

Operaciones Bien Definidas:
Aparte de las operaciones básicas de HTTP, GET, POST, PUT, DELETE.

Sintaxis Universal:
Para identificar los recursos únicamente a través de su URI.

Uso de Hipermedios:
Tanto para la información de la aplicación como las transacciones de estado REST utiliza HTML o XML y así poder navegar desde un recurso REST a otro con facilidad.

La implementación del API se realizará como un proyecto separado y se desplegará en conjunto con Adempiere en el mismo servidor, de esta manera

se puede realizar este paso incluso antes de migrar el resto de la aplicación. (Blog sobre SOA, 2011)

El API tendrá una estructura como la siguiente:

APIAccesor:

- CommandAPI: Para permitir ejecutar comandos en el entorno del sistema.
- IdentityAPI: Administración de usuarios y roles.
- ManagementAPI: Comandos de administración de Adempiere.
- RuntimeAPI: Métodos para trabajar con el ERP, crear documentos, crear registros, completar transacciones, etc.

QueryAPI:

- QueryDefinitionAPI: Obtener información de un documento, registros, transacciónes, etc.
- QueryRuntimeAPI: Obtener información en tiempo de ejecución, usuarios involucrados en un proceso, estado actual de un documento o archivo.
- RESTAccess: Obtiene un Proxy implementando el REST API.

Cada uno de estos elementos se definirán en el diccionario de aplicaciones, obteniendo un API dinámica, según los requerimientos de cada implementación de la aplicación.

Se propone también presentar servicios (WebService) de forma nativa. Esto facilitará la obtención o consulta de información.
La capa de servicios estará integrada en el modelo de entidades, acompañado por un motor de servicios que será creado en el diccionario de la aplicación, permitiendo crear y parametrizar los servicios que se publicarán, además esto permitirá definir los consumidores de servicios externos en el propio diccionario.

En la figura 21 se establece una interfaz WebServiceModelResolver, que será inyectada en las clases Groovy, permitiendo generar los servicios requeridos de manera fácil y en tiempo de ejecución, obteniendo un sistema totalmente orientado a arquitectura SOA.

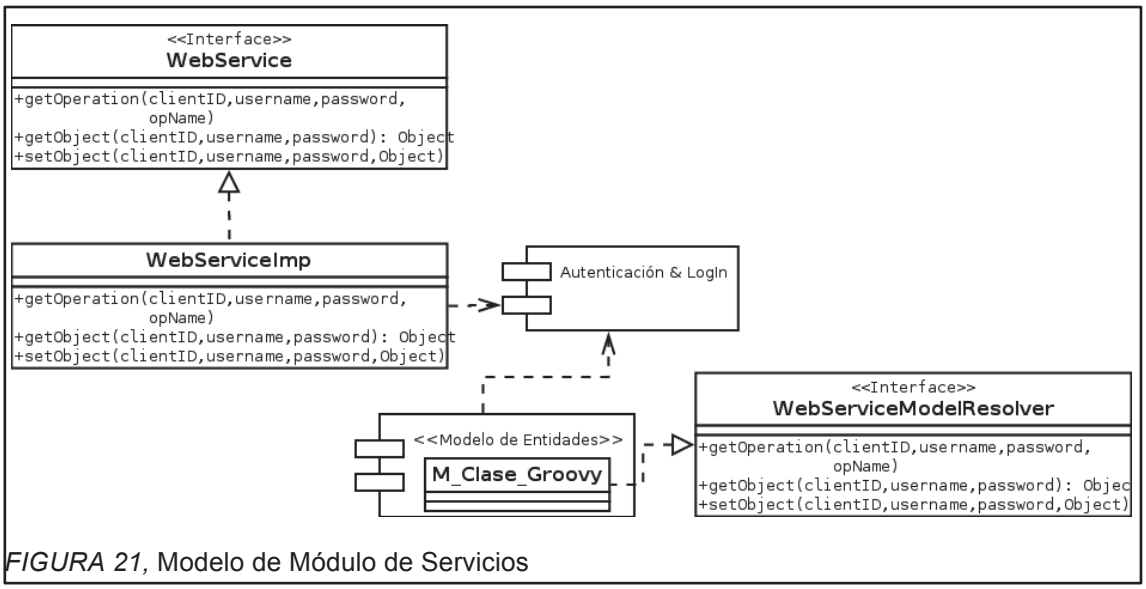


FIGURA 21, Modelo de Módulo de Servicios

2.1.2.4. Interfaz de Usuario

2.1.2.4.1. Independencia de la Interfaz de Usuario

Otro de los problemas encontrados es que la interfaz de Adempiere está integrada en el core. Al aplicar todos los cambios planteados en la arquitectura se pone a disposición todas las herramientas necesarias para desarrollar interfaces de usuario en cualquier arquitectura o plataforma. Es el primer paso para elaborar aplicaciones para nuevos dispositivos como tabletas digitales o teléfonos inteligentes.

2.1.2.4.2. Toolkit de Herramientas en Formas

Al revisar la opción de generar formas personalizadas se notó que para extender Adempiere por este medio se requería prácticamente desarrollar de forma integra la pantalla y su contenido.

La propuesta es elaborar un toolkit de desarrollo bajo el entorno de Eclipse, para crear formas personalizadas y luego poder desplegarlas en el servidor en tiempo de ejecución.

2.2. Arquitectura Propuesta para el Sistema Adempiere

2.2.1. Esquema de la Arquitectura Propuesta

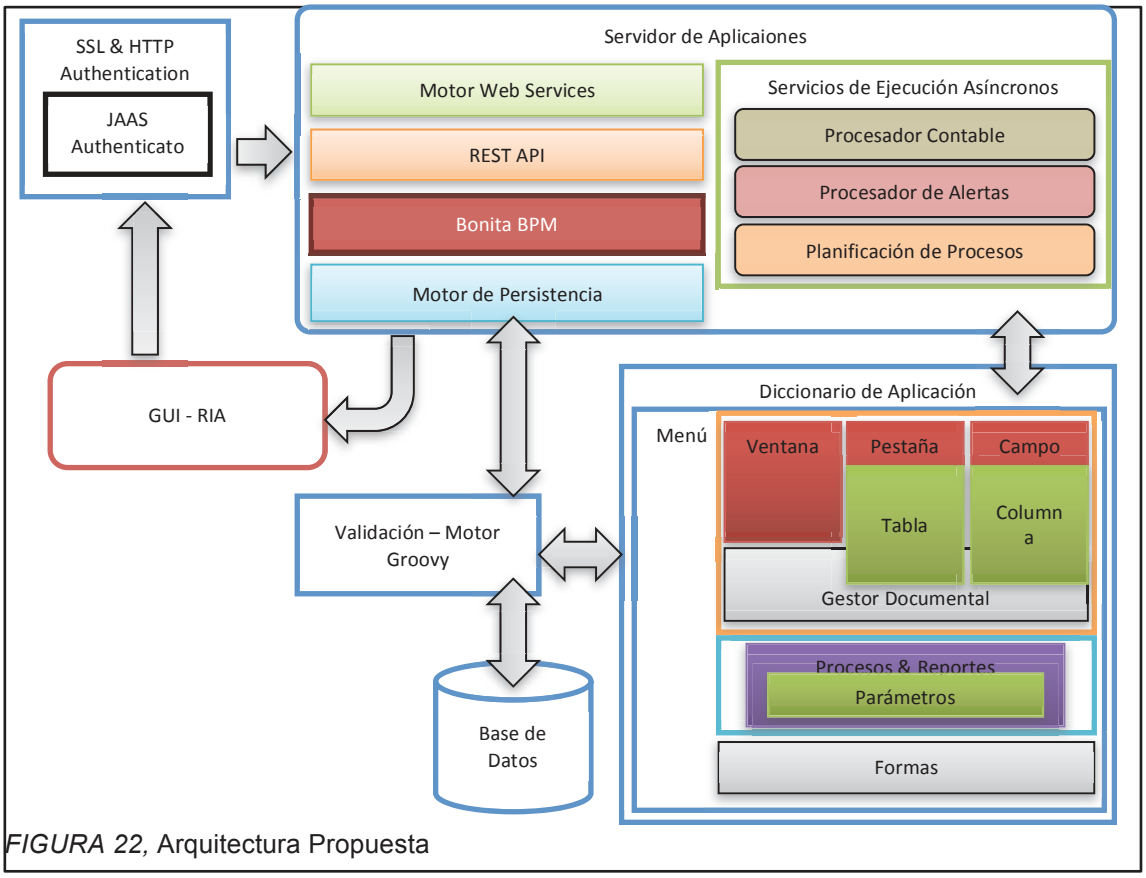


FIGURA 22, Arquitectura Propuesta

Se describe a continuación cada módulo del sistema esquematizado en la figura 22,Arquitectura Propuesta.

- **Módulo de Autenticación:** Punto de entrada de la aplicación, valida las credenciales de los distintos accesos al sistema.
- **Servidor de Aplicaciones:** Gestiona los servicios del sistema, administrando el motor de WebServices, aplicaciones REST API, aplicación BPM y plataforma Spring Framework. Además del servicio de ejecución asíncronos.
- **Diccionario de Aplicación:** Mantiene la Estructura estudiada en el

capítulo 1, agregando toda la funcionalidad del gestor documental, elementos de traducción, auditoría y maquina de groovy descritas en este capítulo.

- **Motor Groovy:** Última capa del modelo, para obtener la posibilidad de modificar el código en tiempo de ejecución. Validadores y Herramientas para extender el código de Adempiere.
- **GUI-RIA:** Interfaz enriquecida que implementará clases @Controller @Scope de la capa de control de Spring.
    - @Controller es una anotación que le indica al framework que se trata de una clase de la capa de control. @Scope sobre escribe el comportamiento por defecto de Spring y permite crear instancias distintas por cada sesión http.
- **Base de Datos:** El servidor de base de datos es transparente a la aplicación por la capa de datos implementada sobre Hibernate, esto facilita la integración con cualquier motor existente en el mercado.

CAPÍTULO 3

3. ESTRATEGIA DE INTEGRACIÓN DE LA NUEVA ARQUITECTURA Y MODELOS DE PRUEBA

3.1. Elementos a integrar del ERP Adempiere

En los capítulos 1 y 2 se han analizado a profundidad los elementos del sistema Adempiere, poniendo énfasis en aquellos que se cambiarán con la nueva arquitectura propuesta. También se analizaron las bondades funcionales y cómo estas son las que convierten a Adempiere en el mejor ERP Open Source del mercado.

Desde el punto de vista funcional, se listan los elementos a mantener en el sistema Adempiere:

- Estructura del sistema basado en procesos de negocio en lugar de módulos.
- Multifuncionalidad de todos los elementos del sistema.
- Cubo contable de análisis financiero tipo OLAP.
- Funcionalidad y manejo de documentos que generan información financiera.
- Manejo de Cuentas por Pagar, Cuentas por Cobrar, Órdenes de Compra/Venta, Inventarios, Activos Fijos y Recursos Humanos.

Junto con el análisis de los elementos técnicos a recuperar de la arquitectura anterior, es necesario evaluar la brecha, estimando el tiempo y costo requerido para la construcción de la nueva arquitectura.

En la tabla 19 se listan los elementos analizados en esta investigación, identificando el estado anterior y el propuesto para cada uno de ellos.

TABLA 19. Análisis de Brechas y Costos

Elementos Anteriores	Nuevos Elementos	Brecha (%)	Horas	Costo*	Brecha (Complejidad)
Estándares ERP					
Nomenclatura de Base de datos	Se mantiene	0%	0	$-	0
Código Fuente Desordenado	Reorganización del Código Fuente	40%	16	$720,00	2,5
Tablas Multilenguaje	Capa de Traducción	50%	40	$1.800,00	7,8
Auditoría Incrustada en las tablas	Módulo de Auditoría	60%	60	$2.700,00	14,1
Persistencia					
Objeto PO	Persistencia con Hibernate	100%	240	$10.800,00	93,8
Funciones de Base de Datos	Funciones Java	50%	271	$12.195,00	52,9
Entidades estáticas	Entidades e Inyección de Dependencias	80%	320	$14.400,00	100
Herramientas, Utilerías e Interfaz					
Log-In & Credenciales	Kerberos + JAAS	80%	80	$3.600,00	25
WorkFlow de Documentos	Integración Bonitasoft	100%	80	$3.600,00	31,3
Adjuntos y gestor de Archivos	Integración Alfresco	100%	40	$1.800,00	15,6
Interfaz Obsoleta	Independencia de Interfaz de Usuario	80%	320	$14.400,00	100
Servidor de Aplicaciones Obsoleto	Independencia, aplicaciones stand-alone	70%	40	$1.800,00	10,9
Extensión de funciones Groovy	Motor Groovy Integrado	100%	160	$7.200,00	62,5
Core					
Diccionario de Aplicación	Cambios para integración de nuevos componentes	30%	80	$3.600,00	9,4
Nueva Funcionalidad					
--	Gestor de Catálogos	0%	40	$1.800,00	0
--	API REST	0%	160	$7.200,00	0
--	WebServices Integrados	0%	80	$3.600,00	0
--	Toolkit para diseño de interfaz	0%	300	$13.500,00	0
Promedio de Complejidad					**29,2**
Costo Total del Proyecto				**$104.715,00**	

Nota: La columna Costo se calcula bajo la referencia de $45 la hora.

La nueva funcionalidad que se implementará en el sistema es clave para el proceso de re-diseño de la arquitectura, el API REST y los WebServices integrados presentarán las interfaces necesarias para interconexión con otros sistemas o entornos, incluso facilitarán el desarrollo de la nueva funcionalidad.

3.2. Estrategia de integración de la nueva arquitectura

Analizada la brecha y los costos de los elementos a modificar en el sistema, se define un listado de prioridades para minimizar el impacto de costo y cambio de arquitectura.

Los elementos descritos podrán ser implementados en cualquier orden, siguiendo los parámetros listados a continuación.

Elementos Críticos

- **Log-In, Kerberos + JAAS**. Tiene un costo de 80 horas, es mandatorio para mejorar la seguridad del sistema. Este cambio permitirá integrar la autenticación con plataformas externas y mejorará radicalmente la seguridad. Se puede implementar desde la arquitectura anterior y mantenerlo en el cambio.
- **Cambios en el diccionario de aplicación para integración de nuevos componentes**. Tiene un costo de 80 horas, es necesario realizarlo previo al cambio de arquitectura para asentar la nueva funcionalidad. La integración con Alfresco, autenticación, webservices, motor groovy y la nueva persistencia, requieren de nueva estructura en el diccionario.
- **API REST**, con un costo de 160 horas, se puede realizar independientemente a la etapa de cambio de arquitectura. Este módulo es deseable para luego poder fácilmente integrar nueva funcionalidad al sistema sin alterar la arquitectura actual.
- **Servidor de Aplicaciones**. Todos los elementos que se implementan en la etapa de elementos críticos necesitan ejecutarse sobre una nueva infraestructura de servidor de aplicaciones. Con un costo de 40 horas es necesario hacerlo antes del cambio de arquitectura, ya que la nueva funcionalidad requiere ejecutarse en servidores de aplicaciones

modernos.

- **Persistencia con Hibernate y Entidades con Inyección de Dependencias**. Suman un costo de 560 horas. Estos módulos modifican fuertemente la arquitectura del sistema, ambos componentes conforman el propósito del cambio de arquitectura sugerido en esta tesis. En el momento en el que se implementen los cambios, se bloquea y desaparece la arquitectura anterior, considerando al sistema "migrado" a la nueva arquitectura.
- **WebServices Integrados**. Con un costo de 80 horas, solo se puede desarrollar una vez que esté creado el nuevo motor de persistencia. Facilitará la integración de aplicaciones externas y hará que el sistema se establezca como una solución SOA.
- **Motor Groovy Integrado**. Tiene un costo de 160 horas, es necesario para completar el modelo e incluir la funcionalidad de modificación en caliente de código en la capa de negocio. Esta funcionalidad permitirá el mantenimiento de la aplicación en tiempo de ejecución.

Elementos Convenientes

- **Funciones Java**. Tiene un costo de 271 horas, es deseable para mantener la independencia del motor de base de datos, pero se puede realizar paulatinamente una vez se encuentre implementado el motor groovy.
- **Reorganización del Código Fuente**. Con tan solo 16 horas de costo mejora notablemente el mantenimiento y acelera el desarrollo.
- **Integración Bonitasoft**. Tiene un costo de 80 horas, es independiente a la arquitectura y mejora notablemente la operatividad de los documentos del sistema. Puede ser integrado mediante el API REST y es necesario establecer una buena estructura en el diccionario de aplicación.
- **Integración Alfresco**. La integración con Alfresco es conveniente para gestionar adecuadamente documentos digitalizados. Actualmente esta funcionalidad se encuentra muy adelantada e integrada a la arquitectura anterior.

- **Independencia de Interfaz de Usuario**. La nueva arquitectura implementada presenta todas las herramientas necesarias para diseñar una interfaz de usuario en cualquier lenguaje de programación, sin embargo el sistema necesita una interfaz propia e independiente para ejecutar las funciones del Framework y propias del ERP. Tiene un costo de 320 horas.

Elementos Accesorios

- **Capa de Traducción**. El costo de 40 horas es fácilmente justificable al mejorar una de las más grandes falencias operativas que tiene el sistema Adempiere, la traducción actual es compleja y difícil de administrar, con el cambio se facilita el mantenimiento y optimiza la resolución de elementos traducidos en cualquier idioma.
- **Módulo de Auditoría**. La auditoría actual es funcional, sin embargo crear el módulo de auditoría mejora el performance y facilita el mantenimiento del sistema al eliminar 4 campos obligatorios en todas las tablas del sistema. Este cambio tiene un costo de 60 horas.
- **Gestor de Catálogos**. Con un costo de 40 horas, esta funcionalidad está pensada también para reducir el mantenimiento de tablas del sistema, al agrupar todos los catálogos en un solo repositorio.
- **Toolkit para diseño de interfaz**. Es una herramienta accesoria para acelerar el desarrollo de formulario personalizados en el sistema, tiene un costo de 300 horas, cabe analizar el beneficio de este elemento, ya que la inversión frente al beneficio no puede ser muy favorable en todos los proyectos de implementación de ERP.

3.3. Modelos de Prueba de la Nueva Arquitectura

3.3.1. Integración de la herramienta externa Alfresco

Como ejemplo de integración de Adempiere con herramientas externas, se analizarán las modificaciones a realizar para obtener la funcionalidad de repositorio e indexación de Alfresco.

Alfresco es un potente gestor avanzado de documentos. En el ejemplo se integrará Adempiere con Alfresco, usándolo como contenedor de archivos digitalizados organizándolos por índices que se configurarán desde el diccionario de aplicación. Estos documentos podrán ser cargados y consultados dinámicamente desde el interfaz de usuario de Adempiere.

Modificación del diccionario de aplicación

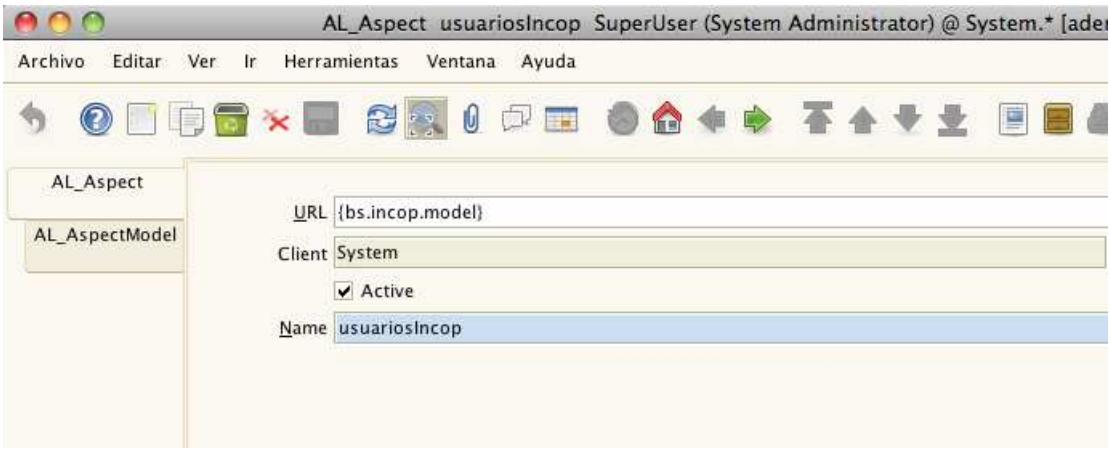


FIGURA 23, Ventana Alfresco Aspect, Pestaña Aspect

El la figura 4.1 se implementa la ventana aspecto donde se definen los índices documentales configurados en Alfresco.

El campo **URL** indica el espacio de nombres establecido en los índices personalizados en Alfresco.
El campo **Active** indica si el registro esta activo en Adempiere.
El campo **Name** registra el nombre que se le da al Aspecto.

FIGURA 24, Ventana Alfresco Aspect, Pestaña AspectModel

El campo **Active** indica si el registro esta activo en Adempiere.

El campo **Name** registra el nombre que se le da al Modelo del Aspecto.

El campo **Data Type** indica que tipo de campo va a ser registrado.

El campo **Default Logic** indica si el campo quedará automáticamente registrado por el contexto de la ventana del usuario. (Cañaveral, Salgado, 2012)

La figura 4.3 Ventana Alfresco Model representa a las modificaciones a realizar en el diccionario para configurar el modelo de Adempiere.

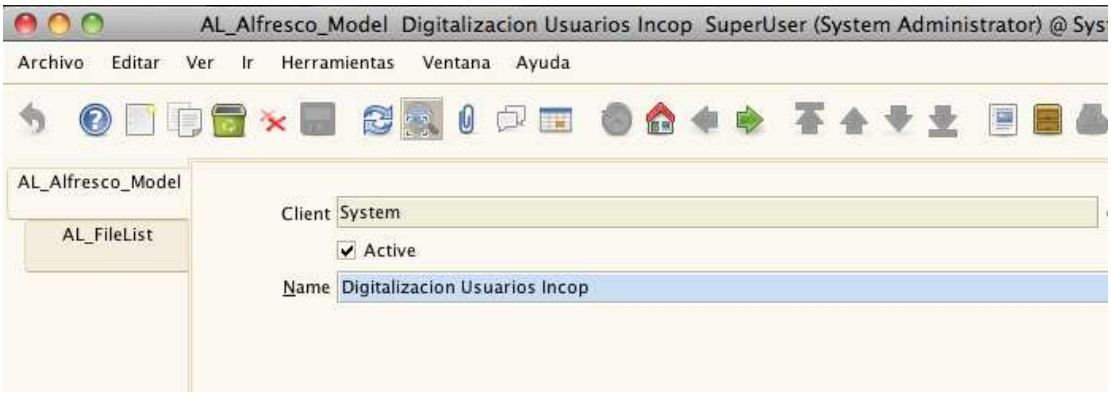


FIGURA 25, Ventana Alfresco Model, Pestaña Alfresco Model

El campo **Active** indica si el registro esta activo en Adempiere.

El campo **Name** registra el nombre que se le da al Modelo.

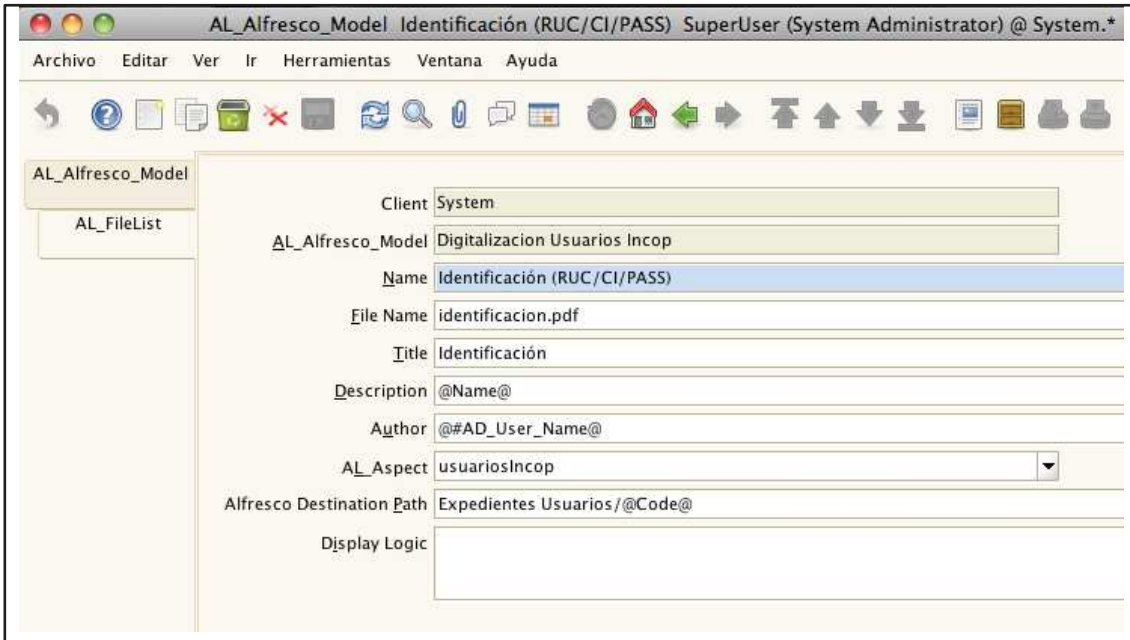


FIGURA 26, Ventana Alfresco Model, Pestaña FileList

El campo **Name** registra el nombre que se le da al Tipo de Modelo.

El campo **File Name** indica el nombre que va a tener el archivo que se está digitalizando en Alfresco.

El campo **Title** indica el título que va a tener el documento en las propiedades de Alfresco.

El campo **Description** indica la descripción que va a tener el documento en las propiedades de Alfresco.

El campo **Author** identifica el nombre del Usuario que realizo el registro.

El campo **AL_Aspect** registra el tipo de aspecto que va a tener este documento.

El campo **Alfresco Destination Path** muestra la ruta donde va a quedar registrado el documento en Alfresco.

El campo **Display Logic** permite desplegar dinámicamente el campo en la interfaz de usuario dependiendo de una condición. (Cañaveral, 2012.)

El código fuente se describe en el Anexo 1, Modelos de Prueba, Integración con Alfresco.

3.3.2. API REST

Se detalla un ejemplo de código fuente del API REST en el cuál se observa la funcionalidad para administrar roles y usuarios. Un cliente externo con las credenciales adecuadas podrá obtener, eliminar y crear roles y usuarios de Adempiere.

```
public class RESTRemoteIdentityAPIImpl extends AbstractRemoteIdentityAPIImpl
        implements RESTRemoteIdentityAPI {

    public List<Role> getRolesByUUIDs(List<String> roleUUIDs,
            Map<String, String> options) throws RemoteException,
            RoleNotFoundException {
        return getAPI(options).getRolesByUUIDs(roleUUIDs);
    }

    public List<User> getUsersByUUIDs(List<String> userUUIDs,
            Map<String, String> options) throws RemoteException,
            UserNotFoundException {
        return getAPI(options).getUsersByUUIDs(userUUIDs);
    }

        public void removeProfileMetadata(List<String> profileMetadataUUIDs,
            Map<String, String> options) throws RemoteException,
            MetadataNotFoundException {
        getAPI(options).removeProfileMetadata(profileMetadataUUIDs);
    }

    public void removeRoles(List<String> roleUUIDs, Map<String, String> options)
            throws RemoteException, RoleNotFoundException {
        getAPI(options).removeRoles(roleUUIDs);
    }

    public void removeUsers(List<String> userUUIDs, Map<String, String> options)
            throws RemoteException, UserNotFoundException {
        getAPI(options).removeUsers(userUUIDs);
    }
}
```

3.3.3. Integración Motor Groovy

Groovy permite instanciar clases “calientes” dinámicamente en java. En el ejemplo se instancia la clase HelloWorld escrita en groovy dentro del contexto de la máquina virtual de java.

```
ClassLoader parent = getClass().getClassLoader();
   GroovyClassLoader loader = new GroovyClassLoader(parent);
   Class groovyClass = loader.parseClass(new File("src/test/groovy/script/HelloWorld.groovy"));
```

Al instanciar una clase dinámica de groovy, se pueden hacer llamadas a sus métodos desde java, como se muestra en el siguiente ejemplo.

```
// llamar a un método de groovy.
    GroovyObject groovyObject = (GroovyObject) groovyClass.newInstance();
    Object[] args = {};
    groovyObject.invokeMethod("run", args);
```

Es más útil para el desarrollo de la nueva arquitectura de Adempiere, obtener las clases a partir de interfaces de java de la siguiente manera

```
    //Usando Interfaces
    GroovyClassLoader gcl = new GroovyClassLoader();
    Class clazz = gcl.parseClass(myStringwithGroovyClassSource, "SomeName.groovy");
    //Otra forma de crear la clase groovy
    Class clazz = gcl.parseClass(new File("SomeName.groovy");
    Object aScript = clazz.newInstance();
    MyInterface myObject = (MyInterface) aScript;
    myObject.interfaceMethod();
```

De esta forma la interfaz donde se describen los métodos del modelo de las clases M serán descritos en una interfaz y la funcionalidad de la lógica de negocio será escrita en groovy. Además es deseable que el código groovy esté en el propio diccionario de Adempiere, por ejemplo, escrito en un registro de la base de datos. La forma de llamarlo en java será obteniendo directamente los bytes del código groovy en vez de parsearlo como un archivo. (Comunidad Groovy, 2012).

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

- El Software Libre ha prevalecido frente al software propietario en los últimos años. Iniciativas como el decreto ejecutivo 1014 en Ecuador (*Utilización de Software Libre en los Sistemas y Equipamientos Informáticos en las Entidades de la Administración Pública Central*), han impulsado el desarrollo y mejora de estos sistemas. Anteriormente se consideraba al Software Libre como inseguro, sin soporte, de mala calidad y sin diseño de interfaz. Empresas como Alfresco, Bonitasoft, RedHat, EnterpriseDB y Canonical, han formado parte del cuadrante de Gartner anteponiéndose a gigantes como Oracle y SAP. La utilización de Software Libre de calidad, permite implementaciones cien por cien a medida, reduciendo significantemente el costo de propiedad.

- La implementación de un ERP constituye un gran salto administrativo y tecnológico dentro de cualquier compañía. Al ser un sistema que integra todos los procesos operativos, fortalece la comunicación entre dependencias, posibilitando llevar a cabo diferentes tareas de manera simultanea dentro de la organización. Aumenta los niveles de productividad disminuyendo el trabajo repetitivo, mejora la automatización de procesos brindando una fuente de información consolidada y estándar, reduce el costo operativo al automatizar procesos manuales, incrementa las buenas relaciones con los clientes y proveedores al establecer un proceso establecido de comunicación. Tecnológicamente reduce costos al contar con un framework desde el cuál se puede modelar cualquier requerimiento de sistema administrativo o financiero mejorando el mantenimiento y tiempo de respuesta al centralizarlo en una sola herramienta, por lo tanto una implementación de ERP se vuelve indispensable en empresas medianas y grandes ya que mejora notablemente su eficiencia y competitividad.

- Mediante análisis detallado de cada uno de los componentes, módulos del sistema, código fuente, extensibilidad y gestión de datos realizado en el capítulo 2, se demostró que la arquitectura del ERP Adempiere es obsoleta y de difícil mantenimiento contrastado con la potente estructura funcional de los módulos financieros, cuentas por pagar, cuentas por cobrar, análisis de desempeño, contabilidad y balances, que permiten registrar las entidades, transacciones contables y catálogos del sistema como dimensiones en un registro cúbico tipo OLAP. El manejo de estado de documentos, borrador, en proceso, completo, revertido y la utilización de tipos de documentos contables, facturas, notas de crédito, pagos y vales, son motivos por los cuales se decide mantener esta definición para la nueva arquitectura planteada en este proyecto de tesis.

- Se propusieron mejoras para cada una de las falencias del sistema. Reorganización del Código Fuente del ERP, mejorando la estructura del proyecto en Eclipse para facilitar el desarrollo, mejoramiento de la capa de traducción, elaborando un módulo que permita la independencia y portabilidad de traducciones, re diseño del módulo de auditoría desacoplándolo de todas las tablas del sistema, persistencia con Hibernate e inyección de dependencias reemplazando la obsoleta gestión de la data, traducción de funciones de la base de datos a funciones java permitiendo la portabilidad del sistema a cualquier motor de base de datos, log-in con Kerberos+JAAS estandarizando la administración de credenciales haciéndolo un sistema más seguro, generar independencia de la interfaz de usuario promoviendo el desarrollo de nuevas interfaces en la comunidad, nuevo módulo para la gestión de catálogos minimizando considerablemente el tamaño de la base de datos, elaboración de un toolkit para diseño de interfaces simplificando el desarrollo de formas personalizadas, la posibilidad de publicar y consumir servicios como parte del diccionario de aplicación e integrado en el sistema, poder modificar la lógica de negocio en tiempo

de ejecución mediante groovy y brindar facilidades de conexión mediante APIS, se vuelven fundamentales al establecer un modelo de sistema que sea adaptable a las diversas implementaciones donde se aplique el ERP.

- En el capítulo 3 se identificó que funcionalidades como el manejo del log-in el cuál es inseguro, persistencia y base de datos obsoleta y repetitiva, manejo documental en la propia infraestructura del ERP, workflow de documentos simple y de difícil administración, auditoría integrada en cada tabla del sistema y demás falencias que constan en el análisis, tienen deficiencias por ser módulos integrados de manera tardía o desordenada. Se consideran estos elementos como un todo y no sus partes por separado creando un sistema distribuido, completo y que considere cada funcionalidad al momento de aplicar la nueva arquitectura.

- Apoyarse en herramientas que tienen resuelta la funcionalidad de su negocio, como Bonitasoft el cual consta de un potente manejador de flujo de procesos, permitiendo crearlos desde aplicaciones externas y Alfresco, gestor documental líder en el mercado; Simplifican notablemente la reestructuración del sistema y a la vez lo dotan de funcionalidad adicional indispensable en la nueva arquitectura del Framework y del ERP, brindando características específicas de estos sistemas dentro de Adempiere.

- Se ha demostrado que la arquitectura propuesta manteniendo los componentes financieros y contables, con un servidor de aplicaciones remodelado, incluyendo la funcionalidad de procesos asíncronos, renovado y potente motor de persistencia, integración con herramientas externas, posibilidad de cambiar la lógica de negocio en tiempo de ejecución, portabilidad de interface de usuario, manejo seguro de la información y estándares adecuados, es un proyecto factible dentro de

un costo que no supera el 30% de una implementación estándar de un ERP y además es posible realizarla en un tiempo adecuado contando con los recursos necesarios para la remodelación.

Al elaborar el análisis de brechas se demostró que la propuesta de realizar el cambio de arquitectura paulatinamente es totalmente viable y segura, empezando por los elementos críticos; log-in, API-REST, Servidor de Aplicaciones, WebServices, motor grooby, persistencia e inyección de dependencias; A continuación se deben implementar los elementos convenientes, funciones java, reorganización del código fuente, integración con Bonitasoft, integración con Alfresco y terminar con los elementos convenientes que son, la capa de traducción, módulo de auditoría, gestor de catálogos y el toolkit de desarrollo de la interfaz. Esto muestra que la propuesta de diseño distribuido e independiente minimizan notablemente el impacto del cambio, siendo el costo en la elaboración de cada uno de los elementos el que determinará el orden en el cuál se desarrollarán las nuevas funcionalidades.

4.2. Recomendaciones

- El incremento de tecnologías distribuidas y la aparición del Cloud Computing, hacen indispensable hacer énfasis en la seguridad y a la vez facilitan la autenticación y credenciales del sistema. Es por esto que se recomienda que el primer cambio a realizar en Adempiere sea el log-in, mejorándolo con un protocolo seguro como es Kerberos, mediante JAAS.

- Se recomienda evitar sobrecargar las tablas del sistema con campos repetitivos como son created, createdby, updated y updatedby. El uso de un módulo de auditoría que gestione los cambios realizados por el sistema reemplaza esta mala práctica.

- Para una implementación de ERP, la necesidad de crear catálogos es indispensable, actualmente esto se realiza creando tablas que faciliten su manejo. Para un mejor mantenimiento, se recomienda utilizar el módulo propuesto de gestión de catálogos.

- Aunque la implementación del nuevo motor de persistencia permite independencia del gestor de base de datos, es recomendable el uso de una base de datos transaccional, con características avanzadas de administración y distribución (recuperación de datos, cluster, manejo de tablespaces, upsizing, etc). Ya que el diccionario de aplicación de Adempiere requiere un alto performance de la base de datos.

- Con la nueva arquitectura diseñada para Adempiere, se podrá extender el sistema de dos formas, por medio de paquetes jar realizados en java y mediante el nuevo motor de groovy. Se recomienda implementar en groovy solamente la lógica de negocio levantada para cada implementación. La funcionalidad adicional y que no sea susceptible de cambio se deberá realizar directamente sobre java, esta recomendación

se fundamente en el mejoramiento de performance requerido en el sistema.

- Las nuevas tendencias sociales, económicas, contables (nuevas NIF) y financieras pueden no ser compatibles con el modelo funcional actual del sistema, una vez que tecnológicamente esté resuelta la capacidad de implementación de cualquier lógica de negocio, se recomienda realizar un amplio análisis, con especialistas administrativos, de recursos humanos y financieros, para que evalúen la posible reestructuración funcional de Adempiere.

5. REFERENCIAS

- Adempiere Community Portal. (s.f.) Recuperado el 20 de Agosto de 2012 http://www.adempiere.com/ADempiere_ERP_Wiki:Community_Portal.
- Blog sobre SOA. (s.f.) Implementación de SOA con REST. Recuperado el 1 de septiembre de 2012 de http://pensandoensoa.com/2011/03/19/869/
- Crespo, C. (s.f.). (2007), Introducción a Hibernate. Recuperado el 12 de agosto de 2012 de http://www.adictosaltrabajo.com/tutoriales/tutoriales.php
- Cañaveral, A. (2009). Manual de Usuario Sistema Adempiere. Quito, Ecuador.
- Comunidad Adempiere. (s.f.) (2012). Recuperado el 28 de agosto de 2012 de http://www.adempiere.org
- Cañaveral, A. Salgado, V. (2012). Manual de Usuario Integración Alfresco. Quito, Ecuador.
- Cañaveral, A. (2010) Manual Técnico, Personalización de Adempiere. Quito, Ecuador.
- Grooby Community (s.f.) Embedding Groovy. Recuperado el 1 de Septiembre de 2012 de http://groovy.codehaus.org/Embedding+Groovy
- Hibernate Community (s.f.) Recuperado el 12 de agosto de 2012 de http://docs.jboss.org/hibernate
- Kumar, A. (s.f.) (2011). *ADempiere 3.6 Cookbook*. Birmingham, UK: Packt Publishing.
- Locke, J. (s.f.) (2004). Open Source Solutions form Small Business Problems. Massachusetts, USA : Networking Series.
- Mahmoud, Q. (s.f.) (2005). SOA y Web Services. Recuperado el 12 de agosto de 2012 de http://www.oracle.com/technetwork/articles/javase/soa-142870.html
- Oon, R. (s.f.) (2010). *Open Source ERP from ADempiere, top world SourceForge project*. Malaysia : Pearson.

- Pamungkas, B. (s.f.) (2004). *ADempiere 3.4 ERP Solutions*. Birmingham, UK: Packt Publishing.
- Rod J., Juergen H., Keith D., Colin S., Rob H., Alef A., Thomas R., Darren D., Dmitriy K., Mark P., Thierry T., Erwin V., Portia T., Ben H., Adrian C., John L., Costin L., Mark F., Sam B., Ramnivas L., Arjen P., Chris B., Tareq A., Andy C., Dave S., Oliver G., Rossen S. (2012). (s.f.) *Spring Framework Reference Documentation 3.1*. Reference Documentation : springsource.org
- Spring Source Community. (s.f.) Recuperado el 12 de agosto de 2012 de http://www.springsource.org/
- Spring Source & Groovy Community, (s.f.) Un lenguaje dinámico para la plataforma Java. Recuperado el 12 de agosto de 2012 de http://groovy.codehaus.org/
- Sourceforge. ADempiere ERP Bussiness Suite. (s.f.) Recuperado el 20 de Agosto de 2012 http://sourceforge.net/projects/adempiere/
- Sicilia, M. (s.f.) ¿Qué es la ingeniería inversa?. Recuperado el 27 de Agosto de 2012 http://cnx.org/content/m17432/latest/
- vaxMan (s.f.) (2011). Seguridad en Web Services. Recuperado el 12 de agosto de 2012 de http://www.hacktimes.com/seguridad_en_web_services/

6. ANEXOS

ANEXO 1

MODELOS DE PRUEBA

Iniciar Entorno de Adempiere en Modo debug.

```
public class AdempiereTest {
  /**
   * @autor Antonio Canaveral
   */
  public static void main(String args[]) {
    if (org.compiere.Adempiere.getVersion().length() != 0) {
      org.compiere.Adempiere.startup(true);
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_UID, "usuario");
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_PWD, "password");
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_ROLE, "Sistemas");
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_CLIENT, "CLIENTE");
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_ORG, "*");
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_WAREHOUSE, "*");
      org.compiere.util.Ini.setProperty(org.compiere.util.Ini.P_LANGUAGE, "Español(MX)");
      org.compiere.util.Login login = new org.compiere.util.Login(Env.getCtx());
      if (login.batchLogin()) {
        Adempiere.startupEnvironment(true);
        CLogMgt.setLevel(Level.FINE);

      }
    }
  }
}
```

Modelo de Prueba, Integración con Bonitasoft.

```
/**
 * @author Elias Ricken de Medeiros
 * @modified Antonio Canaveral
 */
public class Client {

  private static final String LOGIN = "john";
  private static final String PSSWD = "bpm";
  private static final String jaasFile = "src/main/resources/jaas-standard.cfg";

  public static void main(String[] args) throws Exception {

    // establecemos las propiedades del sistema
    System.setProperty(BonitaConstants.API_TYPE_PROPERTY, "REST");
    System.setProperty(BonitaConstants.REST_SERVER_ADDRESS_PROPERTY,
        "http://localhost:8080/bonita-server-rest/");
    System.setProperty(BonitaConstants.JAAS_PROPERTY, jaasFile);

    // login
    // verificación de existencia del usuario
    LoginContext loginContext = new LoginContext("BonitaAuth", new
SimpleCallbackHandler(LOGIN,
        PSSWD));
    loginContext.login();
    loginContext.logout();

    // propagar las credenciales del usuario
    loginContext = new LoginContext("BonitaStore", new SimpleCallbackHandler(LOGIN,
PSSWD));
    loginContext.login();

    // obtenemos las APIs
    final ManagementAPI managementAPI = AccessorUtil.getManagementAPI();
    final RuntimeAPI runtimeAPI = AccessorUtil.getRuntimeAPI();
    final QueryRuntimeAPI queryRuntimeAPI = AccessorUtil.getQueryRuntimeAPI();

    try {

      // crear un proceso simple:
      // - un paso con el actor LOGIN
      // - una variable global String
      ProcessDefinition process = ProcessBuilder.createProcess("myProcess",
"1.0").addStringData(
          "globalVar", "defaultValue").addHuman(LOGIN).addHumanTask("step1", LOGIN).done();

      // desplegamos el proceso
      process = managementAPI.deploy(BusinessArchiveFactory.getBusinessArchive(process));

      System.out.println("----------------\nProcess deployed\n----------------");

      final ProcessDefinitionUUID processUUID = process.getUUID();
      // instanciamos el proceso
      ProcessInstanceUUID instanceUUID = runtimeAPI.instantiateProcess(processUUID);
      System.out.println("----------------\nNew process instance Created\n----------------");

      final Collection<LightTaskInstance> taskList =
```

```
queryRuntimeAPI.getLightTaskList(instanceUUID,
        ActivityState.READY);
    if (taskList.size() != 1) {
      throw new Exception("Incorrect list size. Actual size: " + taskList.size());
    }

    // ejecutamos una tarea
    final LightTaskInstance taskInstance = taskList.iterator().next();
    runtimeAPI.executeTask(taskInstance.getUUID(), true);
    System.out.println("----------------\nTask executed\n----------------");

    final InstanceState state = queryRuntimeAPI.getProcessInstance(instanceUUID)
        .getInstanceState();
    if (!state.equals(InstanceState.FINISHED)) {
      throw new Exception("Incorrect state. Actual state: " + state);
    }

    System.out.println("----------------\nApplication executed sucessfully\n----------------");

  } finally {
    // borramos todos los procesos desplegados en el servidor
    managementAPI.deleteAllProcesses();
    loginContext.logout();
  }

 }

}
```

Modelo de Prueba, Integración con Alfresco

* Pantalla de Opciones de Digitalización – Vista Previa del documento adjunto

Modificaciones en la clase Attachment

```
private void loadAttachments(boolean reload) {
   log.config("");
   // Set Text/Description
   String sText = m_attachment.getTextMsg();
   if (sText == null)
     text.setText("");
   else
     text.setText(sText);

   if (m_attachment.isStoreAttachmentsOnAlfresco()) {
     if (reload)
       m_attachment.deleteAlfrescoEntries();
     // Set List
     Vector<Vector<Object>> model = new Vector<Vector<Object>>();
     Vector<Object> item = null;
     // LLENO LA TABLA CON LOS ARCHIVOS QUE DEBERIAN ESTAR
     for (X_AL_FileList file : m_attachment.getAlfrescoModel().getFileList()) {
       // ME CONECTO A ALFRESCO EN BUSCA DEL ARCHIVO.
       AlfrescoUpload up = m_attachment.getAlfrescoUpload();
       boolean exists = up.fileExists(m_attachment.parsePath(file.getAlfrescoDestinationPath()),
           file.getFileName());
       item = new Vector<Object>();
       item.add(exists);
       item.add(file.getName());
       item.add(file);
       item.add(m_attachment.addAlfrescoEntry(file));
       model.add(item);

     }
     alfrescoTable.setModel(new AttachmentFileAlfrescoModel(model, false));
     alfrescoTable.getColumnModel().getColumn(0).setPreferredWidth(22);
     alfrescoTable.getColumnModel().getColumn(1).setPreferredWidth(180);
     alfrescoTable.setColumnVisibility(alfrescoTable.getColumnModel().getColumn(2), false);
     alfrescoTable.setColumnVisibility(alfrescoTable.getColumnModel().getColumn(3), false);

   } else {
     // Set Combo
     int size = m_attachment.getEntryCount();
     for (int i = 0; i < size; i++)
       cbContent.addItem(m_attachment.getEntryName(i));
     if (size > 0)
       cbContent.setSelectedIndex(0);
     else
       displayData(0);
   }
 } // loadAttachment
```

Modificaciones en la clase MAttachment

```
 private void alfrescoInitialization(Properties ctx, String trxName) {
  final MTable table = new MTable(this.getCtx(), this.getAD_Table_ID(), trxName);
  isStoreAttachmentsOnAlfresco = table.isStoreAttachmentsOnAlfresco();
  if (isStoreAttachmentsOnAlfresco) {
    String whereClause = MALAlfrescoModel.COLUMNNAME_AL_Alfresco_Model_ID + " = ?";
    Query q = new Query(ctx, MALAlfrescoModel.Table_Name, whereClause, trxName)
       .setParameters(new Object[] { table.get_ValueAsInt("AL_Alfresco_Model_ID") });
    m_alfrescoModel = (MALAlfrescoModel) q.first();

    MSystem system = MSystem.get(this.getCtx());
    m_attachmentAlfrescoUrl = system.getAlfrescoUrl();
    m_attachmentAlfrescoUsername = system.getAlfrescoUsername();
    m_attachmentAlfrescoPassword = system.getAlfrescoPassword();

    if ("".equals(m_attachmentAlfrescoUrl)) {
      log.severe("no alfrescoUrl defined");
    }
    if ("".equals(m_attachmentAlfrescoUsername)) {
      log.severe("no alfrescoUsername defined");
    }
    if ("".equals(m_attachmentAlfrescoPassword)) {
      log.severe("no alfrescoPassword defined");
    }

    alfUp = new AlfrescoUpload();
    alfUp.setServer_host(m_attachmentAlfrescoUrl.split(":")[0]);
    alfUp.setServer_port(m_attachmentAlfrescoUrl.split(":")[1]);
    alfUp.setServer_username(m_attachmentAlfrescoUsername);
    alfUp.setServer_password(m_attachmentAlfrescoPassword);
  }
 }

public AlfrescoUpload getAlfrescoUpload() {
   return alfUp;
 }

 public boolean isStoreAttachmentsOnAlfresco() {
   return isStoreAttachmentsOnAlfresco;
 }

 public MALAlfrescoModel getAlfrescoModel() {
   return m_alfrescoModel;
 }

 public int addAlfrescoEntry(X_AL_FileList file) {
   MAttachmentEntry item = new MAttachmentEntry(file.getFileName(), null, null);
   m_itemsAlfresco.add(new Object[] { item, file });
   return m_itemsAlfresco.size() - 1;
 } // addEntry


private boolean loadLOBDataFromAlfresco(String destinationPath, String fileName) {

   m_items = new ArrayList<MAttachmentEntry>();

   try {
```

```
      byte[] data = alfUp.readFile(destinationPath, fileName);
      final File file = new File(fileName);
      MAttachmentEntry entry = null;
      FileOutputStream fileOuputStream = new FileOutputStream(file);
      fileOuputStream.write(data);
      fileOuputStream.close();
      entry = new MAttachmentEntry(file.getName(), data, null, m_items.size() + 1);
      m_items.add(entry);
      log.fine("Listed From Alfresco");
      setBinaryData(null);
      return true;
    } catch (Exception e) {
      log.log(Level.SEVERE, "loadLOBDataFromAlfresco", e);
      // ADialog.error(0, null, e.getLocalizedMessage());
    }
    return false;
  }

private boolean saveLOBData() {
    if (isStoreAttachmentsOnAlfresco) {
      if (!saveLOBDataToAlfresco("", "", ATTACHMENT_FOLDER_PLACEHOLDER)) {
        return false;
      }
    }
    if (isStoreAttachmentsOnFileSystem) {

      return saveLOBDataToFileSystem();
    }
    return saveLOBDataToDB();
  }
```

Implementación del conector con Alfresco

```
/**
 * @autor Antonio Canaveral
 */
public class AlfrescoUpload extends AlfrescoContentService {

  public AlfrescoUpload() {

  }

  public AlfrescoUpload(String server_host, String server_port, String server_username,
      String server_password) {
    this.server_host = server_host;
    this.server_port = server_port;
    this.server_username = server_username;
    this.server_password = server_password;
  }

  public void setServer_host(String server_host) {
    this.server_host = server_host;
  }

  public void setServer_port(String server_port) {
    this.server_port = server_port;
  }

  public void setServer_username(String server_username) {
    this.server_username = server_username;
  }

  public void setServer_password(String server_password) {
    this.server_password = server_password;
  }

  public void setContentDescription(String description) {
    content_description = description;
  }

  public void setContentTitle(String title) {
    content_title = title;
  }

  public void setContentAuthor(String author) {
    content_author = author;
  }

  public void setSpaceDescription(String description) {
    space_description = description;
  }

  public Reference createPathToAlfresco(String path) {
    startSession();
    String alfresco_path[] = path.split(SEPARATOR);
    Reference parent = null;
    try {
      parent = getCompanyHome();
      String currentNode = "";
```

```
    for (String unitPath : alfresco_path) {
      currentNode = currentNode + SEPARATOR + unitPath;
      Reference node = getSpaceReference(currentNode);
      if (node != null && node.getUuid() == null) {
        parent = createSpace(parent, unitPath);
      }
    }

  } catch (Exception e) {
    e.printStackTrace();
    parent = null;
  }
  endSession();
  return parent;
}

private Reference createPathToAlfrescoNoSession(String path) {
  String alfresco_path[] = path.split(SEPARATOR);
  Reference parent = null;
  try {
    parent = getCompanyHome();
    String currentNode = "";
    for (String unitPath : alfresco_path) {
      currentNode = currentNode + SEPARATOR + unitPath;
      Reference node = getSpaceReference(currentNode);
      if (node != null && node.getUuid() == null) {
        parent = createSpace(parent, unitPath);
      }
    }

  } catch (Exception e) {
    e.printStackTrace();
    parent = null;
  }
  return parent;
}

public boolean fileExists(String path, String fileName) {
  boolean exists = false;
  try {
    startSession();
    Reference ref = createPathToAlfrescoNoSession(path);
    if (ref != null)
      if (getContent(ref, fileName) != null)
        exists = true;
  } catch (Exception e) {
    e.printStackTrace();
  }
  endSession();
  return exists;
}

public byte[] readFile(String path, String fileName) {
  byte[] data = null;
  if (fileExists(path, fileName)) {
    try {
      startSession();
      Reference ref = createPathToAlfrescoNoSession(path);
```

```
      if (ref != null)
        data = getContentAsByte(ref, fileName);
     } catch (Exception e) {
      e.printStackTrace();
     }
     endSession();
   }

   return data;
  }

  public Reference uploadFileToAlfresco(Reference spaceName, String name, String
pathToFile) {
   Reference doc = null;

   startSession();
   byte[] data = fileToByte(pathToFile);
   try {
    doc = getContent(spaceName, name);
    if (doc == null)
     doc = createDocument(spaceName, name, data);
    else
     doc = updateDocument(doc, name, data);

   } catch (Exception e) {
    e.printStackTrace();
   }
   endSession();

   return doc;
  }

  public Reference uploadFileToAlfrescoEx(Reference spaceName, String name, String
pathToFile)
     throws Exception {
   Reference doc = null;

   startSession();
   byte[] data = fileToByte(pathToFile);
   doc = getContent(spaceName, name);
   if (doc == null)
    doc = createDocument(spaceName, name, data);
   else
    doc = updateDocument(doc, name, data);

   endSession();

   return doc;
  }

  public Reference uploadFileToAlfresco(Reference spaceName, String name, String
pathToFile,
     Aspect as) {
   Reference doc = null;

   startSession();
   byte[] data = fileToByte(pathToFile);
   try {
```

```
      doc = getContent(spaceName, name);
      if (doc == null)
        doc = createDocument(spaceName, name, data, as);
      else
        doc = updateDocument(doc, name, data, as);
    } catch (Exception e) {
      e.printStackTrace();
    }
    endSession();

    return doc;
  }

  public Reference uploadFileToAlfrescoEx(Reference spaceName, String name, String
pathToFile,
      Aspect as) throws Exception {
    Reference doc = null;

    startSession();
    byte[] data = fileToByte(pathToFile);

    doc = getContent(spaceName, name);
    if (doc == null)
      doc = createDocument(spaceName, name, data, as);
    else
      doc = updateDocument(doc, name, data, as);

    endSession();

    return doc;
  }

  public Reference uploadFileToAlfresco(String path, String name, String pathToFile) {
    Reference doc = null;

    startSession();
    String alfresco_path[] = path.split(SEPARATOR);
    Reference parent = null;
    try {
      parent = getCompanyHome();
      String currentNode = "";
      for (String unitPath : alfresco_path) {
        currentNode = currentNode + SEPARATOR + unitPath;
        Reference node = getSpaceReference(currentNode);
        if (node != null && node.getUuid() == null) {
          parent = createSpace(parent, unitPath);
        }
      }
      if (parent != null) {
        byte[] data = fileToByte(pathToFile);
        doc = createDocument(parent, name, data);
      }
    } catch (Exception e) {
      e.printStackTrace();
      parent = null;
    }
    endSession();
```

```
  return doc;
}

public boolean uploadFileToAlfresco(String path, String name, byte[] data) {
  boolean res = false;
  startSession();
  String alfresco_path[] = path.split(SEPARATOR);
  Reference parent = null;
  try {
    parent = getCompanyHome();
    String currentNode = "";
    for (String unitPath : alfresco_path) {
      currentNode = currentNode + SEPARATOR + unitPath;
      Reference node = getSpaceReference(currentNode);
      if (node != null && node.getUuid() == null) {
        parent = createSpace(parent, unitPath);
      }
    }
    if (parent != null) {
      if (createDocument(parent, name, data) != null)
        res = true;
    }
  } catch (Exception e) {
    e.printStackTrace();
    parent = null;
  }
  endSession();
  return res;
}

private byte[] fileToByte(String pathToFile) {
  File f = new File(pathToFile);
  FileInputStream fin = null;
  FileChannel ch = null;
  byte[] bytes = null;
  try {
    fin = new FileInputStream(f);
    ch = fin.getChannel();
    int size = (int) ch.size();
    MappedByteBuffer buf = ch.map(MapMode.READ_ONLY, 0, size);
    bytes = new byte[size];
    buf.get(bytes);

  } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  } finally {
    try {
      if (fin != null) {
        fin.close();
      }
      if (ch != null) {
        ch.close();
      }
    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
```

```
    }
    return bytes;

  }
}
```

ANEXO 2 - MANUAL TÉCNICO ADEMPIERE

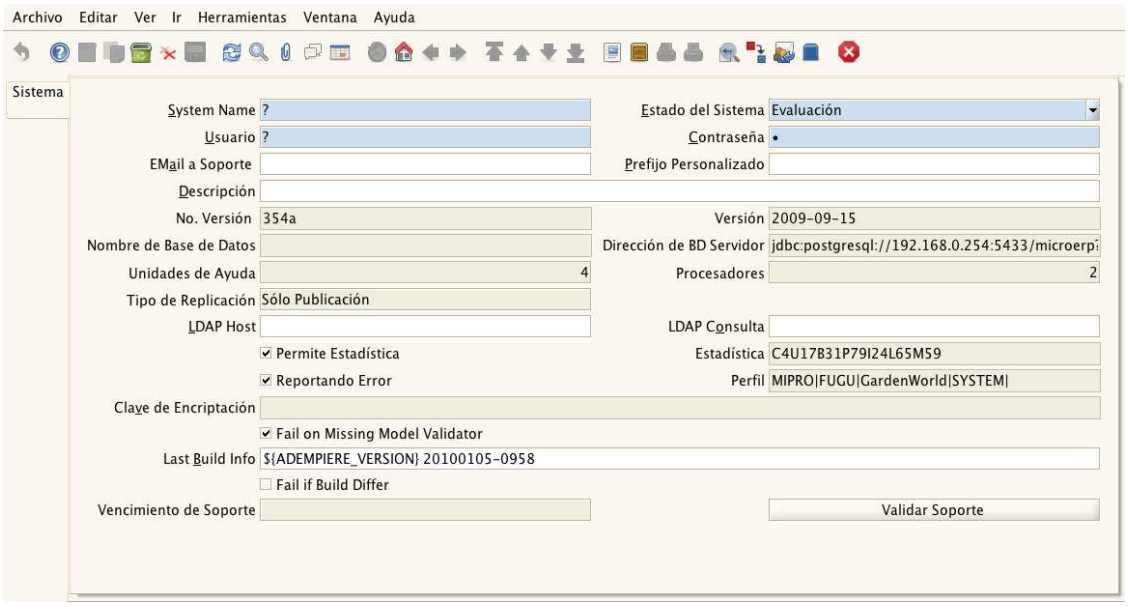
Estructura de datos
(Diccionario de aplicaciones y personalización de vistas y tablas)

Diccionario de aplicación

¿Como podemos evitar el trabajo de programación al querer personalizar Adempiere?. Primero que nada tenemos que entender el funcionamiento de las aplicaciones, el trabajo fundamental de las aplicaciones, es el de procesar información, usualmente adquiriéndola de tablas y depositando nuevamente en tablas. Todo esto es hecho vía una interface donde el usuario da la orden. Ahora imaginemos el separar las cosas que no cambian y las que si cambian, los elementos cambiantes son por ejemplo el look and feel, el acomodo de las ventanas, las pestañas, los campos y los reportes. Los elementos que no cambia serian: el nuevo, ve al siguiente registro, borrar, buscar, etc., los accesos a estas acciones se encuentran en el menú.

Otros ejemplos de cosas que cambian son las reglas o lógica del negocio que se refieren a que hacer antes o después de procesar la información. La manera en que Adempiere maneja la personalización de la aplicación es muy inteligente y tiene detrás una muy bien planeada base de datos. Adempiere trata de absorber todos estos cambios lo más que pueda con el uso de *metadata*, que es información de información, por lo que en lugar de programar, tiene una interface la cual hace la parte pesada de la programación, nosotros solo tenemos que dar las instrucciones a una interface llamada *Diccionario de Aplicación.* Esto da como resultado que la mayoría de los cambios que deseamos hacer pueden ser hechos aquí.
El diccionario de aplicación se compone de las siguientes partes:

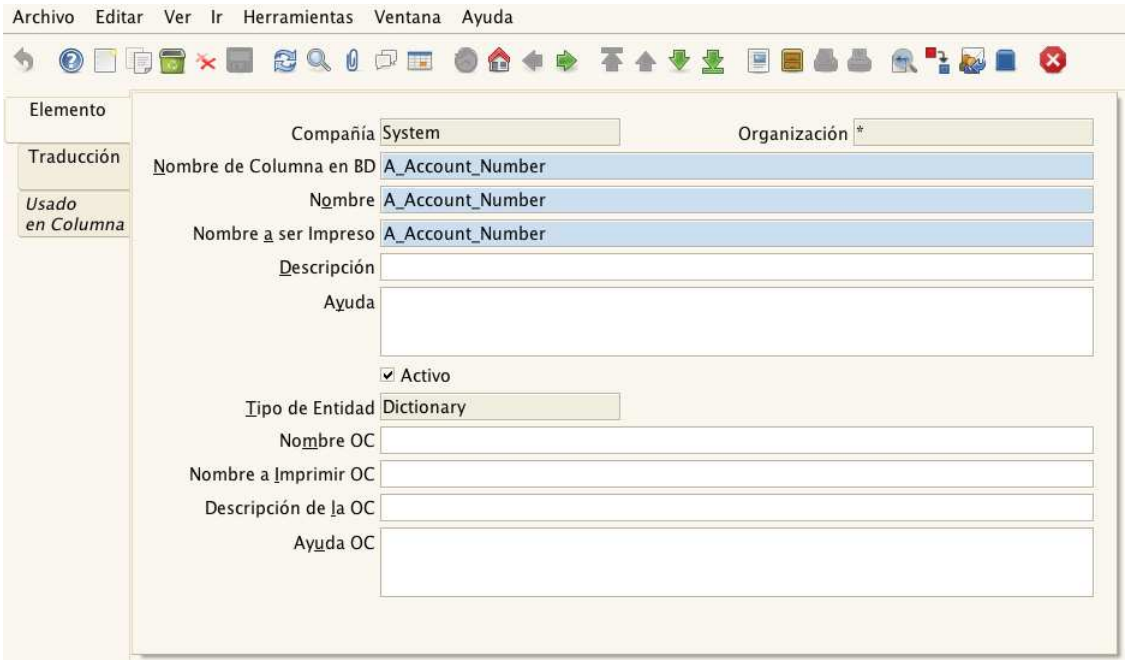
Sistema
En esta ventana se definen los datos del sistema, indicando la información y estadística del mismo.



Elemento

Se encuentran todos los elementos o variables de Adempiere, estos elementos se dan de alta al tener una columna nueva que no existía en ninguna de las otras tablas. En esta ventana se definen las propiedades de los campos o elementos, con el objetivo de tener centralizadas esas propiedades.

Las propiedades controladas aquí son: el nombre a ser impreso en las ventanas, la descripción, la ayuda y la traducción. Si hacemos un cambio en alguno de los elementos este se verá reflejado en todas las ventanas que utilicen ese elemento, después de sincronizar la terminología.



DB Column Name: En este campo tenemos que poner el nombre exacto de cómo esta definida la columna en la tabla.

Name: Se coloca el nombre de la tabla a la que se hará referencia

Print text: Es el nombre del campo que tendrán las ventanas que utilicen este elemento.

Description: Este texto es el que despliega cuando el cursor se encuentra sobre algún campo de las ventanas.

Comment/Help: La información capturada en este campo servirá para alimentar la ayuda del sistema para el usuario final, la cual puede ser encontrada en cada ventana proporcionándole una idea más clara de la función de cada uno de los campos de dicha ventana.

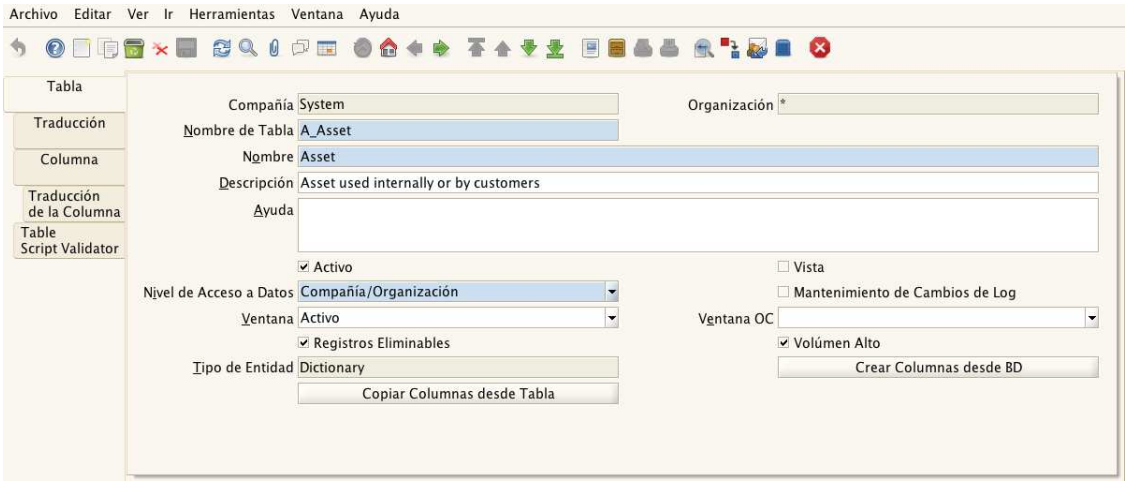
PO Name (Nombre OC): En caso de que el elemento sea usado tanto en ventanas del módulo de ventas y del módulo de compras se definen aquí las características ó preferencias que tendrá este campo en el módulo de compras.

Nota: los campos PO Print Name, PO Description y PO Help tienen las mismas funciones que los campos: Print Text, Description y Comment/help respectivamente; la diferencia es que el campo tomara estas características si es usado en una ventana del modulo de compras.

Tablas y Columnas

En este lugar se dan de alta nuevas tablas y sus campos, ya sea que los campos se den de alta aquí uno por uno y después hacer la sincronización para realizar el cambio en la base de datos, ó primero hacer el script y subir la tabla con sus campos y después importar la tabla para crear los campos conforme a la tabla en la base de datos.

En la pestaña de tabla debemos resaltar los siguientes campos que son el nombre, que es el nombre con que será listada la tabla en Adempiere el cual debe ser igual al nombre de la tabla, otro campo que debemos de analizar es el checkbox de vista, el cual debe estar seleccionado si es que esta tabla se utilizará para la creación de algún reporte, el campo de nivel de acceso, es para limitar el acceso a esta tabla.



La opción de Active es utilizada para que a la hora de abrir la tabla este de forma activa para el usuario que esta intentando ingresar a ella.

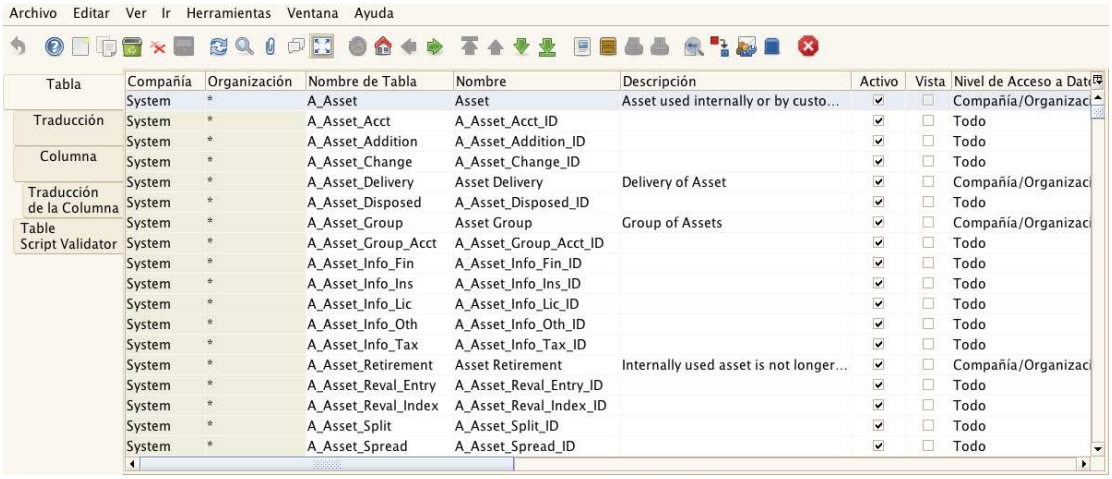
Data Access Level: aquí seleccionamos para tipo de usuario puede estar

activa nuestra nueva ventana. Las opciones de *Windows y PO windows* son para establecer el tipo de acceso teniendo la misma
diferencia de lo explicado anteriormente.

Al tener activa la opción de *Records deleteable* podremos borrar los registros establecidos en esta tabla. En la opción de *Entity type,* de donde estar direccionado nuestra tabla.

En cuanto a las columnas de la tabla tendríamos que resaltar las siguientes, Nombre de Columna en la base de datos*,* que es el nombre que le pusimos a ese campo, el cual debe ser el mismo nombre que en el elemento, en dado caso que ese elemento no exista tendremos que darlo de alta accensando a la ventana de elemento utilizando la propiedad de zoom de Adempiere, *el campo de referencia* es muy importante ya que se refiere al tipo de dato que es cada una de las columnas, las referencias que podemos ver en esta lista son las siguientes: cuenta, cantidad, asignación, binario, botón, color, fecha, fecha y hora, id, imagen, entero, lista, localización, menu número, pattribute, cantidad, búsqueda, cadena, tabla, tabledir, texto, textolong, si/no, tiempo. El campo de *lógica predeterminada* es donde se pone el valor por default que queremos que tenga esa columna en particular por ejemplo un campo tipo fecha que queremos que tenga la fecha de hoy a la hora que creamos el nuevo registro, se tendría que poner @#Date@ en el campo de lógica predeterminada. Los checkboxs de Columna clave, es la columna de enlace a tabla padre, entrada obligatoria, actualizable, despliegue encriptado, el campo de lógica de solo lectura es para que de acuerdo a los valores de otras variables. Este campo pueda ser cambiado o no, el campo de llamada es utilizado para llamar a las clases de callout los cuales son utilizados para realizar algún calculo en específico y regresar algún valor.

Otra forma de visualizar la ventana de la tabla, se muestra en la siguiente:



Compañía	Organización	Nombre de Tabla	Nombre	Descripción	Activo	Vista	Nivel de Acceso a Dat
System	*	A_Asset	Asset	Asset used internally or by custo...	☑	☐	Compañía/Organizaci
System	*	A_Asset_Acct	A_Asset_Acct_ID		☑	☐	Todo
System	*	A_Asset_Addition	A_Asset_Addition_ID		☑	☐	Todo
System	*	A_Asset_Change	A_Asset_Change_ID		☑	☐	Todo
System	*	A_Asset_Delivery	Asset Delivery	Delivery of Asset	☑	☐	Compañía/Organizaci
System	*	A_Asset_Disposed	A_Asset_Disposed_ID		☑	☐	Todo
System	*	A_Asset_Group	Asset Group	Group of Assets	☑	☐	Compañía/Organizaci
System	*	A_Asset_Group_Acct	A_Asset_Group_Acct_ID		☑	☐	Todo
System	*	A_Asset_Info_Fin	A_Asset_Info_Fin_ID		☑	☐	Todo
System	*	A_Asset_Info_Ins	A_Asset_Info_Ins_ID		☑	☐	Todo
System	*	A_Asset_Info_Lic	A_Asset_Info_Lic_ID		☑	☐	Todo
System	*	A_Asset_Info_Oth	A_Asset_Info_Oth_ID		☑	☐	Todo
System	*	A_Asset_Info_Tax	A_Asset_Info_Tax_ID		☑	☐	Todo
System	*	A_Asset_Retirement	Asset Retirement	Internally used asset is not longer...	☑	☐	Compañía/Organizaci
System	*	A_Asset_Reval_Entry	A_Asset_Reval_Entry_ID		☑	☐	Todo
System	*	A_Asset_Reval_Index	A_Asset_Reval_Index_ID		☑	☐	Todo
System	*	A_Asset_Split	A_Asset_Split_ID		☑	☐	Todo
System	*	A_Asset_Spread	A_Asset_Spread_ID		☑	☐	Todo

Como se puede observar en la imagen anterior existen diferentes prefijos que se utilizan para diferenciar el tipo de tabla, los prefijos son los siguientes:

AD: Son para los diccionario de datos.

A: Son para las tablas de activos.

B: Son las tablas de subastas.

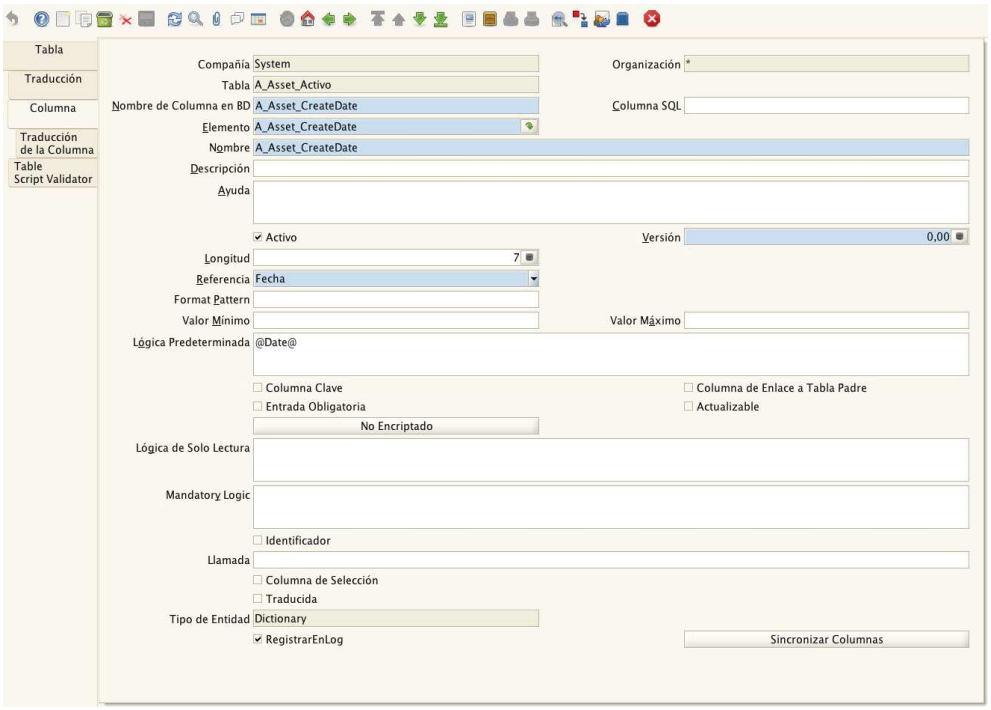
C: Es el core de Adempiere.

CM: Están basadas en el web de Adempiere. FACT_ACCT: Son todos los asientos contables. GL: Contabilidad.

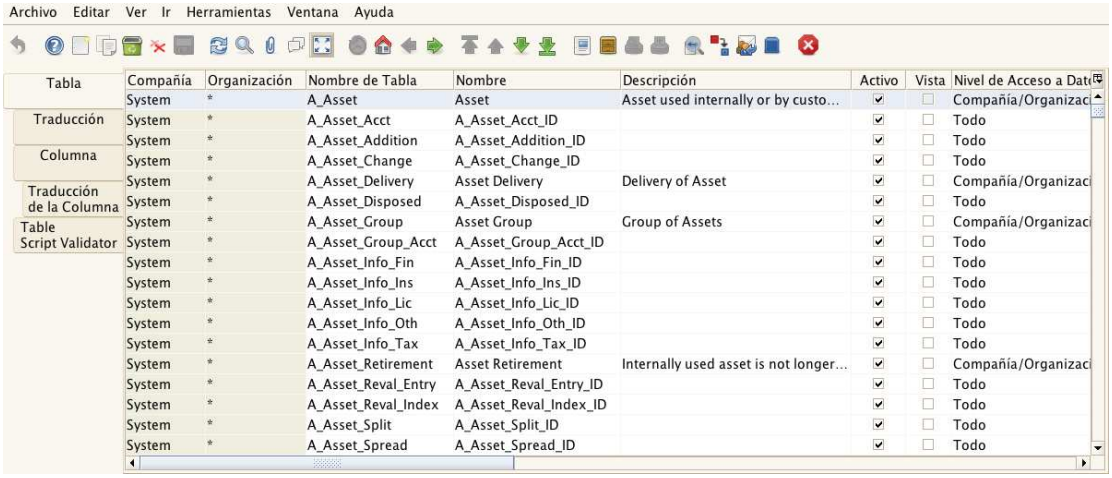
I: Importaciones.

M: Es el prefijo de material

La columna se representa como se muestra a continuación:



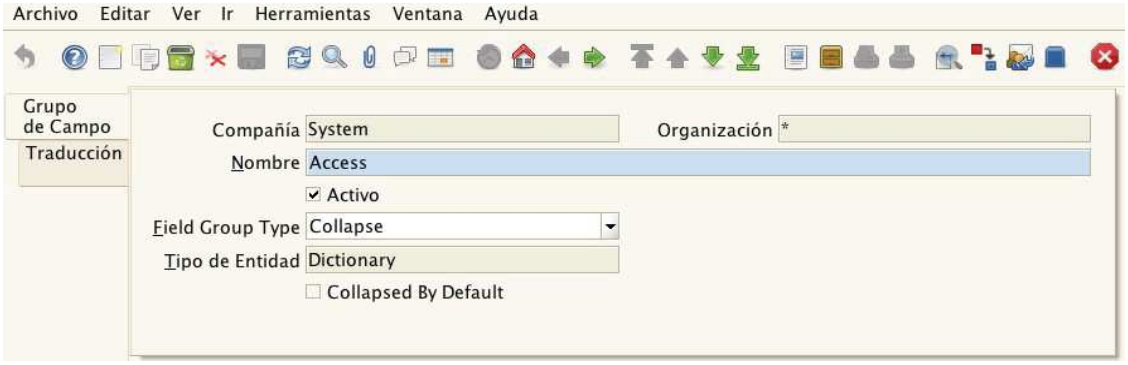
Otra forma de visualizar la ventana de la columna, es de la siguiente manera, para eso hay que oprimir el botón de detalle.



Grupo de Campo

En esta ventana se dan alta los grupos de campo que sirven para agrupar los campos dentro de las ventanas, por ejemplo para poner los totales de una orden bajo una línea con un subtitulo para separarlos de los demás elementos

de la ventana.

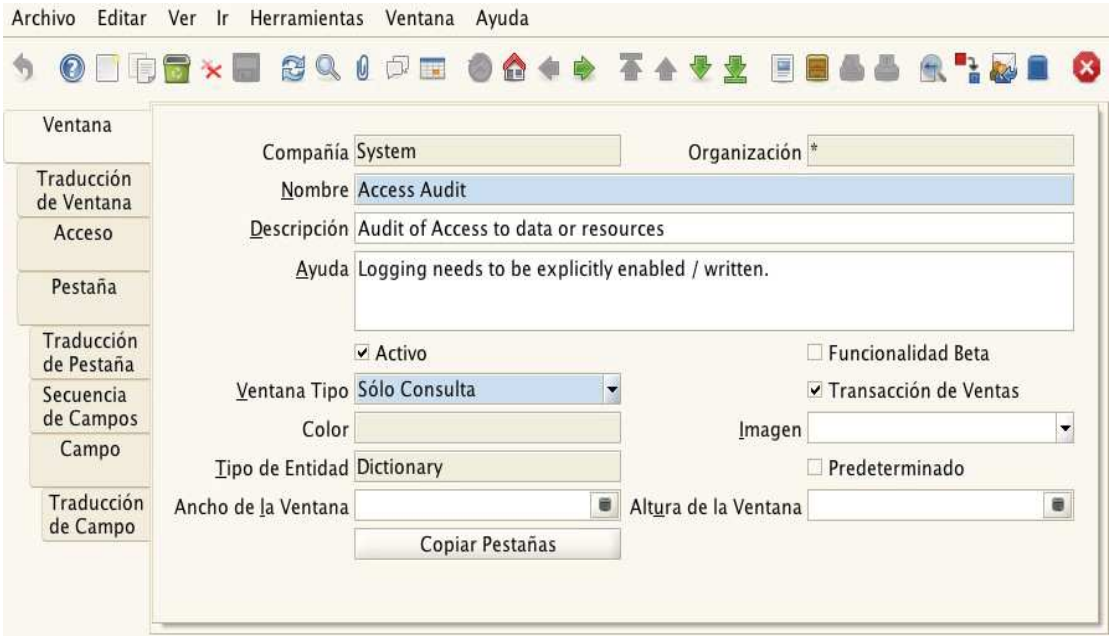


Client: Se tiene que escribir el nombre con el cual ingresamos, tal y como se muestra en la imagen.

Name: En esta opción, es el nombre de la pestaña que se muestra en la tabla de **Windw, tab y field**. Los cuales se muestran en la siguiente sección.

Ventana, Pestaña y Campo

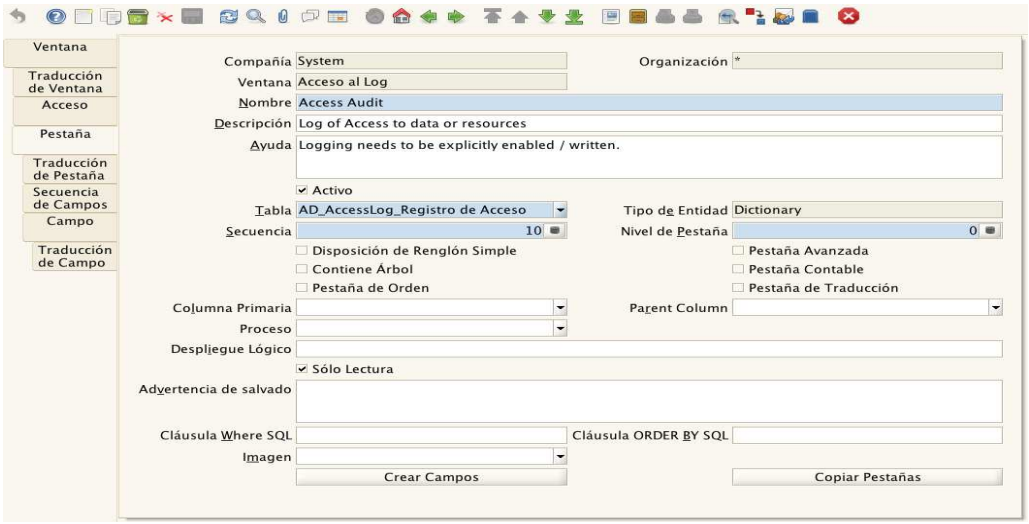
Esta opción nos permite crear ventanas que estarán en nuestra aplicación, consta de 5 pestañas principales que son Window, Access, Tab, Field Sequence y Field. En la opción de Window se da de alta la ventana la cual incluirá las pestañas y campos.



Los comandos siguientes realizan funciones similares a los mostrados anteriormente en el manual.

WindowType: Proporciona tres opciones de las cuales nosotros, entrando en forma de System Administration, tenemos la posibilidad de cambiarlas.
System Color: Permite elegir el tipo de color que las ventanas tendrán.
Imagen: Da la oportunidad de poner imágenes prediseñadas o otorgadas por el usuario, a las pestañas del programa.
Entity Type: Otorga el nombre de la aplicación que estamos usando.
Windows Width and Windows Height: Permiten modificar el tamaño de la ventana.
El botón de Copy Window Tabs: Es un proceso que crea los Tabs en la ventana que esta la base de datos; cuando se ejecuta nos pide la ventana de la cual deseamos copiar los datos; La pestaña de Tab se muestra enseguida, después de la vista de Window

En la opción de Tab se dan de alta las pestañas que tendrá la ventana seleccionada, la tabla de donde obtendrá los campos, el nivel de la pestaña (0 es el nivel principal), se incluirá alguna otra pestaña con cláusulas de where sql que funciona como un filtro de los registros y la cláusula de ordenamiento que también es un comando sql que determina el orden en el que van a ser desplegados los registros.



Sequence: Es el rango de secuencia con lo que aparecen los tab.

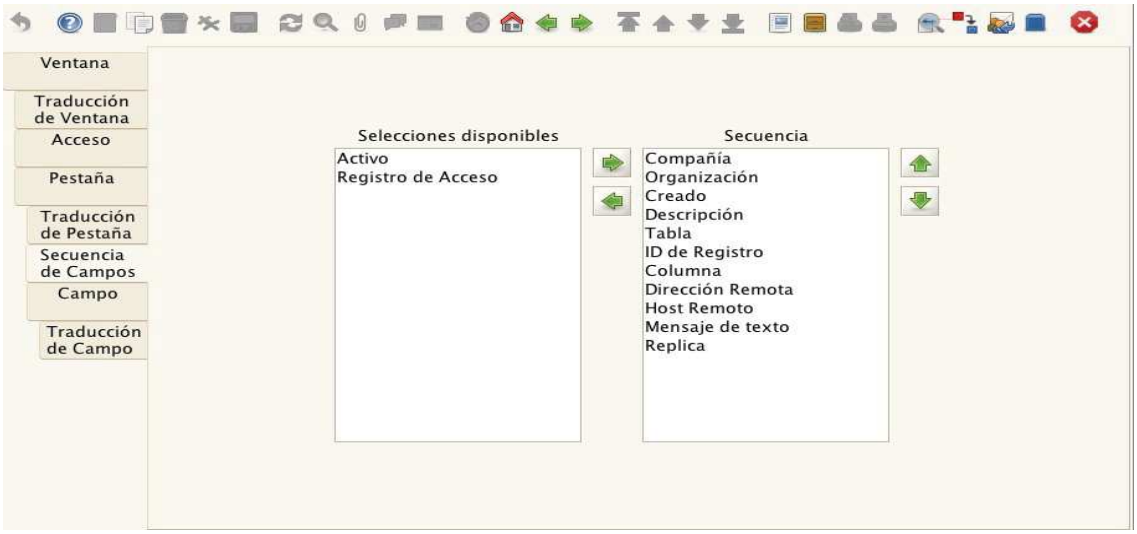
Advanced Tab: Para tener un Tab avanzado, es necesario tener habilitada esta opción y la que permite que esta pueda ser seleccionada, esto mismo ocurre con las opciones de Accounting Tab, Translation Tab.

Has Tree: Es usado para poner el árbol en ventanas.

Read Only: Si esta opción esta activada, los campos de Access por ejemplo, no se pueden modificar ya que están en forma de solo lectura.

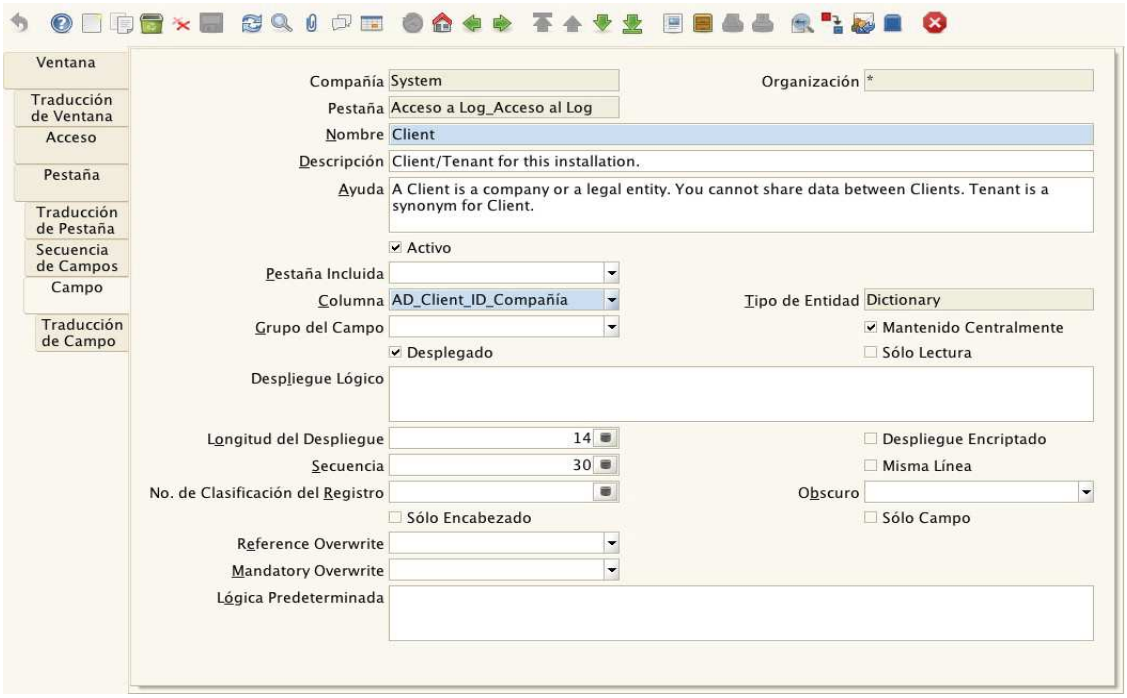
Order Tab: Es el orden de cómo se presentan las pestañas, un ejemplo seria la de order que es una pestaña padre por lo cual tiene que estar al principio, y las demás pestañas que son hijos le seguirán dependiendo del Order Tab que se establezca

En la pestaña de secuencia de comandos, se define que campos de la tabla serán desplegados y cuales no, y en que orden aparecerán en la ventana; para poder hacer esto es necesario seleccionar de la tabla de lado izquierdo la opción que necesitemos que este en una tabla y con las flechas que indican el cambio, hacer le mismo.



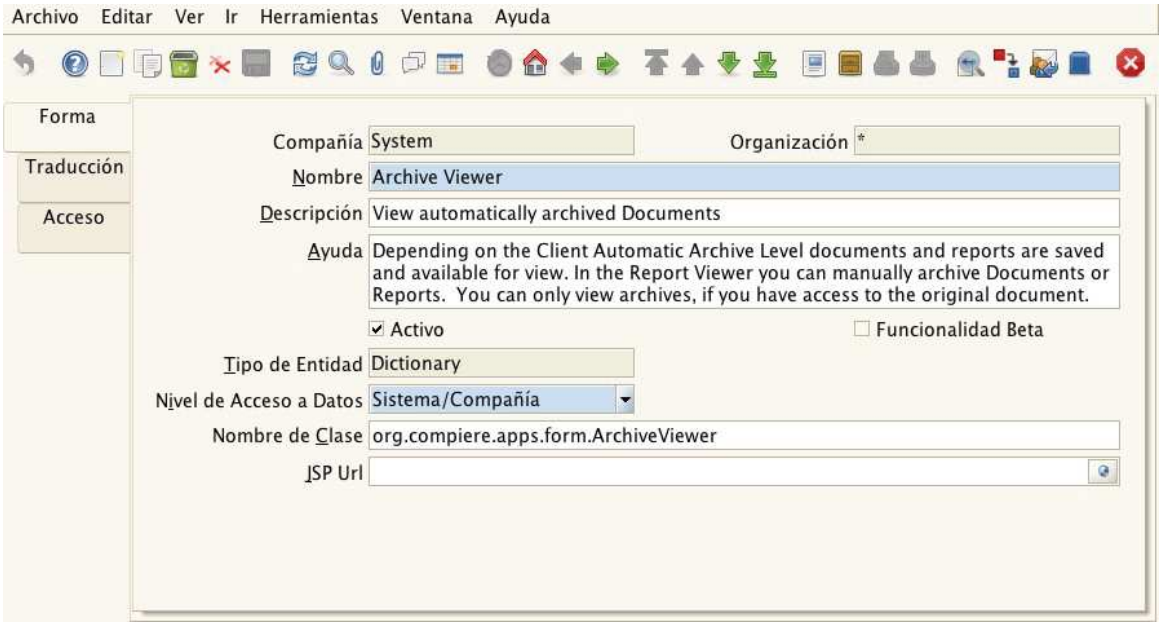
En la opción de campo se define como será desplegado el campo, como será agrupado de acuerdo al tipo de campo, si es de solo lectura, el campo de lógica de despliegue, establece si un campo va a ser desplegado o no en base a

ciertas condiciones de otras variables, el de misma línea sirve para acomodar dos campos en una misma línea en la ventana.



Formas

En esta opción se define cualquier ventana que no es construida automáticamente y se tiene que llamar a una clase para que sea ejecutada y construya la ventana.

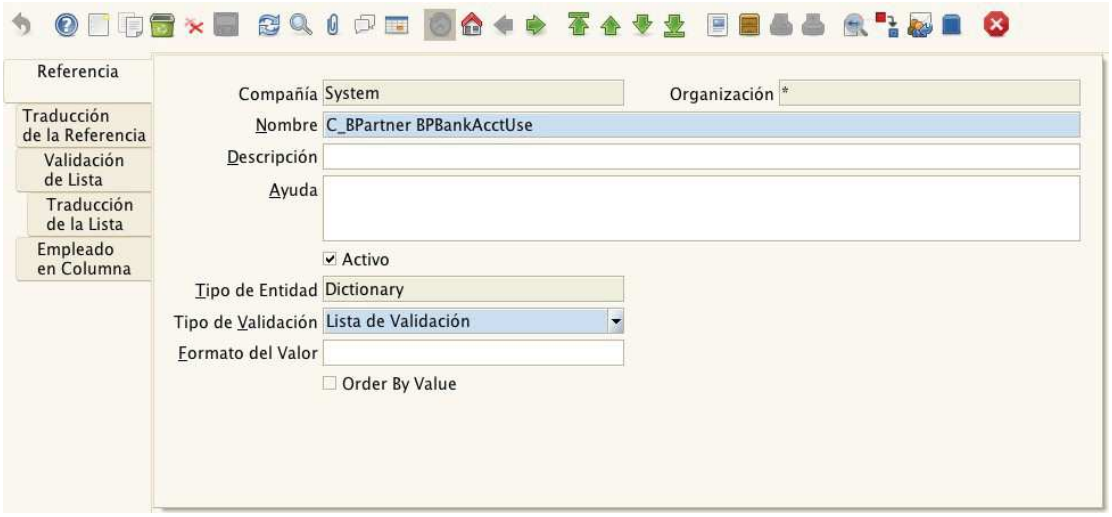


Las formas se extienden de la clase Cpanel e implementan el método FormPanel, las formas deben de tener el siguiente método para que se pueda inicializar y llamar a los demás métodos.

```
public void init (int WindowNo, FormFrame frame)
{
Log.trace(Log.l1_User, "VBOMDrop.init");
m_WindowNo = WindowNo;
m_frame = frame;
try
{

//      Parte superior del panel

createSelectionPanel(true, true, true);
m_frame.getContentPane().add(selectionPanel, BorderLayout.NORTH);
//      Centro createMainPanel();
CScollPane scroll = new CScollPane (this);
m_frame.getContentPane().add(scroll, BorderLayout.CENTER);
confirmPanel.addActionListener(this);
//      Parte inferior

m_frame.getContentPane().add(confirmPanel, BorderLayout.SOUTH);
}

catch(Exception e)
{

Log.error("VBOMDrop.init", e);
}
sizeIt();
}       //      init
```

Referencias

En la siguiente ventana, puede haber dos tipos de referencia, de lista y de tabla, la de lista se utiliza cuando es llamada desde una variable tipo lista, se utiliza para crear las listas y sus elementos.



Validation Type: Permite ver la validación de las tablas, para eso hay que cambiar a una forma de lista, la cual viene en las opciones.

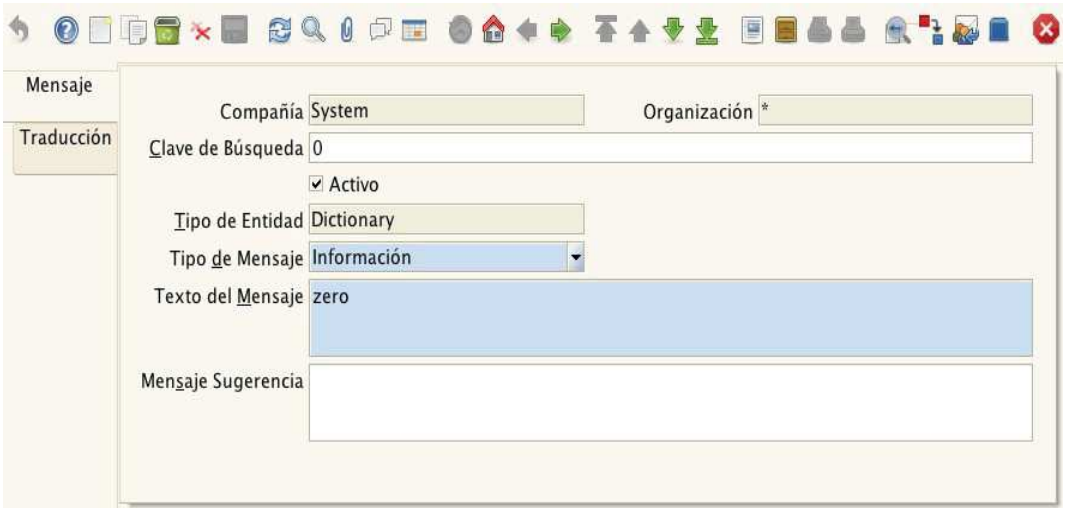
Los cuadros del lado izquierdo de la ventana de arriban muestran una figura la cual se les puso en la opción de imagen, tal y como se explico anteriormente.

Reglas de Validacion

Podremos definir todas las reglas dinámicas usadas en la introducción de datos y mantenimiento de columnas y campos. Estas reglas definen como un campo es determinado para ser valido, se pueden usar variables para validación dinámica.

Mensaje

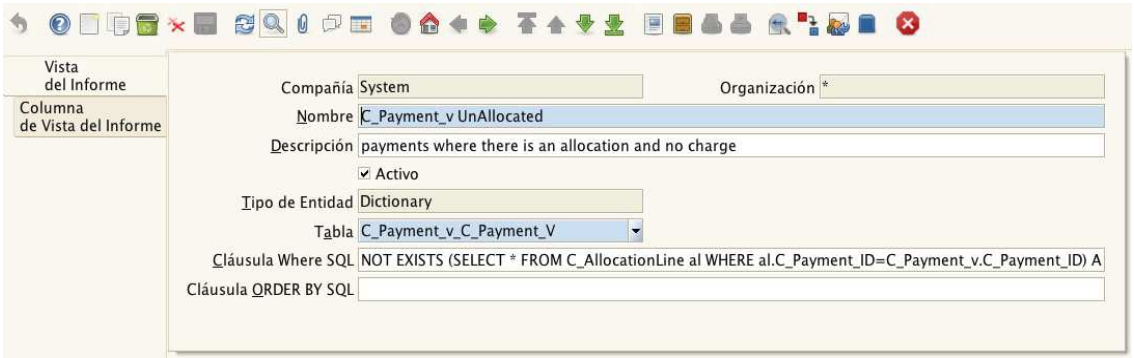
En esta ventana se define el mensaje de texto o tips para cada uno de los mensajes definidos por el sistema.



Message Type: Es un mensaje informativo, ayuda para saber si es un elemento del menú o es un error
Message Text: Es el texto que contiene el mensaje.
Message Tip: Es un texto adicional a la ayuda

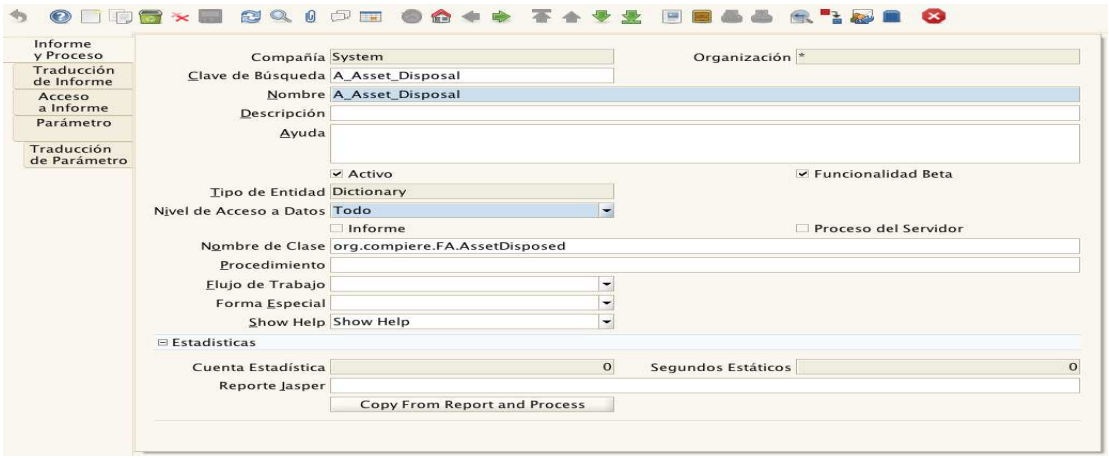
Vista de Reportes

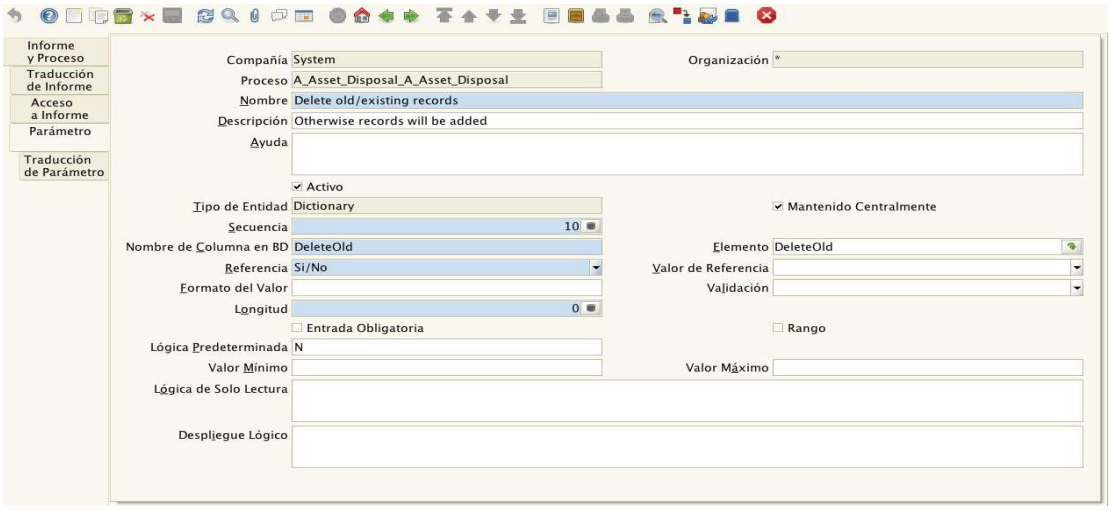
Son definidas las vistas usadas para la generación de reportes, las cuales son en base, en tabla o en a un view en sql, también se puede definir un filtro con una cláusula where sql y una cláusula order by sql.



Reportes y Procesos

Es usada para otorgar los parámetros y reglas de acceso para cada reporte o proceso en el sistema.





Formas de Impresión y personalización de reportes

Personalización de reportes

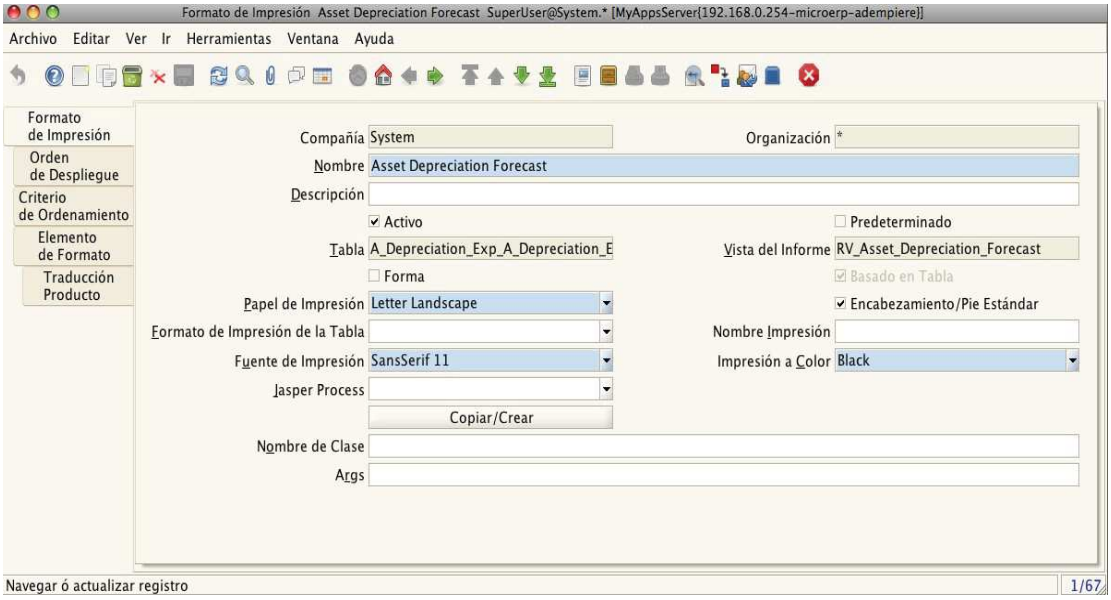
Adempiere le permite personalizar sus reportes adaptándolos a sus necesidades; Los elementos siguientes son los que se nos permite cambiar:

- Orden de Despliegue
- Clasificación
- Formato
    - Campo, Imagen, Texto, Formato de Impresión
    - %Alineación Ancho/Alto
    - Calcular: Suma – Promedio – Cuenta
    - Agrupado por (Requiere Clasificación)

Formato de Impresión

Para poder entrar al *Print Fortmat* es necesario abrir la ventana **Formato de Impresión** para personalizar su reporte con clic en **Reglas Generales < Impresión** o clic en el botón  Personalizar Reporte.

Nota: Cuando se abre el Formato de Impresión usando la opción del menú debe cerrar y reabrir su reporte para ver los cambios que haya hecho. Si abre el Formato de Impresión usando el botón Personalización de Reporte su reporte será re-desplegado con los cambios hechos cuando cierra el Formato de Impresión.



Sr ve el **Nombre** del Reporte, que corresponde a la **Tabla** en la cual reside el campo.
El campo **Vista del Reporte** indica la vista a ser usada para generar este reporte.

Quite la opción del cuadro de verificación **Forma** para imprimir un reporte de lista con columnas. Selecciónelo para imprimir una forma. La personalización de las formas será discutido en la siguiente sección, **Personalización de Formas**. Una forma tiene elementos individuales con información de la disposición (ejemplo: factura, cheque). Un reporte de lista columnas tiene columnas independientes (ejemplo: lista de facturas o líneas de inventario).

El Formato de Tabla de Impresión determina Fuentes, Colores de la Tabla Impresa.

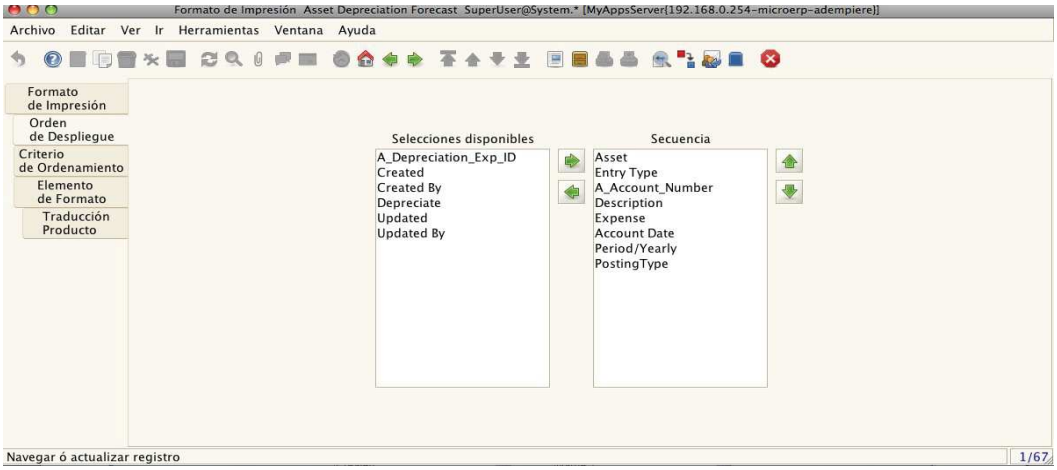
Seleccione el cuadro de verificación Cabecera/Pie Estándar para indicar que cabecera y pie de página estándar será usada. Si no desea usar una cabecera estándar, deberá definirla explícitamente.

El Papel de Impresión define tamaño, orientación y márgenes del documento.

Los botones Copiar/Crear le permiten copiar un formato de Impresión existente o crear un nuevo
Formato de Impresión desde una tabla.

Orden de Despliegue

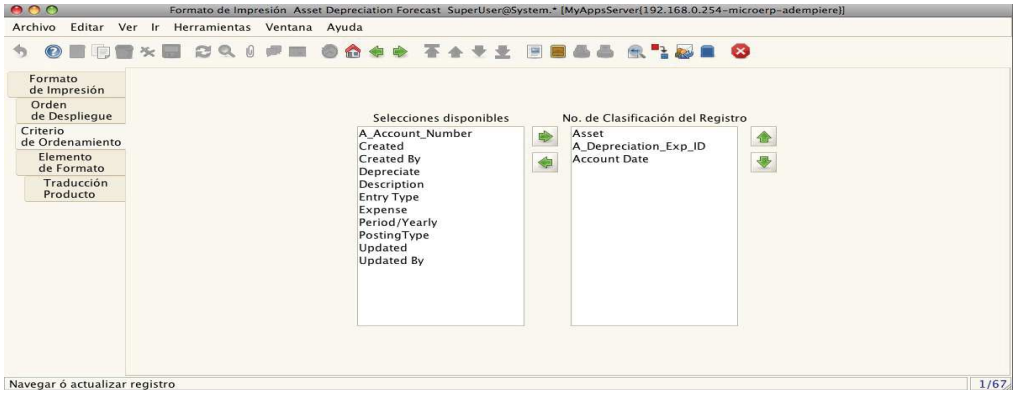
Seleccione la pestaña Orden de Despliegue para cambiar el Orden de Despliegue de los elementos en el reporte y adáptelos a sus necesidades:



Para añadir un elemento al reporte, haga clic en él, en la lista de Selecciones Disponibles y muévalo a la lista Secuencia por medio de clic en la flecha derecha. Cambia la secuencia de un elemento seleccionándolo y usando las flechas Arriba y Abajo. Para remover un elemento del reporte selecciónelo y de clic en la flecha izquierda. Salve sus cambios. Los cambios sarán efectivos inmediatamente en su *Reporte.*

Criterio de Ordenamiento

Puede seleccionar la pestaña Criterio de Ordenamiento para cambiar el orden de clasificación y adoptarlo a sus necesidades:

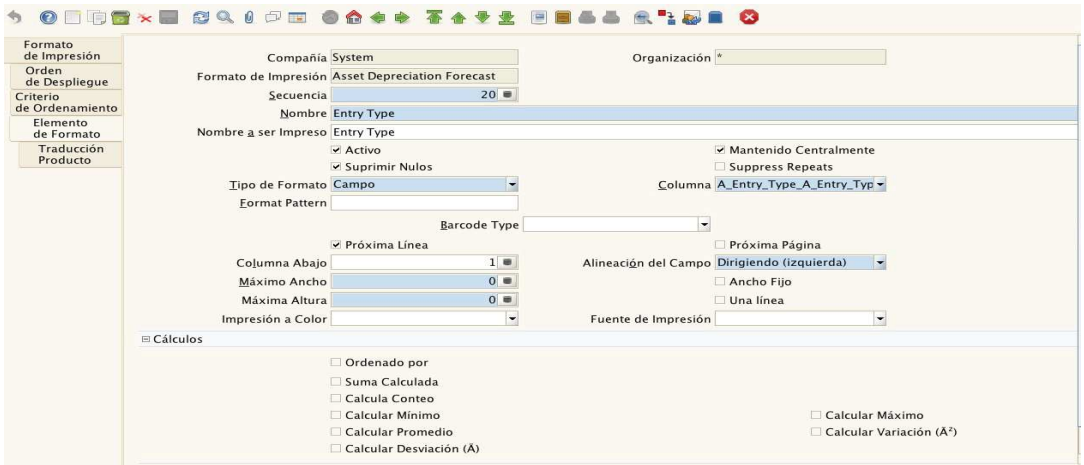


Para añadir un elemento al campo No. de Clasificación del Registro clic en la lista de Selecciones Disponibles y muévalo al campo No. de Clasificación del Registro haciendo clic en la flecha derecha. Cambia el No. de Clasificación del Registro de un elemento seleccionándolo y usando las flechas Arriba y Abajo. Salve sus cambios.

Solamente insertaría campos que necesitaran ser clasificados, por ejemplo el Name.

Elemento de Formato

Muévase a la pestaña Elemento de Formato para ver que elementos están en su Reporte. Los elementos listados corresponden a la lista de Secuencia en la pestaña Oren de Despliegue:



La Secuencia indica el orden de este elemento de Formato.
Cambia el Nombre a ser Impreso si desea que otro término sea usado en su Reporte.
El cuadro de verificación Impreso indica si esta columna se desplegará e imprimirá en el Reporte. Seleccione el cuadro de verificación Suprimir Nulos para indicar que esta columna no se desplegará si no
hay valores en ella.
El Tipo de Formato le dice que formato tiene el elemento, por ejemplo, Campo, Imagen, Texto, Formato de Impresión.
Tiene varias opciones para dar formato a sus elementos: De entrada al Ancho Máximo del elemento en 1/72 de pulgada (punto). El valor es igual a cero (0), no hay restricción en el ancho.
Selección el cuadro de verificación Ancho Fijo para indicar que la columna tiene un ancho fijo independiente del contenido.
De entrada a la Altura Máxima del elemento en 1/72 de pulgada (punto). El valor es igual a cero (0), no hay restricción en la altura.
Selección el cuadro de verificación una línea para indicar que solamente debe imprimirse una línea. La columna tiene una restricción de ancho, el texto se divide en líneas múltiples. Si se selecciona una línea, se imprime solamente la primera línea.
Seleccione un Color de Impresión para indicar que valor tiene que ser usado para indicar el valor en el elemento.
Seleccione una Fuente de Impresión para indicar la fuente que se usa para indicar el valor del elemento. Seleccione el cuadro de verificación Ordenado por para indicar que los registros sean ordenados por el
valor de esta columna. Si una columna se utiliza para agrupar, necesita estar incluida en el orden de clasificación.
El No. de Clasificación del Registro determina en que orden se despliegan los registros.
Use el campo Agrupado por para agrupar elementos diferentes. La agrupación le permite indicar sub.- totales. Si un grupo cambia, se implementan los totales.

Las columnas Agrupado por necesitan estar incluidas en el orden de clasificación.

Selección el cuadro de verificación Calcular Suma para indicar que la suma total de datos tiene que ser calculada si los datos son numéricos. Si no, la longitud de la suma total del campo será calculada.

Si el cuadro de verificación Calcular Suma se selecciona; El cuadro de verificación Total por Pagina es desplegada. Seleccione este cuadro de verificación para indicar que los totales por página deben ser desplegados. Introduzca el número de Líneas por Página para indicar el número de líneas a indicar en el reporte antes de desplegar un Total por Página.

Seleccione el cuadro de verificación Calcular Conteo para indicar que el número total de elementos NO
vacíos tiene que ser calculado (máximo es el número de líneas).

Selección el cuadro de verificación Calcular Mínimo para indicar que el valor mínimo debe desplegarse. Seleccione el cuadro de verificación Calcular Máximo para indicar que el valor máximo debe
desplegarse.

Selección el cuadro de verificación Calcular Media para indicar que tiene que ser calculado el promedio de los datos si este campo es numérico. Si no, la longitud de promoción del campo sera calculado.

Seleccione el cuadro de verificación Calcular Varianza para indicar que tiene que ser calculada la varianza de los datos si este campo es numérico. La varianza se calcula como el promedio de las diferencias de los registros de datos y su media elevados al cuadrado.

Seleccione el cuadro de verificación Calcular Desviación para indicar que tiene que ser calculada la desviación de los datos si este campo es numérico. Esta se calcula como la raíz cuadrada de la varianza.

Si selecciona Imagen como su Tipo de Formato tendrá que indicar si la Imagen es Adjunta. Si no, introduce el URL de la Imagen. La imagen no se almacena en la base de datos, sino que se recupera en tiempo de ejecución. La imagen puede ser un archivo GIF, JPEG, o PNG.

Traducción de Elemento

Clic en la pestaña Traducción de Elemento para dar entrada a una traducción de su elemento.

Nota: La pestaña Traducción se desplegará solamente si lo indica en sus preferencias.