Universidad Nacional de La Plata

Facultad de Informática

Autogeneración dinámica de Interfaz de Usuario a partir del modelo de datos

Autor: Jorge A. Vidal Castilla

Nº Alumno 1616/5

Directores: Mg. Rodolfo Bertone – Lic. Raúl Champredonde

"Como disminuir el trabajo necesario en Delphi para la creación de aplicaciones"

Índice

1. Introducción. Motivación

En el proceso de ingeniería de software, el desarrollo de pantallas para ingreso de datos es una tarea que suele ser repetitiva y que consume mucho tiempo de programación poco específico en relación al dominio de la aplicación.

Si bien cada aplicación tiene características distintivas que definen su interfaz, el proceso de generación de ABM para las tablas del modelo es un proceso reiterativo que, en general, debería ser resuelto bajo los mismos patrones de diseño.

Disponer de un framework que abrevie, abstraiga y estandarice dicho proceso de programación redundaría en una mayor productividad a la vez que contribuiría a reducir los errores de programación y homogeneizar la interfaz con el usuario.

Además, otra motivación interesante para el desarrollo del framework consiste en poder generar en tiempo de ejecución esta interfaz de usuario. De esta forma dinámica de generación, se desprende como conclusión inmediata que los costos de mantenimiento ser reducen, debido a que cualquier cambio introducido por el diseñador de la BD sobre el modelo serán embebidos rápidamente por el entorno a desarrollar. Esta reducción del tiempo de respuesta brindado por el desarrollador ante un requerimiento de un cliente será beneficioso para todos los actores del sistema.

Si bien existen herramientas para la generación automática de interfaces de usuario en Delphi, todas las encontradas se basan en la generación de código estático a compilar dentro de la aplicación. Al generar código estáticamente, la inserción, modificación y eliminación de campos y tablas no se pude hacer en tiempo de ejecución, necesitando, al menos, la ejecución del generador de código y la compilación del mismo. Una herramienta que genere la interfaz de usuario de forma dinámica, en la cual se puedan agregar y eliminar campos sin necesidad de modificar código fuente y por lo tanto de compilar dicho código, sería un interesante aporte a la flexibilidad y velocidad del desarrollo de software.

Otra ganancia derivada de esta generación dinámica de la interfaz de usuario es obtener un estilo de interfaz de usuario consistente y de estilo uniforme. Interfaces de usuario “consistentes y uniformes” permiten que el usuario se concentre en el manejo adecuado de la información y no distraiga su atención en cuestiones de representación visual de la misma. Este objetivo, que es importante para el usuario, suele ser una tarea ingrata para los programadores, que encuentran algunas tareas de generación de

interfaces de usuario altamente repetitivas. La consistencia y la uniformidad de la interfaz de usuario se vuelve aún más difícil de lograr en grupos de trabajos heterogéneos, donde las habilidades de los programadores son dispares.

1.1. Delphi, un poco de historia

Delphi es por naturaleza un compilador Pascal. Delphi 2006 es un eslabón más en la cadena de productos de Borland desde que Anders Helsberg escribiera el primer compilador Turbo Pascal hace más de 20 años. Se desataca por su velocidad de compilación y su estabilidad. Con sus orígenes en el Turbo Pascal para DOS, Delphi ha ido atravesando diferentes versiones, empezando por la 1 hasta la actual versión 2006, planteando una fuerte integración con el .NET framework de Microsoft. Ha habido, a lo largo de su historia una competencia con Visual Basic, forzando a que cada producto mejore a lo largo de sus versiones.

El IDE *(Integrated Development Enviroment)* tiene como componentes principales: la ventana principal, la Paleta de componentes, las barras de herramientas, el Diseñador de formularios, el Editor de código, el Inspector de objetos y el Explorador de código.

El lenguaje Object Pascal usado en Delphi tiene algunas similitudes con Java: carece de herencia múltiple, pero una clase pude implementar múltiples interfaces. Ambos carecen de sobrecarga de operadores pero si de métodos.

Delphi provee un mecanismo para obtener información de tipos y clases en tiempo de ejecución a través de su RTTI *(run time type information)*.

1.2. Generación dinámica de interfaces de usuario

Según el diccionario la real academia española [RAE]:

dinámico, ca.
(Del gr. δυναμικός, de δύναμις, fuerza).

1. adj. Perteneciente o relativo a la fuerza cuando produce movimiento.
2. adj. Perteneciente o relativo a la dinámica.
3. adj. coloq. Dicho de una persona: Notable por su energía y actividad.

4. f**.** Parte de la mecánica que trata de las leyes del movimiento en relación con las fuerzas que lo producen.
5. f. Sistema de fuerzas dirigidas a un fin.
6. f. Nivel de intensidad de una actividad.

interfaz.
(Del ingl. interface, superficie de contacto).
1. f. Inform. Conexión física y funcional entre dos aparatos o sistemas independientes.

Por lo tanto, la **interfaz de usuario** es la forma en que los usuarios pueden comunicarse con una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo. [Wiki]

Al agregar el componente "dinámico" a la interfaz de usuario, se hace referencia a los puntos 5 y 6 de la definición, haciendo hincapié en el hecho que la interfaz estará fuertemente orientada a un fin, y para alcanzarlo realizará actividades intensas de procesamiento. Dichas actividades se realizan "al vuelo" y son basadas en decisiones que toma el programa durante la ejecución, no antes. Contrasta con estático [FREE2].

La interfaz de usuario determina la interacción entre el usuario y el sistema. Cualquier sistema con el cual una persona interactúe posee una interfaz de usuario, por más primitiva que sea. La facilidad de uso y la «consistencia» de la misma pueden llegar a determinar el éxito o fracaso de un sistema. En el ámbito de este trabajo, se define una interfaz de usuario como "dinámica" cuando su contenido es generado durante la ejecución del programa. Es decir, en el momento de presentar la pantalla al usuario, el programa determinará qué datos y en qué ubicación se deben presentar los mismos al usuario.

Los lineamientos que hacen una interfaz de usuario consistente:

- ✓ Las etiquetas e iconos siempre deben significar lo mismo, una cosa u operación siempre se debe representar de la misma manera.
- ✓ Los objetos o controles deben ser desplegados en la interfaz de usuario de modo consistente, manteniendo siempre una convención sobre el lugar donde se encontrarán.
- ✓ Ciertos objetos se deben emplear como puntos de referencia, estando siempre disponibles en el mismo lugar. El usuario los utilizará como

punto inicial de la navegación. Por ejemplo, el menú de "Archivo" siempre se debe encontrar en la parte superior izquierda de la pantalla.

1.3. Trabajo relacionado

La generación dinámica de interfaces de usuario está íntimamente relacionada con una adecuada formalización de las necesidades del usuario con respecto las tareas que debe desempeñar. Es claro que el objetivo del usuario de un sistema no es utilizar el sistema, sino realizar su trabajo "lo más rápidamente posible", para tener tiempo disponible para emplear en sus objetivos personales [Copper 99].

Para la formalización de las interfaces de usuario, se han desarrollado métodos para la detección y diseño de ventanas. Las metodologías para detección y definición de ventanas se pueden diferenciar entre las basadas en el modelo de datos y las basadas en las tareas que el usuario debe desempeñar.

Entre los basados en modelos de datos, encontramos los siguientes:

- ✓ GENIUS [Jansen et. Al., 1993] Asigna automáticamente una ventana a cada vista definida en el modelo de datos.

- ✓ JANUS [Balzert, 1993] Asigna una ventana para cada clase de objeto encontrado en el modelo orientado a objetos. Partiendo del análisis orientado a objetos, genera la interfaz de usuario correspondiente. Utilizando los atributos, los métodos, la herencia y las relaciones de agregación entre las clases, define dos formularios para cada clase:
  - o Uno modo "edición" con todos los atributos de la misma y botones de acción para sus métodos
  - o Uno modo "tabla", que muestra en cada fila de la tabla una instancia de la clase, junto a todos los servicios de clase disponibles.

- ✓ Mecano [Puerta 1996] Define un lenguaje de modelado de interfaces de usuario (llamado MIMIC), con el cual se pueden especificar tanto los aspectos generales como particulares. También provee un modelo genérico de interfaz, llamado MIM. MMIC sigue los siguientes principios:
  - o Representación explícita de la organización y estructura de los modelos de interfaz.

- No hay modelo genérico único: Se descarta la idea. MIMIC soporta la definición de varios modelos, aunque se provee uno "por defecto".
- Representación explicita del diseño de la interfase. Los modelos de interfaz escritos en MIMIC definirán no solo los elementos de la misma, sino también las características del proceso de diseño de la misma.

Sin embargo, este proceso termina en algo muy similar a asignar a cada clase del dominio una ventana.

Estos tres esquemas resuelven el problema de la identificación de ventanas de forma automática. Entre las contras que encontramos con estos esquemas hallamos:

- ✓ El usuario se puede encontrar, o bien con una ventana con "demasiados" datos, o con muy pocos. Cualquiera de estos enfoques "automáticos" terminan agregando carga al usuario, ya en forma de:
  - Pantallas demasiado cargadas de datos innecesarios
  - Pantallas con pocos datos y una alta carga de navegación.
  - Por otro lado, si el usuario tiene que realizar actualizaciones parciales de datos, de todas formas debe trabajar con las ventanas completas, obligándolo a ver muchos más datos de los que necesita.

Entre las metodologías basadas en las tareas encontramos:

- ✓ TRIDENT [Bodart 1994], presenta un enfoque interesante a la hora de definir las ventanas y el comportamiento del sistema. Se basa en la identificación de tareas, con lo que se obtiene un modelo interactivo de las tareas que el usuario debe realizar. El modelo se va refinando en sucesivas iteraciones, identificando las subtareas que componen cada tarea. Dicho modelo puede representarse en un gráfico de "encadenamiento de tareas". También provee un mecanismo para la identificación de las ventanas (que en la jerga de la metodología se llaman "unidades de presentación" [Bodart 1995]) y de los procesos que esta debe efectuar (procesos que pueden ser tanto de ingreso/egreso de datos, como de validación).

Si bien un análisis a fondo de la metodología TRIDENT está más allá del ámbito de este trabajo, cabe recalcar que el empleo de ésta metodología junto con la herramienta propuesta pueden llegar a brindar un alto nivel de productividad y una mejora en la percepción del usuario con respecto a la utilidad del sistema.

1.4. Generadores de código existentes

Un generador de código automático es una herramienta que deriva, a partir de determinados patrones, el código fuente de una aplicación. El uso de estas herramientas reduce el tiempo que se necesita para el desarrollo del software como así también asegura que el grado de errores de programación permanezca acotado, reduciendo por tanto los tiempos de depuración y puesta a punto. Los 4GL constan de procedimientos que generan el código fuente en función de lo expresado en el diseño de la aplicación o modelo de datos. Para esto, el usuario especifica la funcionalidad del programa, o parte del mismo, y la herramienta determina cómo realizar dicha tarea. [Walsamakis]

1.4.1. GeneXus

GeneXus es una herramienta para el desarrollo de software basado en requerimientos. Captura el conocimiento del usuario en diversos objetos: Transacciones, Reportes, Procedimientos, Work Panels, Web Panels, Temas, Menús, Data Views y Transacciones de BI.

La tarea del analista/programador consiste en identificar dichos objetos junto al usuario y definirlos en el sistema. Con estas definiciones GeneXus conforma una base de conocimiento que luego utiliza para definir los modelos de datos físicos (base de datos) y las aplicaciones.

Se garantiza que la base de datos generada está en tercera forma normal, aunque luego el analista puede introducir redundancias de ser necesario.

Como la definición de la aplicación en si se hace desde la base de conocimiento, en forma abstracta, la aplicación puede generarse para distintas plataformas objetivo.

Para validar los requerimientos, GeneXus provee una ambiente de prototipado, con el cual se puede verificar la aplicación a generar, que es funcionalmente equivalente a la aplicación definitiva. Esto acelera el ciclo Diseño -> Prototipo -> Producción. La idea fundamental de la herramienta es que moverse dentro de este ciclo no debe ser costoso,

y que obteniendo rápidamente prototipos funcionales para validar con el usuario, el producto final será satisfactorio.

1.4.2. Clarion

Clarion es un producto para desarrollo de aplicaciones de bases de datos. El desarrollo se hace utilizando un lenguaje propietario de cuarta generación (4GL), que se traduce y compila tanto en C++ como en Modula-2.

El producto almacena en sus "metabases":

- ✓ Metadatos (lógica de negocios y esquema de base de datos)
- ✓ Lógica de aplicación
- ✓ Interfaces de usuario

Las metabases son llamadas "Diccionario de datos" y "Registro de modelos".

Las aplicaciones son generadas a través de un generador de código que utiliza como entrada principal la metabase. Debido a esta generación de código unificada desde los metadatos, las aplicaciones generadas tienen el mismo *look and feel,* asegurando consistencia entre las aplicaciones.

Se proveen modelos o *"templates"* que facilitan la tarea de programación, por ejemplo para manejo de grillas maestro/detalle. Se podría decir que la tarea del programador consiste en elegir las tablas, correr el *"Wizard"* y luego personalizar el código generado. En caso de haberse explicitado relaciones en los metadatos, la interfaz de usuario presentará los medios adecuados para seleccionar los registros de la tabla de relación.

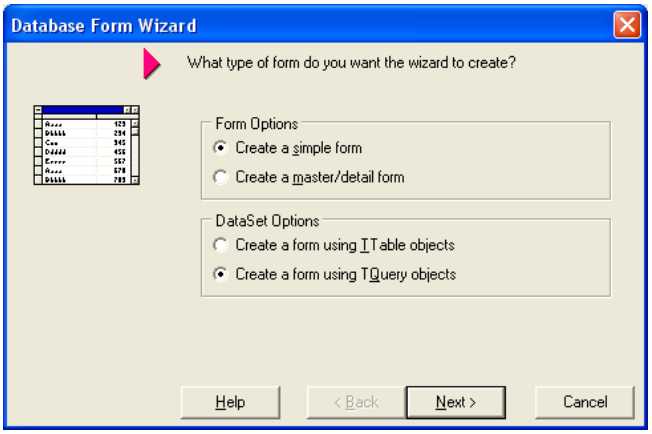
Luego de personalizar el código de la aplicación, se genera la misma.

Nuevamente encontramos la misma limitación que en GeneXus: ante el requerimiento de una modificación en la aplicación, la misma debe ser recompilada y distribuida nuevamente. La interfaz de usuario se genera a partir de los metadatos, pero al consultar los metadatos solo para la generación inicial de la aplicación, una modificación de los mimos requiere recompilar.

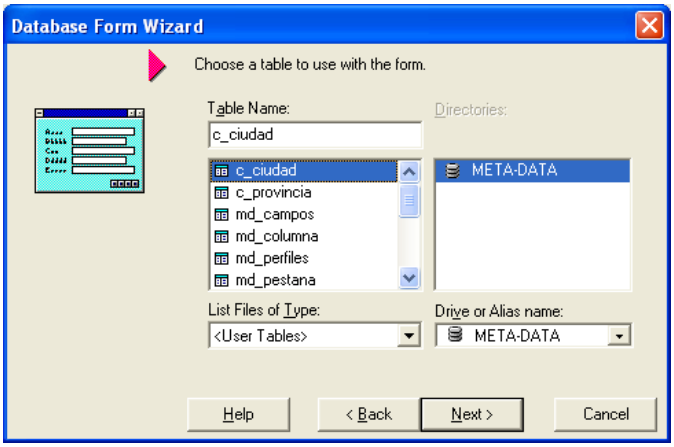
1.4.3. Delphi Wizards

El IDE de Delphi incluye un wizard o asistente para generar pantallas o formularios. Dicho asistente, llamado “Form Wizard” se puede utilizar para crear pantallas de alta, baja y modificación de registros de una base de datos.

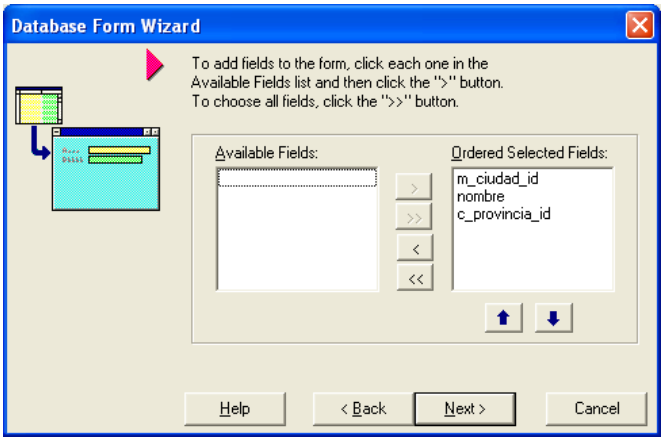
Paso 1: Elección del tipo de formulario a generar



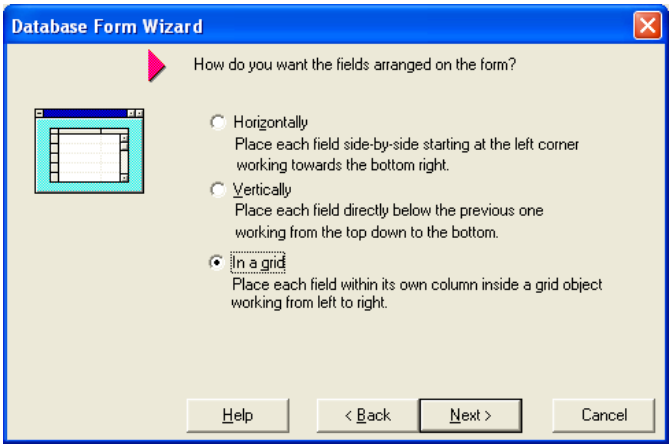
Paso 2: Selección de la tabla de origen de datos



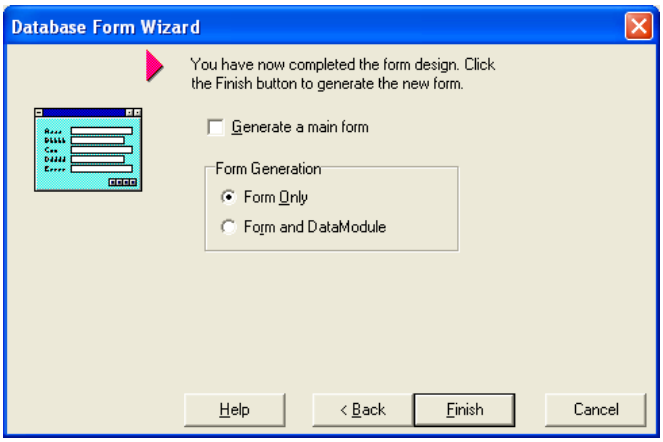
Paso 3: Selección de los campos a mostrar



Paso 4: Selección del estilo de visualización de registros



Paso 5: Confirmación y opciones finales sobre el formulario a crear



El resultado de ejecutar el asistente es una unidad .PAS que se incluye en el proyecto Delphi en uso. Para utilizarse, se debe compilar junto a toda la aplicación, y se debe agregar el código necesario para su invocación.

Como se verá luego, la mayoría de las opciones presentadas por este asistente, estarán disponibles en el framework presentado.

Este asistente no presenta ninguna de las propiedades "dinámicas" que se buscan en esta tesis: Cualquier modificación al formulario, como por ejemplo agregarle un campo nuevo, requiere codificación y luego la correspondiente recompilación de la aplicación.

Otro aspecto negativo de este asistente es que sólo se puede utilizar para la creación inicial del formulario. Una vez que está creado, todas las modificaciones se deben hacer utilizando el IDE, ya sea programando o modificando las propiedades de los objetos del formulario. Si bien ésta es una característica común a la mayoría de los asistentes de este tipo, el usuario puede "perderse" en la forma de utilizar la aplicación. Sería más sencillo para el usuario utilizar siempre la misma pantalla, ya sea para la creación de un nuevo formulario o para la modificación de uno existente.

1.4.4. PHP myEdit

PHPMyEdit es una herramienta para generación de páginas PHP para el mantenimiento de tablas MySQL. Se utiliza como una librería que provee un conjunto de herramientas para la manipulación de datos.

Entre sus cualidades se encuentran:

- Generación de código para la manipulación de datos
- Inserción, modificación y eliminación de datos.
- Paginación de tablas, orden y filtrado de datos.
- Búsquedas en otras tablas (Relaciones 1:N)
- Configuración de permisos
- Múltiples estilos de navegación
- Log de las acciones de los usuarios
- Soporte multilenguaje
- Capacidad de ser extendido

Utilizando el programa, dada una tabla del sistema, se crea la página PHP para la edición de la misma. En la página se configuran los aspectos de visualización y navegación requeridos. También se pueden agregar scripts de validación de datos (llamados triggers en la jerga de la aplicación). Existen extensiones para agregar reportes a las páginas.

Si bien la utilización de este sistema facilita mucho la tarea del programador, la solución provista difiere con el objetivo de ésta tesis debido a que, fundamentalmente, PHP es un lenguaje interpretado. Utilizando esta característica, PHPMyEdit lo que hace es generar "al vuelo" la página con el código fuente que el compilador PHP procesará para generar la respuesta a la petición del usuario. En lenguajes como Delphi, esto no es posible debido a que el código se compila estáticamente, con lo cual el código fuente que se encarga de la visualización de datos debe estar compilado de antemano.

1.5. Frameworks

En el diccionario se puede encontrar la siguiente definición de framework [FREE1]

- ✓ Una estructura para soportar o encerrar una cosa, especialmente un esqueleto usado como soporte para las bases del objeto construido.
- ✓ Una plataforma externa de trabajo
- ✓ Una estructura fundamental, por ejemplo para trabajo escrito
- ✓ Un conjunto de asunciones, conceptos, valores y prácticas que constituyen una manera de ver la realidad.

La programación orientada a objetos es comúnmente empleada con el objetivo de incrementar la reutilización de código. Aun así, el código reutilizado generalmente está basado o forma parte de un framework. Un Framework es un diseño reutilizable expresado en código. Esto permite la reutilización del diseño y también del código. El framework adecuado puede disminuir el tiempo empleado en desarrollar un artefacto de software en un orden de magnitud. A pesar de este importante beneficio, diseñar y utilizar un framework son tareas difíciles. El beneficio real solo se obtiene luego de estar adecuadamente familiarizado con el mismo.

¿Qué es exactamente un framework? Se puede pensar en un framework como un diseño reutilizable de un dominio específico. Por ejemplo, en el mundo real, se puede construir una casa usando partes reusables mucho más rápido que si las tuviera que construir de cero (por ejemplo, se compra las ventanas hechas, no se fabrican). En el mundo del software, se construyen programas utilizando objetos de un framework (por ejemplo las clases de acceso a las bases de datos en Delphi).

Al ser un elemento común a muchos programas, un framework puede evolucionar cuando los programadores y diseñadores del mismo encuentran nuevas abstracciones o conceptos que ameritan ser incluidos.

El diseño y la calidad del framework utilizado tienen un alto impacto en el producto obtenido por su utilización. Si en el ejemplo de la construcción de casas sólo dispusiéramos de ventanas de 50 cm. de lado, la casa construida se vería, por lo menos, algo extraña.

Como en el mundo real, tener las partes que componen un framework no alcanza. Un framework es un diseño reutilizable, no un conjunto de partes inconexas. Aprender a utilizar un framework implica conocer sus partes, el tipo de productos que se pueden construir con ellas; conocer como se relacionan entre ellas y conocer las propiedades que son útiles para nuestros fines.

1.6. Metadatos

Definición:

Metadatos *(del griego μετα, meta, «después de» y latín datum, «lo que se da», «dato»[COMP]), literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos refiere a un grupo de datos, llamado recurso. [Wiki metadato]*

Se podría decir que un metadato es un dato que se encarga de mantener un registro sobre el significado, contexto o propósito de un objeto informativo, de tal forma de poder descubrir, entender, extraer y administrar dicho objeto.

Los metadatos están estrechamente ligados al objeto que describen.

Las ventajas que se obtienen al usar metadatos incluyen:

- ✓ Los metadatos agregan contenido, contexto y estructura a los objetos de información, asistiendo de esta forma al proceso de recuperación de conocimiento desde colecciones de objetos.

- ✓ Los metadatos permiten un acceso a los recursos en forma controlada ya que se conoce con precisión el objeto descrito. Es posible entonces establecer sistemas de filtrado y permiten generar bases para una autentificación y mecanismos para definir grados de confianza sobre las fuentes de información.

En el entorno propuesto, se llamará "metadatos" a los datos que utilizará la herramienta para generar la interfase de usuario de forma correcta.

La relación entre un registro de metadatos y el objeto informativo al que describe puede darse de una de estas dos formas:

1. los elementos pueden estar separados del objeto informativo, como en el caso del registro de un catálogo de bibliotecas;
2. los metadatos pueden estar incluidos [embedded] en el objeto.

Tipos de metadatos y sus funciones

Los metadatos se pueden categorizar, con fines prácticos, en tres grandes grupos: descriptivos, estructurales y administrativos. Es común que un metadato pueda ubicarse en más de una categoría, lo cual hace que los límites entre ellas sean, al menos en algunos casos, difíciles de encontrar. [UCorn1]

Por ejemplo, los metadatos administrativos pueden incluir una amplia gama de información que podría ser considerada como metadatos descriptivos y estructurales.

Tipo de metadato	Objetivo	Ejemplo
Metadatos descriptivos	Descripción e identificación de recursos	Textos de ayuda, descripciones, etc.

	de información. En el nivel Aplicación, permite a los usuarios descubrir recursos (por ejemplo, que tabla se utiliza para almacenar determinados datos).	
Metadatos estructurales	Facilitan la navegación y presentación de recursos electrónicos, o elementos informativos. Proporcionan información sobre la estructura interna de los recursos, incluyendo nombres, formatos de presentación, tipos de dato, etc. Describen la relación entre los elementos del sistema. Unen los elementos del sistema formando estructuras de información complejas.	Tablas, columnas, campos.
Metadatos administrativos	Facilitan la gestión y procesamiento de la información tanto a corto como a largo plazo Incluyen datos técnicos sobre la creación y el control de calidad. Incluyen gestión de derechos y requisitos de control de acceso y utilización. Información sobre el tipo de persistencia deseada (registros temporales, persistentes, no eliminables, etc.)	Esquema de seguridad, perfiles, usuarios, etc.

1.7. Evolución de las interfaces de usuario

Las primeras computadoras tenían interfaces de usuario que eran tan rudimentarias como ellas mismas. La computadora comunicaba información al usuario a través de luces, y el usuario comunicaba información a la computadora a través de dispositivos mecánicos. Los usuarios debían ser muy entrenados para comunicarse con la computadora.

El siguiente paso evolutivo llegó con computadoras que se comunicaban con los usuarios a través de dispositivos de impresión y los usuarios a través de tarjetas perforadas. Si bien esto fue una mejora, todavía resultaba ineficiente y complicado. La comunicación todavía debía ser realizada por personal especializado.

Las interfaces de usuario abandonaron la oscuridad con la aparición de las pantallas de vídeo para la comunicación desde la computadora y los teclados similares a las máquinas de escribir para comunicarse con el ordenador. Este avance permitió que los usuarios "comunes" se comunicaran con las computadoras. La pantalla de vídeo estaba limitada a mostrar sólo los caracteres que el usuario podía ingresar, lo que se convirtió en una limitación. Los usuarios debían memorizar comandos, que generalmente estaban orientados a la computadora. Todavía se necesitaba un alto nivel de entrenamiento para utilizar la computadora.

Se podría decir que las interfaces de usuario ingresaron a la era moderna cuando un innovador grupo de diseñadores del Centro de Investigación de Xerox en Palo Alto presentaron la "Interfaz gráfica de usuario" o GUI por sus siglas en inglés (*graphical user interface)* y abandonaban paradigma orientado a caracteres. Había dos factores determinantes que diferenciaban al nuevo paradigma del antiguo: Uno era presentar información a los usuarios de forma gráfica además de textual, el otro era presentar al usuario un número finito de opciones en vez de requerirle que memorice los comandos e ingresarlos manualmente desde un universo casi ilimitado. La interfaz de usuario pasó a estar centrada en las necesidades del usuario en vez de las de la computadora. Esto redujo significativamente el nivel de entrenamiento que una persona debía tener para utilizar una computadora, y por primera vez usuarios no iniciados podían ser productivos casi inmediatamente. [UCORN2]

1.8. Interfaces de usuario

Como se ha mencionado anteriormente, la interfaz de usuario rige la comunicación hombre-máquina. Entre los aspectos fundamentales para obtener una interacción exitosa se encuentran [PRES93]:

- ✓ *Ser consistentes:* se debe ser consistente en el formato a utilizar para la presentación de menús, el ingreso de órdenes y datos y la visualización de los mismos.

- ✓ *Ofrecer una retroalimentación significativa:* Se debe proporcionar al usuario una retroalimentación visual y auditiva para asegurar una comunicación bidireccional.
- ✓ *Consultar al usuario por acciones destructivas no triviales:* El usuario debe ser advertido antes de eliminar datos o perder información.
- ✓ *Permitir la vuelta atrás en la ejecución de un programa:* Se deben poder revertir los cambios hechos. El programa debe permitir al usuario recuperarse de errores cometidos.

Entre los aspectos relevantes en la visualización de la información encontramos:

- ✓ *Mostrar solo aquella información que sea relevante en el contexto actual:* El usuario no debe tener que buscar a través de datos extraños, menús y gráficos para obtener información relevante para una función del sistema
- ✓ *No abrumar al usuario con datos*: En dónde sea posible, usar formatos que permitan una rápida asimilación de la información. Los gráficos deben reemplazar las tablas cuando sea posible.
- ✓ *Utilizar etiquetas consistentes, abreviaciones estándar y colores predecibles:* El significado de una pantalla debe ser obvio, evitando fuentes externas de información.
- ✓ *Utilizar ventanas para modularizar los diferentes tipos de información:* Las ventanas mantienen accesibles al usuario muchos tipos diferentes de información.

Claramente la usabilidad de una aplicación depende de la calidad de su interfaz de usuario. Lineamientos claros sobre el diseño de la misma son indispensables para asegurar la calidad del software.

Se puede citar un estudio de laboratorio [NIELSEN] en el cual dado un estándar de sólo 2 páginas, 26 estudiantes lograron un insuficiente 71% de apego al mismo. En dicho estudio, el 53% de los desarrolladores dijeron que las reglas eran difíciles de recordar.

Para que un estándar sobre interfaz de usuario mejore la usabilidad de un producto, deben cumplirse dos condiciones:

- ✓ El estándar debe especificar una interfaz usable
- ✓ El estándar debe ser claro y utilizable por los desarrolladores de la aplicación.

El entorno desarrollado propone un estándar para la interfaz de usuario que ayuda al desarrollador a crear aplicaciones que cumplan con los lineamientos anteriormente planteados.

1.9. Estudios sobre la arquitectura de las aplicaciones

A continuación se presentan estadísticas recabadas con el objetivo de dimensionar la relación entre la complejidad del modelo de datos y la cantidad de ventanas del sistema. Veremos como impacta la cantidad de ventanas y su dificultad en el costo del desarrollo de sistemas.

1.9.1. Estadísticas

Se proseguirá con estadísticas de sistemas a los cuales se tuvo acceso, tanto en su desarrollo como implementación. Lamentablemente, no se dispone de datos sobre la cantidad de horas que empleó en el desarrollo de cada uno. Se presenta una breve reseña de cada sistema analizado:

Gestión de controles

El sistema de gestión de controles fue desarrollado para laboratorios de análisis clínicos. Tiene como meta analizar los resultados de las muestras de control que se corren sobre los analizadores, utilizándose para validar que el analizador o método analítico empleado entregue resultados "confiables". Para esto se utilizan reglas de validación de procesos analíticos (desvío estándar, varianza, cálculo del error medio…), comunes en el ambiente químico / bioquímico. El sistema presenta los datos en forma de planillas y también como gráficos de líneas. Dentro del sistema se debe realizar una configuración que comprende: equipos analíticos, analitos, lotes de control, valores medios esperados, valores de alarma, etc. Actualmente es utilizado por aproximadamente 20 usuarios simultáneos. El sistema está orientado a usuarios que sean bioquímicos o técnicos. Ha sido complementado con interfaces hacia los analizadores, para evitar el ingreso manual de los datos.

Análisis Controles cruzados

Complementario al sistema anterior, el sistema de controles cruzados tiene como objetivo validar, a través de muestras que se corren en paralelo, que dos equipos o métodos analíticos sean equivalentes. Es decir, se utiliza para decidir si es equivalente correr una muestra en un equipo o en el otro. Presenta pantallas para el ingreso de los resultados de las muestras de control corridas, y en base a dichos datos, realiza un análisis de correlación que determina la equivalencia de los métodos. Con un rango menor de usuarios, se utiliza aproximadamente por 15 usuarios diariamente. El sistema está orientado a usuarios que sean bioquímicos.

Agenda de turnos

Sistema de turnos general, en el cual se reservan turnos de un profesional médico/bioquímico. Se utiliza en 10 puestos simultáneos. El sistema está orientado a empleados administrativos.

Gestión de reclamos documentales

Sistema de seguimiento de documentos. Se utiliza para registrar el movimiento de documentos entre los distintos departamentos de una empresa. El sistema está orientado a empleados administrativos, que en empresas medianas no llegan a ser más de 5 usuarios concurrentes. El objetivo del sistema es mantener actualizada la información sobre la ubicación física de documentos que se mueven entre departamentos.

TAT Watchdog (Turn Around Time Watchdog)

Sistema de alarma, que instalado en las terminales de la sección de emergencias de un laboratorio, presenta ventanas pop-up cuando una muestra de emergencias está pendiente mas allá del tiempo pactado para la resolución de la misma. Sin interacción con el usuario, el sistema busca las muestras de emergencias pendientes y verifica que no haya pasado el tiempo máximo de respuesta. Si el tiempo paso, se presenta el pop-up al usuario. Se instaló en 7 terminales, y realiza por búsqueda

consulta unas 300 muestras, con 20 analitos promedio cada una. El programa apunta a los técnicos y bioquímicos de laboratorio que procesan las muestras de urgencias.

Sistema	# Tablas	#Pantallas Ingreso	#Pantallas especiales ingreso	Consultas	%Incidencia
Gestión de controles	18	21	3	4	116,67
Análisis Controles cruzados	6	11	2	6	183,33
Agenda de turnos	9	6	1	2	66,67
Gestión de reclamos documentales	9	6	1	5	66,67
TAT Watchdog	3	3	0	1	100,00

Nota:

% de incidencia = (#Tablas / #Pantallas de ingreso) * 100

1.9.2. Costos de desarrollo

A continuación se detallan costos de desarrollo, obteniendo una métrica del esfuerzo empleado en distintos artefactos de un software de ERP. El sistema fue desarrollado en java utilizando JDeveloper. Es un software administrativo para empresas de medianas a grandes. Sólo se relevó el desarrollo del circuito de compras y ABM generales. Comprende el alta de un proveedor, el alta de sus productos, la autorización de los mismos, la emisión de ordenes compra a los proveedores, ingreso de mercadería, etc. Estas métricas se usarán luego para evaluar los beneficios obtenidos con la herramienta propuesta.

Función	Horas Estimadas	Horas Empleadas	Error de la estimación (hs)
ABM Bancos	24	30	6
ABM Monedas	8	8	0
ABM Proveedores	24	30	6
ABM Sucursales	8	6	-2
Autorización de proveedores	24	32	8
ABM Productos	24	24	0
Autorización de productos a sucursales	16	16	0

Emisión de Orden de compras	40	62	22
Ingreso de remitos	40	56	16

1.10. Ámbito y restricciones generales

El presente trabajo se enmarca en el ámbito de la ingeniería de software. Específicamente en el desarrollo de aplicaciones basadas en el lenguaje Delphi, con utilización de bases de datos. No pretende presentar un framework para el desarrollo íntegro de una aplicación, pero si uno que simplifique y acorte la repetitiva tarea de desarrollo de la interfase de usuario para el alta, modificación y eliminación de datos.

Los datos a presentar se deben encontrar en un motor de bases de datos relacional (RDBMS). El objetivo es poder utilizar cualquier motor de base de datos que se conecte vía drivers ADO con Delphi. Para esta primera implementación se decide utilizar MySQL versión 4.1 como motor de base de datos. La independencia de base datos, es un objetivo general de este desarrollo, aunque por ser una primera implementación, sólo se plantearán los lineamientos generales de una arquitectura de aplicación multi base de datos.

2. Estado del arte

2.1. Frameworks actuales

2.1.1. CompiereERP + CRM

CompiereERP es un producto *open source* basado en tecnología Java, que permite un alto nivel de personalización. La falta de documentación del mismo agrega un nivel de dificultad que no es menor a la hora de trabajar con el producto.

CompiereERP ofrece un framework en el cual se pueden agregar y eliminar tablas, campos y ventanas dinámicamente. En algunos casos es necesario modificar el código fuente, pero el proceso en general es sencillo. El esquema de tablas en el que almacena los metadatos, y las utilidades que posee para generarlos son una fuente interesante para el objetivo de esta tesis. Si bien es un diseño fuertemente propietario, la sencillez con la cual se han resuelto algunos temas no puede descartarse.

La implementación de CompiereERP incluye metadatos contenidos en tablas de la base de datos sobre:

1. Tablas
2. Columnas de tablas
3. Relaciones entre tablas
4. Reglas de validación de relaciones
5. Traducción – Internacionalización
6. Ventanas
7. Pestañas
8. Campos (visualización de columnas en pestañas)
9. Tipos de datos
10. Formatos de visualización de datos.
11. Procesos
12. Informes
13. Replicación entre servidores.

Lo interesante sobre esta herramienta es que el mantenimiento de los metadatos se realiza con la misma infraestructura de software que luego se utiliza en la aplicación para la visualización y edición de datos. El desarrollo en Java de la misma ha impactado fuertemente en las capacidades del producto, utilizando instanciación dinámica de clases y objetos.

CompiereERP es suficientemente flexible para que utilizando el diccionario de la aplicación (metadatos) se puedan agregar y/o eliminar elementos del menú, ventanas para visualizar tablas o vistas y personalizar el nivel de acceso a cada ventana por medio de un sistema de usuarios y perfiles.

La generación de informes también se realiza utilizando los metadatos, lo que le permite al producto entregar un cierto nivel de *drill-down* sin necesidad de programar.

El producto está orientado a ser personalizado creando procesos que deben ser programados en Java, que luego pueden ser invocados desde el menú del usuario o desde la pantalla de edición de datos. Para esto, se programa el proceso, se lo agrega al diccionario de datos de la aplicación, y luego al menú del mismo. En la definición del proceso en el diccionario se especifica la clase y el método que debe ser instanciado. Cuando el usuario dispara el proceso, la aplicación instancia dinámicamente: la clase del proceso, un objeto de dicha clase y el método de instancia que debe llamar. Cuando es instanciado el objeto y el método, se realiza la llamada del mismo.

A pesar de todas estas bondades, el código fuente es antiguo, y no responde a un enfoque "moderno" del paradigma de programación orientada a objetos. Básicamente carece de una mapeo ORM (*object relational mapping)* adecuado, generando confusiones a la hora de codificar. Por otro lado, a pesar de usar instanciación dinámica de clases y objetos, parte de la potencialidad se pierde al no usar patrones de diseño adecuados para controlar la personalización de framework.

Posee una orientación a capas, pero en el código la utilidad de la implementación es escasa. [COMP]

2.1.2. openXpertya

Basado en CompiereERP, openXpertya es un más bien un sub-proyecto del mismo. Entre las modificaciones que difieren de Compiere se pueden hallar la

traducción del sistema y la adecuación del mismo a mercados tanto hispanos como latinoamericanos. En lo que respecta al esquema de definición de pantallas y metadatos, difiere muy poco del proyecto original. Entre las pocas modificaciones que se encuentran, tiene soluciones para algunos inconvenientes del producto original, tales como visualización de tablas con altos volúmenes de datos, mejoras en el *look-and-feel,* algunas de las cuales impactan de forma directa en el diccionario de la aplicación (metadatos).

La versión para Latinoamérica del producto posee circuitos administrativos modificados para la operatoria local, sobre todo contempla la operatoria fiscal local (retenciones, percepciones, etc..).

Si bien originalmente este producto fue desarrollado utilizando el motor de base de datos Oracle, hoy en día está disponible tanto para Oracle como para postgreSQL, reforzando así el enfoque open source, que quedaba "incompleto" al utilizar un motor de BD "cerrado".

También ofrece un cliente WEB, con funcionalidades B2B y B2C. El cliente Web utiliza los metadatos para presentar en el navegador el mismo menú que se dispone en la aplicación. Las páginas son creadas a través de un servidor JBOSS que utiliza los metadatos como información base para el diseño de las páginas. La funcionalidad del mismo es algo limitada, pero suficiente para escenarios donde los requerimientos no son complejos.

2.1.3. Oracle BC4J + JDeveloper

Oracle ofrece el framework de trabajo BC4J + JDeveloper, que posee un interesante enfoque para la definición de la capa de presentación de la aplicación. Fuertemente orientado al desarrollo en 3 capas, guarda la definición del vínculo entre los objetos de la interfaz de usuario y la capa de negocios en archivos XML, utilizando un el lenguaje UIXML. Si bien la disposición de los metadatos en XML agrega portabilidad y transparencia, en la práctica, la modificación de los mismos suele ser algo complejo y poco intuitivo. El costo de agregar o modificar un campo de datos en la interfaz de usuario puede variar ampliamente, debido a los errores del entorno de desarrollo (IDE), que en muchos casos terminan introduciendo errores en los archivos XML de definición de interfaces de usuario. En BC4J + JDeveloper se hace un uso extensivo del soporte de metadatos en XML. Por ejemplo, las vistas tienen su definición en XML, los objetos también, los iteradores sobre las vistas también, etc.…

Se encuentra fuertemente orientado hacia la base de datos Oracle, aunque es posible utilizarla con otras bases de datos, por ejemplo MySQL.

Es una herramienta orientada al desarrollo rápido de aplicaciones (RAD *rapid application development*) que, en muchos casos, complica tareas que deberían ser sencillas. La fuerte orientación a capas, y la forma en la que se deben estructurar las aplicaciones hacen que la curva de aprendizaje de la herramienta sea algo "elevada".

El hecho que la misma herramienta y framework se utilice tanto para aplicaciones Web como para aplicaciones Java cliente pesado hace que algunos aspectos, sobre todo de la persistencia, se tornen complejos de comprender y analizar.

2.2. Estándares existentes

2.2.1. UIML (User Interface Markup Lenguaje)

El objetivo de UIML es proveer un lenguaje independiente, neutral y canónico que se mapea a lenguajes de programación existentes. UIML provee un método que es independiente del dispositivo para describir la interfaz de usuario

Entre las motivaciones de UIML encontramos:

- ✓ Facilitar la implementación de interfaces de usuario para cualquier dispositivo sin la necesidad de conocer el lenguaje de programación del mismo ni las API.
- ✓ Reducir el tiempo de desarrollo de Interfaces de usuario para una familia de dispositivos.
- ✓ Proveer una forma natural de separar la lógica de la aplicación de la interfaz de usuario.
- ✓ Permitir que usuarios "no programadores" implementen interfaces de usuarios.
- ✓ Facilitar la prototipación rápida de interfaces de usuario
- ✓ Simplificar la internacionalización y la "localización".
- ✓ Permitir una descarga eficiente de la interfaz de usuario en la terminal cliente.
- ✓ Brindar un mecanismo de extensión para soportar tecnologías futuras.

UIML descompone la descripción de la interfaz de usuario en seis piezas ortogonales que responden seis preguntas:

1. Cuáles son las partes que componen la UI
2. Cuál es la presentación (*look and feel*) para cada parte
3. Cuál es el contenido (x ejemplo texto, imágenes..) usado en la UI
4. Cuál es el comportamiento de la UI
5. Cómo se mapean las partes de la UI a un lenguaje en particular (por ejemplo, Java Swing, HTML tags…)
6. Cuál es la interfaz a través de la cual la UI se conecta con la lógica de negocios.

El objetivo UIML es proveer una representación canónica de cualquier UI, que sea capaz de mapearse a cualquier lenguaje. [UIML]

Se encuentran cuatro conceptos claves de UIML:

- ✓ UIML es un metalenguaje: Para utilizar UIML uno debe definir un *"toolkit vocabulary"* El vocabulario especifica las clases *de partes* y sus propiedades. Diferentes personas pueden definir distintos vocabularios que sean mas adecuados a sus necesidades.
- ✓ UIML factoriza o separa los elementos de la UI: El diseño en UIML empieza con una hoja de papel en blanco y la pregunta: ¿Cuáles son los elementos fundamentales necesarios para describir la interacción hombre-máquina? Tras esta identificación primaria de partes se procede a definir para cada parte, los estilos de presentación, los contenidos y las relaciones con orígenes de datos externos. También se define la lógica de negocios y el vocabulario empleado.
- ✓ UIML interpreta a la estructura de una UI como un árbol partes de interfaces de usuario que cambia a través del tiempo. Hay un árbol inicial de partes, que es la interfaz de usuario que se presenta a un usuario cuando se inicia el ciclo de vida de la UI. Al avanzar en el ciclo de vida de la UI, el árbol de partes puede cambiar de forma dinámicamente, agregando o eliminando partes. Por ejemplo, abrir una ventana con botones y demás controles en una interfaz gráfica puede corresponder con agregar un subárbol

de partes al árbol UIML. UIML provee elementos para describir la estructura de árbol inicial y también los necesarios paras modificarla dinámicamente.

- ✓ UIML facilita la creación de *templates* para empaquetar partes y partes-árbol. Dichos *templates* luego pueden ser reutilizados en otras interfaces de usuario. Esta es una aproximación a la noción de clases.

Debido a estos conceptos, UIML es útil para crear interfaces de usuario que sean multiplataforma, multi-lenguaje y dinámicas.

2.2.2. XUL (Extensible User Interface Lenguaje)

XUL es una gramática basada en XML para definir interfaces de usuario. Original del proyecto Mozilla, estrechamente ligado al navegador, por lo que su fuerte radica en aplicaciones web. Está pensado para el desarrollo de aplicaciones multiplataforma, donde la portabilidad de la aplicación esté garantizada. XUL brinda las ventajas de los otros lenguajes XML, que pueden ser usados indistintamente desde la aplicación. Por otro lado, el texto mostrado es fácilmente "localizable", facilitando la tarea de traducción de una aplicación. [XUL]

XUL tiene la capacidad de crear la mayoría de los elementos que encontramos en las interfaces de usuario modernas, por ejemplo:

- ✓ Controles de ingreso de datos .. (textbox, checkbox, etc…)
- ✓ Barras de herramientas
- ✓ Menús
- ✓ Diálogos
- ✓ Árboles para representaciones jerárquicas
- ✓ Atajos de teclado

Las aplicaciones XUL pueden entregarse en distintos formatos:

- ✓ Extensión de Firefox: Personalizar el browser a través de "overlays"
- ✓ Aplicación "StantAlone". La aplicación se ejecuta con el XUL Runner, que es una versión empaquetada del browser mozilla. De esta manera, la aplicación se ejecuta de forma independiente, utilizando el runner.

- ✓ Aplicación remota: Se puede poner el código XUL en un servidor Web y abrirlo desde el browser, como se haría con cualquier página. Sin embargo, este enfoque tiene limitaciones, debido a problemas de seguridad.

2.2.3. XIML (Extensible User Interface Lenguaje)

XIML es un lenguaje basado en XML que define un framework para la definición e interrelación de la interacción de ítems de datos. Como tal, XIML intenta ser un leguaje universal de especificación de interfaces de usuario. Provee una manera completa de describir una interfaz de usuario, con sus elementos, relaciones y atributos sin tener que especificar cómo serán implementados. En otras palabras, es un framework capaz de proveer un mecanismo estándar de intercambio de datos de interacción y operación para aplicaciones y herramientas. [XIML]

El lenguaje XIML incluye los siguientes unidades de representación:

- ✓ **Componentes**: En el sentido básico, XIML es una colección organizada de *elementos* de interfase que son categorizados en uno o más *componentes*. Estos componentes son aquellos hallados frecuentemente en un modelo de interfaz: tareas de usuario, objetos de dominio, tipo del usuario, elementos de presentación y elementos de diálogo.

- ✓ **Relaciones:** En XIML una relación es una definición o una regla que une dos o más elementos XIML, que no necesariamente deben estar en el mismo componente. Al capturar de manera explicita las relaciones entre elementos, XIML crea un cuerpo de conocimientos que luego puede ser utilizado en diseños basados en conocimiento, operación y evaluación de funciones para interfaces de usuario.

- ✓ **Atributos:** los atributos son las propiedades de los elementos que pueden ser asignadas a un *valor*. El valor de un atributo puede ser de un tipo "base" o puede ser una instancia de otro elemento.

Uno de los usos más importantes que se le puede dar a XIML es en el desarrollo de aplicaciones multiplataforma, que deben ser utilizados en una amplia variedad de dispositivos. XIML puede ser usado para mantener una única interfaz de usuario que se usará en todos los dispositivos objetivos de la aplicación.

Otro de los objetivos fundamentales de XIML es proveer recursos para el manejo de la interfaz de usuario en runtime. Al centralizar en una única definición la interacción de datos de una interfaz, se debería poder construir herramientas que utilicen dicha definición para mejorar las funciones relacionadas al uso de la interfaz de usuario.

Entre estas funciones encontramos:

- ✓ Reorganización dinámica de contenidos
- ✓ Personalización
- ✓ Gestión distribuida de la interfaz: La definición de la misma puede estar disponible en un recurso de red, y de esta manera la actualización se vuelve tan sencilla como actualizar dicho recurso.

3. Descripción del trabajo realizado

3.1. Objetivo

Diseñar y desarrollar artefactos de software que permitan:

- ✓ Generar metadatos desde una tabla existente en la base de datos.
- ✓ Modificar dichos metadatos, de manera tal que se puedan explicitar relaciones entre ellos y estilos de visualización de los mismos.
- ✓ Incorporar a una aplicación un Delphi un conjunto de clases que utilizando los metadatos obtenidos generen de forma dinámica la interfaz de usuario.

3.2. Consideraciones generales

Sobre metadatos

Se presentan distintas alternativas a la hora de almacenar y presentar los metadatos. Entre ellas se encentran las siguientes:

a. Almacenarlos como archivos XML, modificándolos con alguna herramienta existente hoy en el mercado.
b. Almacenarlos como archivos XML y utilizar las herramientas provistas por Delphi para su visualización y modificación.
c. Diseñar tablas en la base de datos para contener los metadatos. El mantenimiento de dichas tablas se haría con el mismo framework que se usaría en la aplicación. Es decir, se podría llegar a pensar en metadatos de metadatos.

De los enfoques presentados, se opta por el esquema c., diseñar tablas en la base de datos para contener los metadatos. La ventaja de este esquema radica en que se puede utilizar el mismo framework que se utilizará en la aplicación, para el mantenimiento de los metadatos. Es decir, luego de una configuración manual de los metadatos del framework, se puede utilizar el mismo framework para continuar con la configuración de la aplicación. Se descarta la utilización de archivos en formatos XML, utilizando

algún estándar como XUL o XIML debido a que con el enfoque descrito, se obtiene una visión más integrada del producto. Por otro lado también se evita la tarea de desarrollar herramientas para la manipulación de dichos archivos, puesto que, con el esquema elegido, el mantenimiento de los metadatos se hará con la misma herramienta que de todas maneras se debe desarrollar para presentar los datos al usuario.

Sobre el diseño de la Interfaz de usuario

La interfaz de usuario es fundamental en todo sistema, puesto que el usuario lo percibe e interactúa a través de ella.

En este sentido, la interfaz a desarrollar contendrá los siguientes fundamentos:

- ✓ Vista de grilla de datos o *browser*: Los datos se presentan en una tabla, pudiéndose visualizar y/o editar muchos registros a la vez. Se debe contemplar un método de filtrado para los registros. Sería interesante que el filtro sea obligatorio automáticamente en casos de tablas con alta cantidad de registros.
- ✓ Vista de detalles: La vista de detalles presenta como "formulario" los datos, visualizándose un solo registro. Esta sería la vista principal para realizar el ingreso y modificación de datos.
- ✓ Indicador de campos obligatorios: La interfaz de usuario presentará de forma distinta aquellos campos cuyo ingreso sea obligatorio. En particular, cambiará el color de fondo de los mismos.
- ✓ Campos Lógicos: La interfaz de usuario presentará los campos lógicos (verdadero o falso) como *check box*, que al estar seleccionados indican el valor "verdadero".

Sobre el lockeo de registros y manejo de transacciones

- ✓ Se propone utilizar un esquema de *lockeo* "optimista", con "COMMIT" realizado al momento de guardar el registro. Es decir, no se efectúa un "COMMIT" automático, sino que se realiza cuando se confirman los cambios del registro.
- ✓ Se desestima la opción de realizar "COMMITS" manuales desde la UI en operaciones que involucren a más de un registro (presentado en la UI), puesto que esto agrega trabajo al usuario y además tiende a confundirlo sobre los pasos que ha realizado.

Valores lógicos en la base de datos

Para representar valores lógicos (verdadero o falso) en la base de datos se adopta la siguiente convención:

- ✓ Se crearán en las tablas como valores enteros
- ✓ El valor 0 (cero) representa el valor lógico falso
- ✓ El valor 1 (uno) representa el valor lógico verdadero.

Vale aclarar que el sistema interpretará a cualquier valor distinto de cero como verdadero. Pero se recomienda respetar la convención.

Definición de claves primarias

Las tablas a utilizar en el framework deben tener definida una clave primaria del siguiente modo:

"nombre de tabla" + "_ID"

Algunos ejemplos:

Tabla	Clave primaria
Ciudad	Ciudad_ID
Campos	Campos_ID
Usuario	Usuario_ID

Actores del sistema

Es posible diferenciar al menos tres actores que interactúan con el sistema:

Usuarios

Programadores

Implementadores

Las actividades que realiza cada uno se pueden ver en el cuadro:

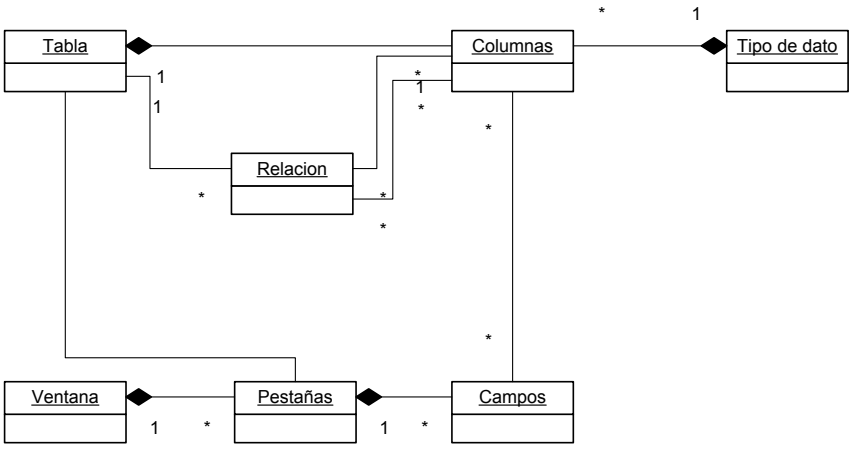
Actor	Procesos
Usuario generales	Uso general de la aplicación. No accede al mantenimiento del sistema.
Programadores	Personalizan la aplicación, agregan comportamiento específico, desarrollo general. *Modificación de código.*
Implementadores	Personalizan la aplicación, ubicación de campos, visibilidad, etc…

La diferencia entre programadores e implementadores se da debido a que el trabajo desarrollado permite una clara diferenciación entre las necesidades de conocimiento específico que debe tener cada uno.

El programador, debe conocer de forma cabal como funciona el producto, agrega código al programa y lo personaliza para que cumpla con los objetivos propuestos.

El implementador, por otro lado, puede resolver por si solo algunos aspectos sencillos, tales como asignar el nombre de campo adecuado, asignar el orden de los campos en la pantalla y afines. Las tareas del implementador no requieren programación, pero si un conocimiento del dominio de la aplicación desarrollada. Se podría decir que “pule” la aplicación.

3.3. Arquitectura de metadatos



Modelo conceptual de metadatos

Metadatos

Se definen los siguientes conjuntos de metadatos

Tabla: Catálogo de tablas a usar. Una tabla contiene una colección de columnas.

Nombre	Tipo dato	Propósito/Comentarios
Md_tabla_ID	Entero	Identificador único de la tabla.
Nombre	String	Nombre de la tabla
Comentario	String	Comentario descriptivo de la tabla
FiltroInicial	Boolean	

Columnas: Columnas de las tablas a utilizar. En la columna se puede:

1. Especificar una relación con otra tabla.
2. Especificar el formato de ingreso del dato

3. Especificar si es obligatorio o no.

Nombre	Tipo dato	Propósito/Comentarios
MD_Columna_ID	Entero	Identificador único de tabla
MD_Tabla_ID	Entero	Tabla a la que pertenece la columna
Nombre	String	Nombre de la columna
Descripción	String	Comentario descriptivo de la tabla
Largo	Entero	Longitud del campo
Formato	String	Formato de ingreso y presentación de los datos del campo
MD_Relacion_ID	Entero	Especifica la relación con otra tabla de datos. Se usa para definir claves foráneas en la tabla.
Obligatorio	Boolean	El campo es de carácter obligatorio
Filtro	Boolean	El campo se debe usar para filtrar registros en la tabla.
MD_Tipo_Dato_ID	Entero	Especifica el tipo de dato de la columna.

Relaciones: Especifica la forma de visualizar una relación entre tablas.

Nombre	Tipo dato	Propósito/Comentarios
MD_Relacion_ID	Entero	Identificador único de la relación.
MD_Tabla_ID	Entero	Identificador de la tabla destino de la relación.
MD_Columna_ID	Entero	Identificador de la columna en la tabla de destino que será usada para la relación.
Descripción	String	Descripción del objetivo de la relación.
Mostrar_Columna_ID	Entero	Columna de la tabla destino que se debe mostrar en la tabla origen.

Ventanas: Ventanas a utilizar, la ventana será el punto de entrada de la aplicación Delphi al framework. Una ventana podrá contener una o varias pestañas.

Nombre	Tipo dato	Propósito/Comentarios
MD_Ventana_ID	Entero	Identificador único de registro
Titulo	String	Titulo a desplegar en la ventana.
Descripción	String	Descripción de la ventana, objetivo de la misma.

Pestañas: Una pestaña define la visualización de una tabla. En ella se especifica que tabla se mostrará y si hay una relación maestro-detalle con la pestaña anterior, cual es el campo que vincula la tabla maestro con la tabla detalle. La pestaña, contiene un conjunto de campos a mostrar.

Nombre	Tipo dato	Propósito/Comentarios
MD_Pestana_ID	Entero	Identificador único de la pestaña
MD_Ventana_ID	Entero	Ventana a la que pertenece la pestaña
Secuencia	Entero	Orden de la pestaña dentro del conjunto de pestañas de la ventana
MD_Columna_Primaria_ID	Entero	Columna primaria de la pestaña. Se utiliza para especificar relaciones maestro/detalle entre pestañas.
Nro_columnas_verticales	Entero	Cantidad de columnas verticales a mostrar en el modo de edición/formulario
Descripción	String	Descripción de la pestaña
Activo	Boolean	Indicador de validez de la pestaña. La ventana solo mostrará pestañas que estén activas.
MD_Tabla_ID	Entero	Tabla de origen para los datos a mostrar en la pestaña.

MD_TabType	Entero	Identifica la clase que mostrará la pestaña
Nombre	String	Nombre a mostrar en la pestaña.

Campos: Un campo modela la forma de visualizar una columna en una pestaña. Define la posición del mismo, y su lógica de visibilidad.

Nombre	Tipo dato	Propósito/Comentarios
MD_Campo_ID	Entero	Identificador único del campo
MD_Pestana_ID	Entero	Pestaña a la que pertenece el campo
Nombre	String	Nombre a desplegar junto al ingreso de datos.
Visible_Grilla	Boolean	El campo se verá en la grilla de datos.
Secuencia	Entero	Orden del campo dentro de la pestaña
Visible_Editar	Boolean	El campo estará disponible en la vista de edición/formulario.
Pos_columna	Entero	Columna en la que se debe desplegar el campo
Solo_lectura	Boolean	El campo no será modificable si es de solo lectura.
Descripción	String	Comentario sobre el campo, su funcionalidad y sentido.
Obligatorio	Boolean	El ingreso de datos es obligatorio en el campo para poder grabar un registro.
Md_Columna_ID	Entero	Columna de la tabla a visualizar en el campo.

Tipo de dato: Tipo de dato utilizado por una columna. Puede ser por ejemplo texto, entero, moneda, Boolean, Fecha.

Nombre	Tipo dato	Propósito/Comentarios
MD_Tipo_Dato_ID	Entero	Identificador único de registro
Descripción	String	Descripción del tipo de dato

Los tipos de datos utilizables en el sistema son los siguientes:

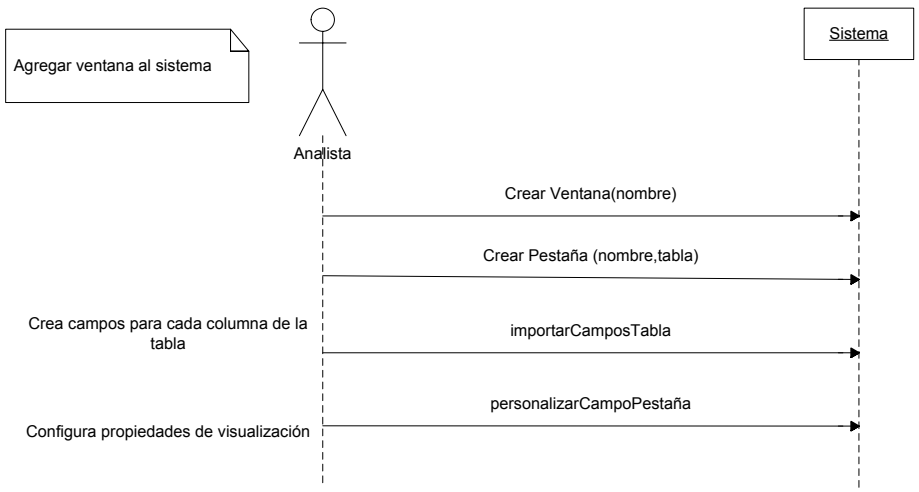
MD_Tipo_Dato_ID	Descripción	Comentarios
1	ID	Usado para identificar clave en los metadatos
2	Entero	
3	Date	
4	Date/Time	
5	Cadena	
6	Boolean	
7	Moneda	Números reales con dos decimales.

3.4. Metodología de uso

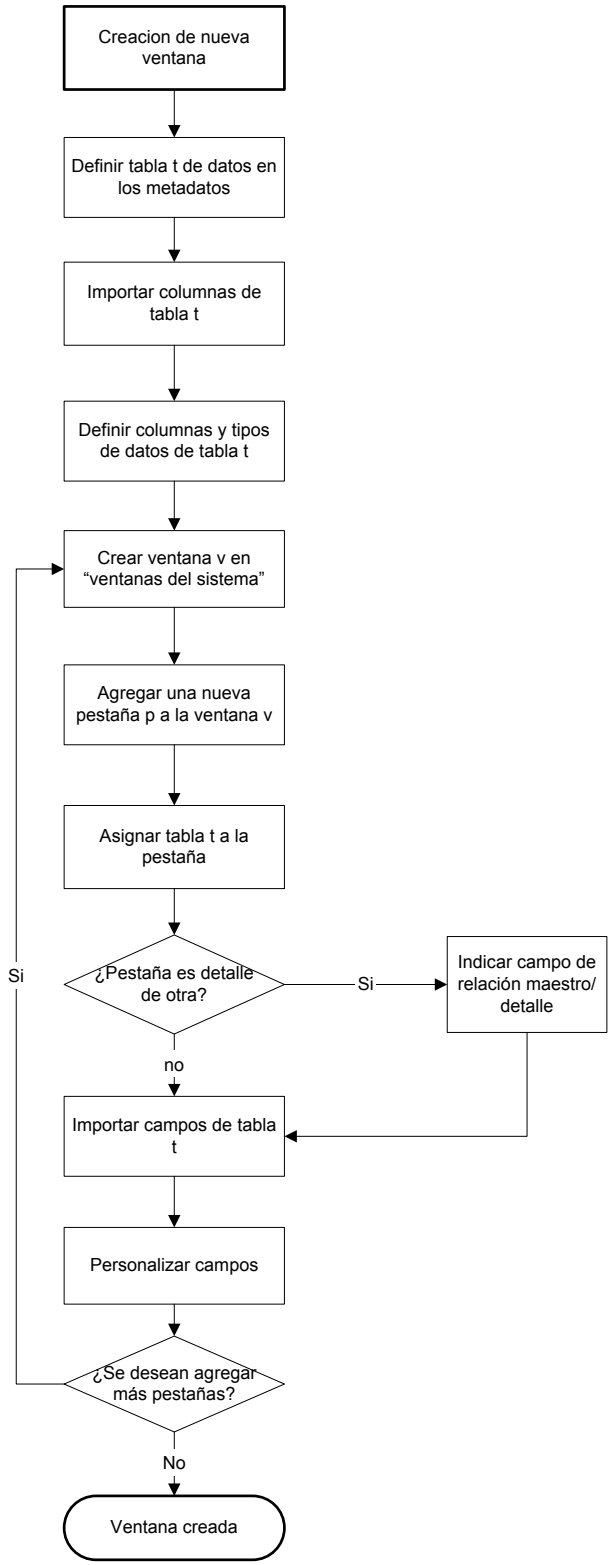
3.4.1. Pasos generales

La metodología propuesta consiste en realizar los siguientes pasos:

1. Agregar la tabla a los metadatos, utilizando la utilidad de importación de tabla.
2. Revisar la definición de la tabla y agregar las referencias en las columnas que corresponda.
3. Crear una ventana con una pestaña que muestre los datos de la tabla.
4. En caso de ser necesario un comportamiento específico, generar las clases correspondientes.
5. Desde el menú de la aplicación, instanciar la visualización de datos invocando el nombre de la ventana creada.

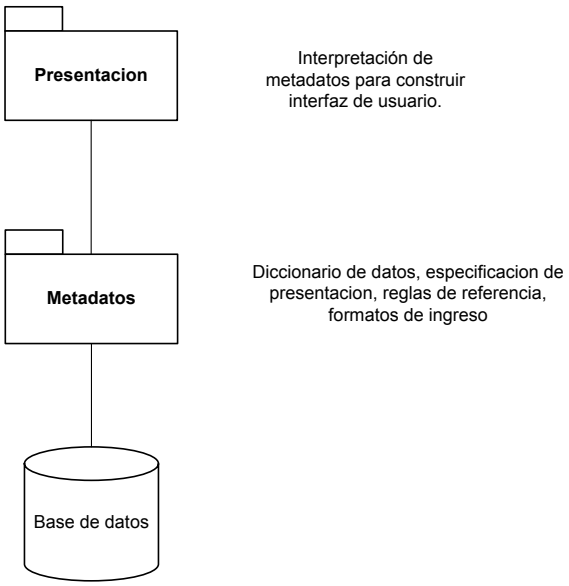


El siguiente diagrama de flujos es una guía sobre los pasos a realizar:



3.4.2. Detalles

Arquitectura de la aplicación



Personalización del framework

El framework presentado está pensado para ser extendido de forma tal que se pueda personalizar para alcanzar las necesidades de la aplicación objetivo. Con este fin se utilizó el *factory pattern:*

El factory pattern se utiliza para crear objetos sin especificar la clase exacta del objeto a crear. El objeto usuario del factory solicita una instancia de un objeto, y el factory le retorna el objeto necesario según las decisiones que él toma.

En el caso puntual del framework presentado, los pasos y las opciones disponibles para la personalización son las siguientes:

a) Personalizar la clase que muestra la grilla de datos: Si se desea modificar la forma en que se visualiza una grilla, o agregar procesos al comportamiento de la aplicación:

   Crear una subclase de TTabView, que implemente los métodos necesarios, para facilitar esta tarea, TTabView provee los siguientes métodos a ser sobrescritos:

insertarBotonExtra: Agrega a la pestaña un botón al lado de la barra de navegación.

afterSetup: Este método se invoca luego de realizar toda la configuración de la pestaña. Aquí se pueden agregar, por ejemplo, botones adicionales utilizando **insertarBotonExtra.** También se podría agregar aquí cualquier modificación que se necesite.

beforeDelete: Se invoca antes de eliminar un registro del dataset.

afterDelete: Se invoca después de eliminar un registro del dataset.

beforeInsert: Se invoca antes de insertar un registro del dataset.

afterInsert: Se invoca después de agregar un registro del dataset.

beforePost: Se invoca antes de grabar los cambios que se han realizado en un registro del dataset.

afterPost: Se invoca después de grabar los cambios que se han producido en un registro del dataset.

b) Personalizar el comportamiento del manejo de datos: Si bien los métodos beforeDelete, afterDelete, beforeInsert, afterInsert, beforePost, afterPost pueden ser re-escritos en las subclases de TTabView, es posible también delegar estas funciones a una clase que implemente dichos métodos definidos en la interface IPersistentObject, y asignar dicha clase al objeto persistentObject que es miembro de TTabView. En este caso, TTabView delegará todas las funciones al objeto persistente. Para utilizar ésta característica, los pasos serian los siguientes:

Crear una subclase (S) de TTabView.

Sobrescribir en S el método afterSetup de la siguiente manera:

```
Procedure S.afterSetup()
Begin
        persistentObject:=MiObjetoPersistente.Create();
End;
```

3.5. Diseño de la interfaz de usuario

El trabajo principal se desarrolla sobre la pantalla de ingreso/visualización de datos. La misma presenta los siguientes controles:

Ejemplo: Ventana en modo "grilla"

✓ En la parte superior de la ventana se presenta la barra de navegación, que permite:

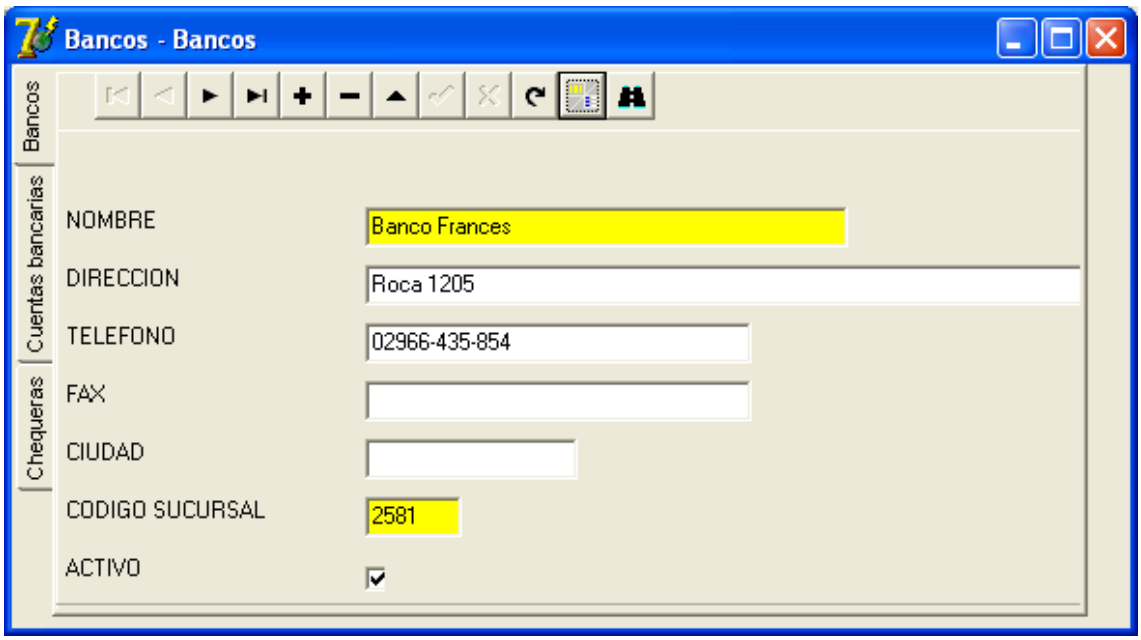
- Moverse entre los registros, ir al último, al primero, al siguiente y al anterior
- Agregar un nuevo registro
- Eliminar el registro actual.
- Confirmar los cambios realizados
- Cancelar los cambios realizados
- Cambiar el modo de visualización a modo "formulario/edición"
- Filtrar / buscar registros

✓ En la parte central de la misma se presenta la grilla de datos, formada por filas y columnas, en la cual se puede realizar la edición de datos si es que así se desea.

- ✓ En el borde lateral izquierdo se presentan las pestañas que componen la ventana. Cuando existe una relación entre los registros de las ventanas sucesivas en una relación "maestro detalle", las pestañas de orden inferior siempre mostrarán sólo los registros que estén relacionados con los del nivel superior. En el ejemplo, en las pestaña de ciudades sólo se mostrarán aquellas que pertenezcan a la provincia seleccionada en la pestaña de provincias.

Los campos que requieran ingreso de datos obligatorios se marcarán con fondo amarillo. Dicha convención se utilizará tanto en el modo grilla como en el modo detalle.

Ejemplo: La misma ventana en modo "formulario/edición"



Una de las ventajas obtenidas al utilizar este esquema es la consistencia. Recordando lo expuesto en el punto 1.3 sobre Interfaces de usuario, se pueden destacar los siguientes aspectos que compartirán todas las ventanas desarrolladas empleando el framework:

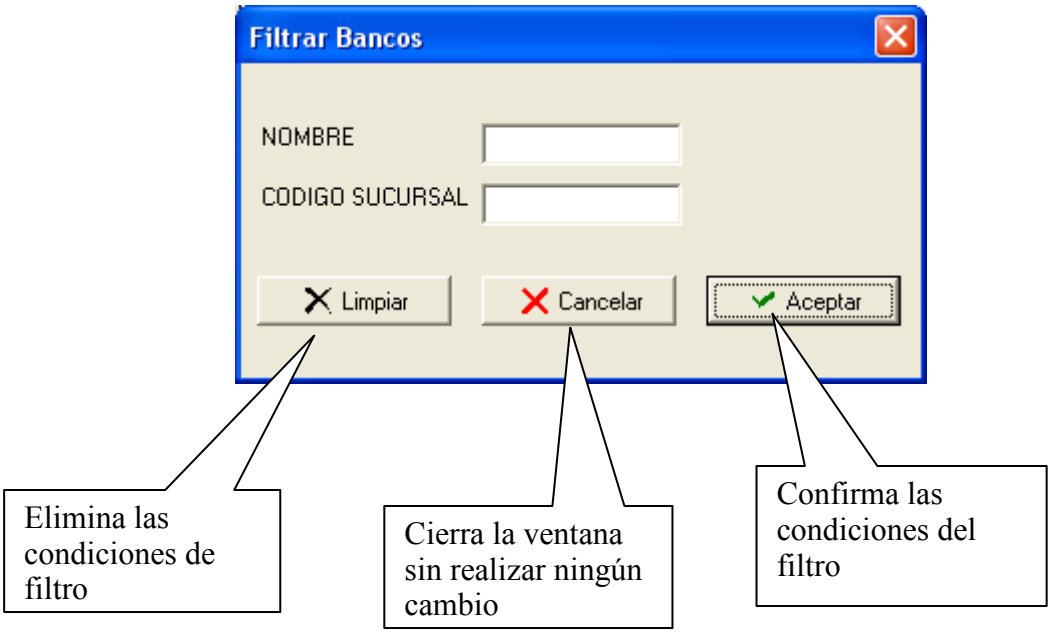
- ✓ La barra de navegación de registros estará siempre en el mismo lugar.
- ✓ Las opciones de dicha barra serán siempre las mismas, estando habilitadas o no según las circunstancias.
- ✓ Las operaciones adicionales disponibles que dependan de una ventana en particular siempre estarán en el mismo lugar (al lado de la barra de navegación)

- ✓ La información sobre que es lo que se está viendo en la grilla de datos siempre estará disponible en el nombre de la pestaña.
- ✓ En el caso de haber una relación maestro-detalle, el maestro siempre estará en la pestaña anterior al detalle.
- ✓ El modo grilla/detalle estará disponible en todas las ventanas desarrolladas con el framework
- ✓ En todas las ventanas, los campos obligatorios se identificarán de igual manera.
- ✓ El filtrado y la búsqueda de registros se realiza siempre de la misma forma.

Como se puede ver, todas estas características garantizan una interfaz de usuario consistente en el marco del trabajo propuesto.

El botón de filtro/búsqueda se encontrará habilitado solo cuando se halla definido al menos un campo de búsqueda para la tabla.

Un ejemplo de una pantalla de filtro:



3.6. Casos de estudio

3.6.1. Caso: Esquema de seguridad: usuarios, perfiles, roles, funciones

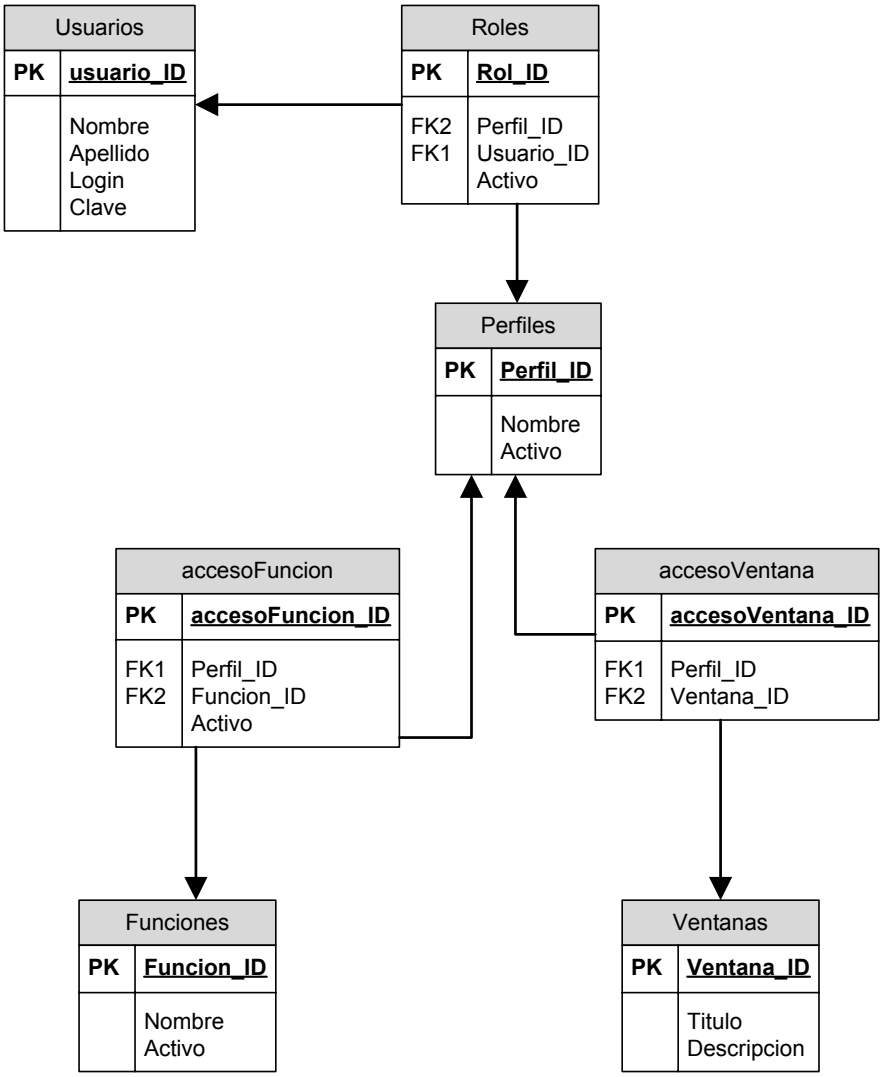
A continuación se detallará, paso a paso, como se agregó al framework un esquema de seguridad utilizando el mismo framework. Con este ejemplo se persiguen dos objetivos: mostrar paso a paso como puede utilizar el framework y a la vez proveer al mismo de un esquema de seguridad.

Diseño del esquema de seguridad

El esquema de seguridad propuesto se basa en las siguientes entidades:

Usuarios: Usuarios que interactúan con el sistema.
Perfiles: Un perfil es un agrupamiento de accesos a funciones y ventanas.
Roles: Es la asignación de perfiles a usuarios.
Funciones: Funciones dentro del sistema donde se requiere que se aplique una validación de seguridad.
Acceso a funciones: Especifica si el perfil puede o no ejecutar determinada función.
Acceso a ventanas: Especifica si el perfil puede o no ingresar a una ventana.

Esquema de seguridad: Diagrama de entidad / relacion



3.6.1.1. Definición de tabla y ABM de usuarios

Se crea la tabla de usuarios con el siguiente script:

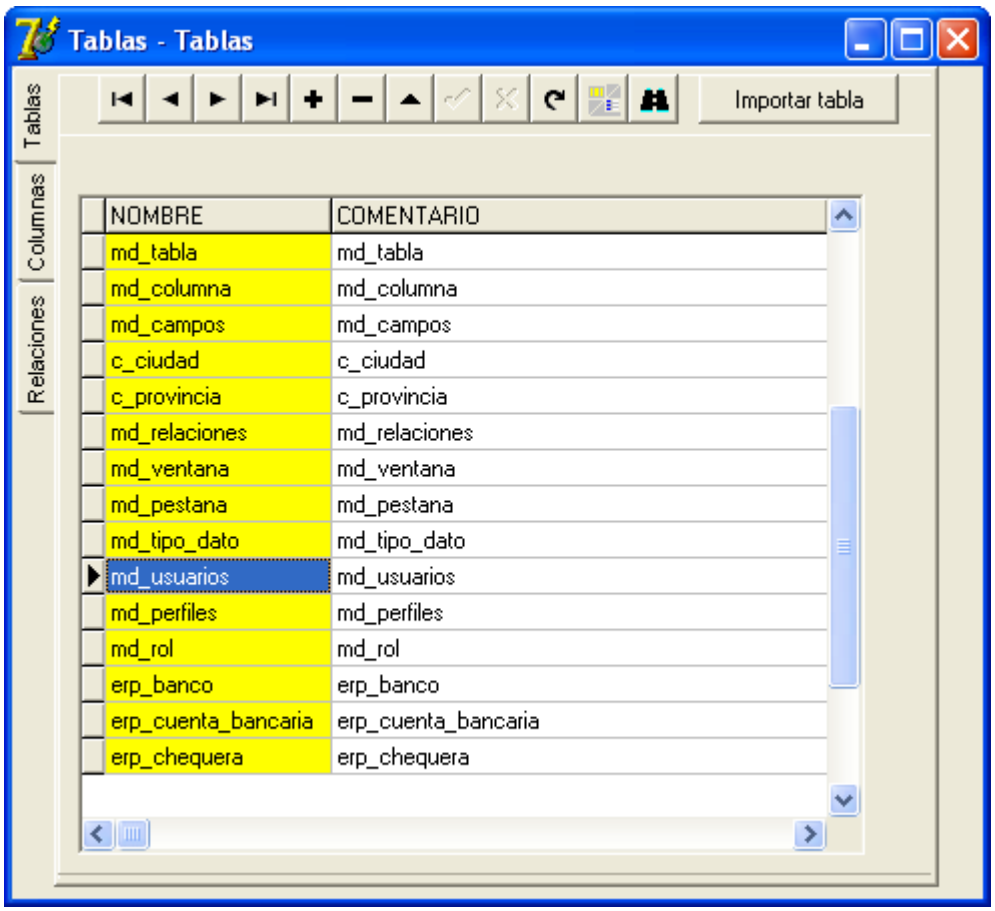
```
CREATE TABLE `md_usuarios` (
 `md_usuario_id` int(11) NOT NULL auto_increment,
 `nombre` varchar(30) NOT NULL default '',
 `apellido` varchar(30) NOT NULL default '',
 `login` varchar(15) NOT NULL default '',
 `clave` varchar(50) NOT NULL default '',
```

```
 PRIMARY KEY (`md_usuario_id`)
);
```

3.6.1.2. Registro de la tabla de usuarios en los metadatos

Para registrar la tabla en los metadatos, se debe ingresar a la ventana de "Tablas" y agregar un registro con el nombre de la tabla (debe ser exactamente igual).

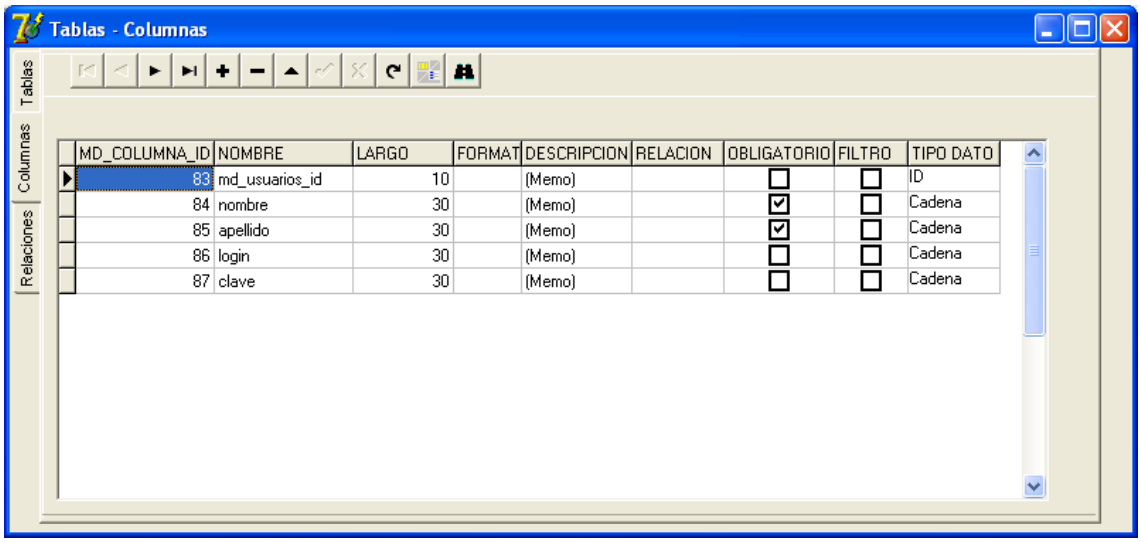
Luego de esto, presionar el botón "Importar tabla". El sistema recuperará de la base de datos la estructura de la tabla, creando las columnas de la misma.



3.6.1.3. Revisión de las columnas de la tabla

Luego de la importación de las columnas, se recomienda revisar con detalle las definiciones de las columnas, con el objetivo de asegurar que los tipos de datos y los largos de las columnas sean correctos. Para esto:

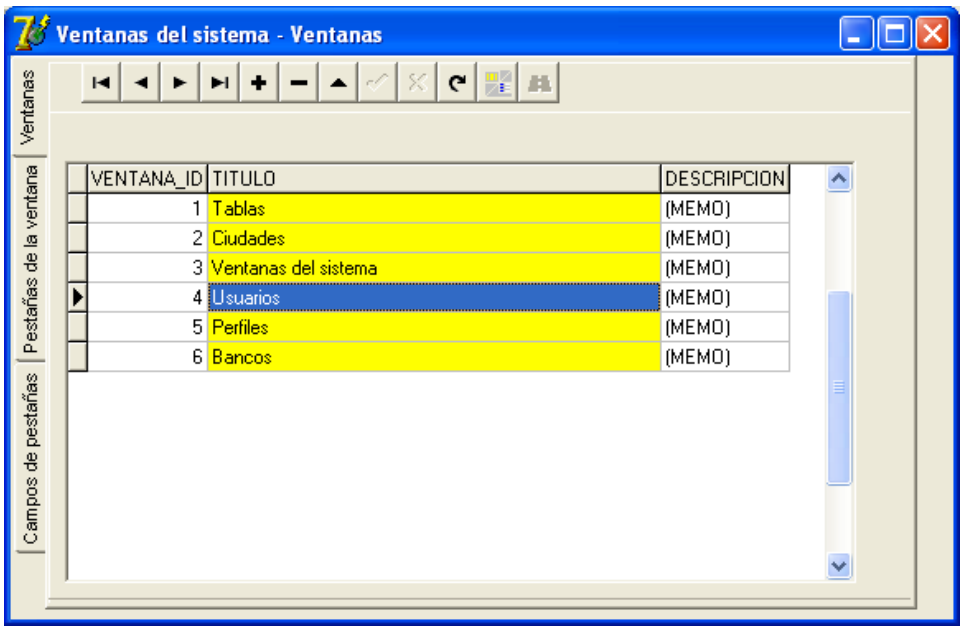
Desde la ventana de tablas del sistema, posicionarse en la tabla “md_usuarios” y cambiar a la pestaña de columnas.



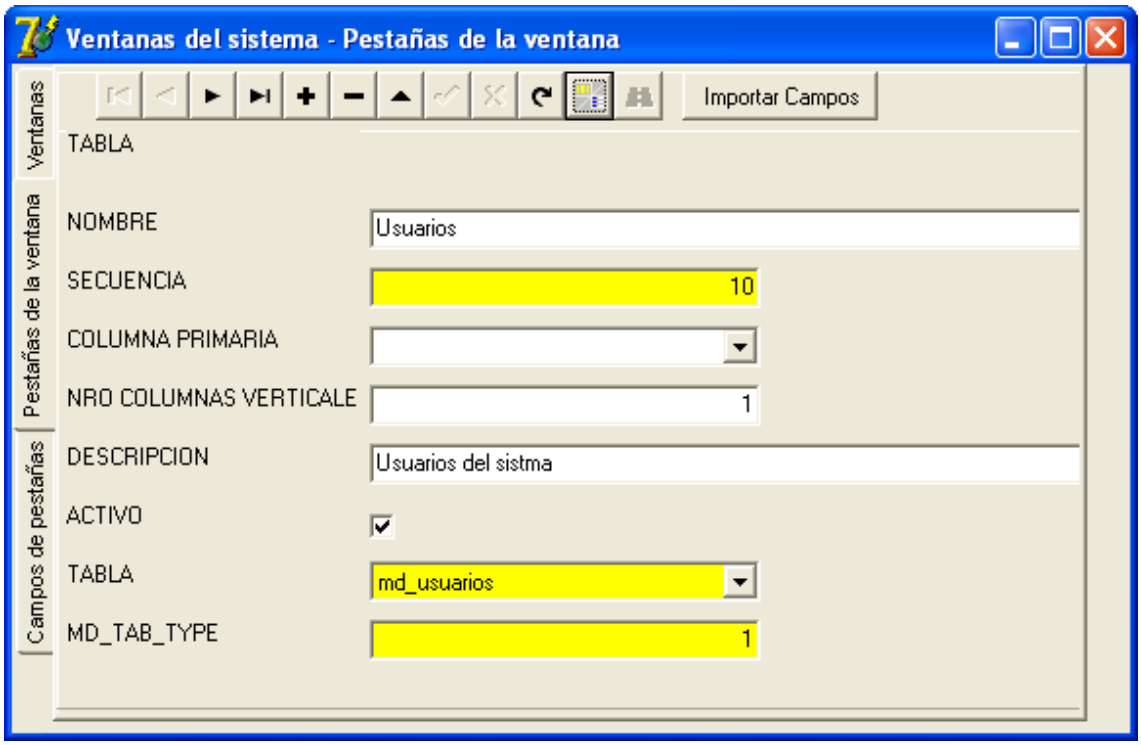
3.6.1.4. Definición de la ventana de usuarios:

Una vez verificados los metadatos de la tabla de usuarios, se definirá una ventana para su visualización.

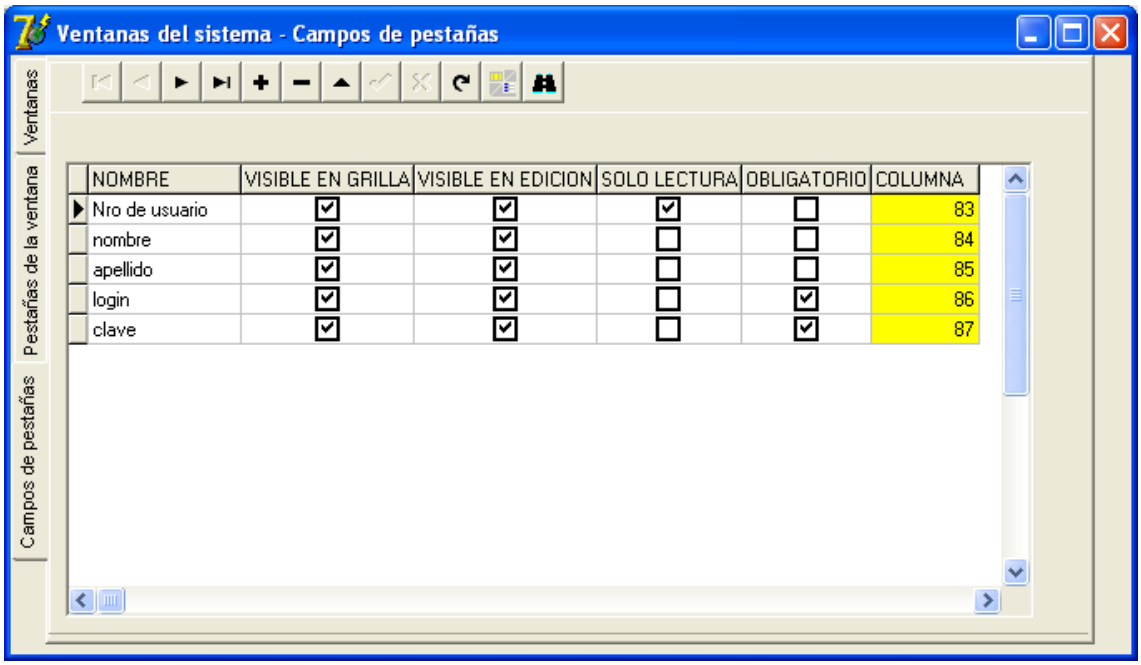
- ✓ Ingresar a la ventana “Ventanas del sistema”
- ✓ Definir una nueva ventana con el nombre “usuarios”



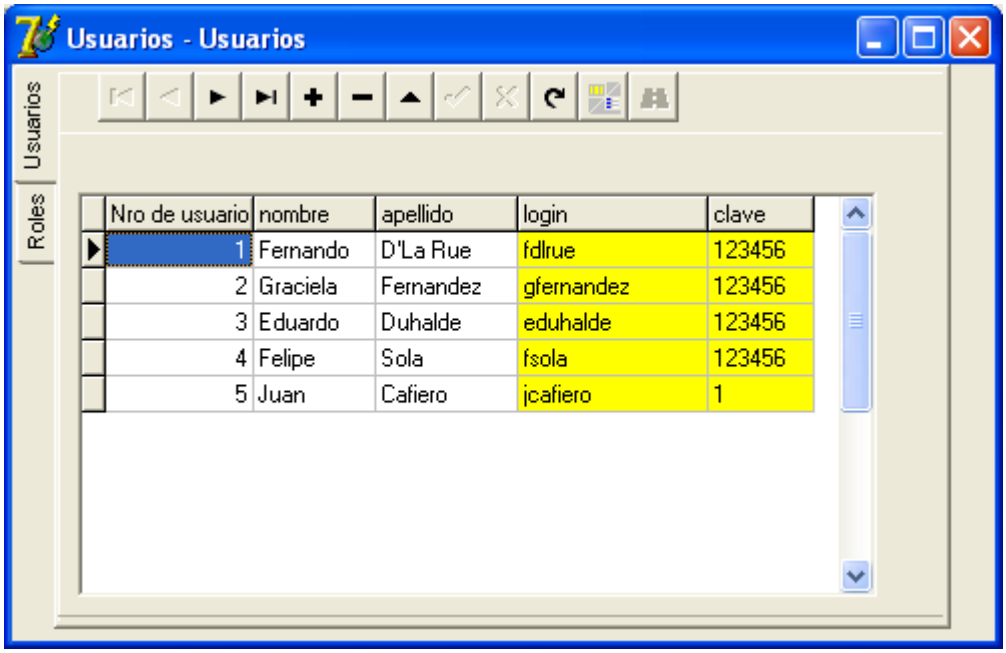
Cambiar a la pestaña de pestañas de la ventana, y definir una nueva pestaña con la tabla de usuarios.



Presionar el botón “Importar campos” para crear los campos en la pestaña con los campos de la tabla. Revisar los campos y acomodarlos a gusto.



El resultado de toda esta configuración se ve en la ventana “Usuarios”:



3.6.1.5. Definición de tabla de perfiles

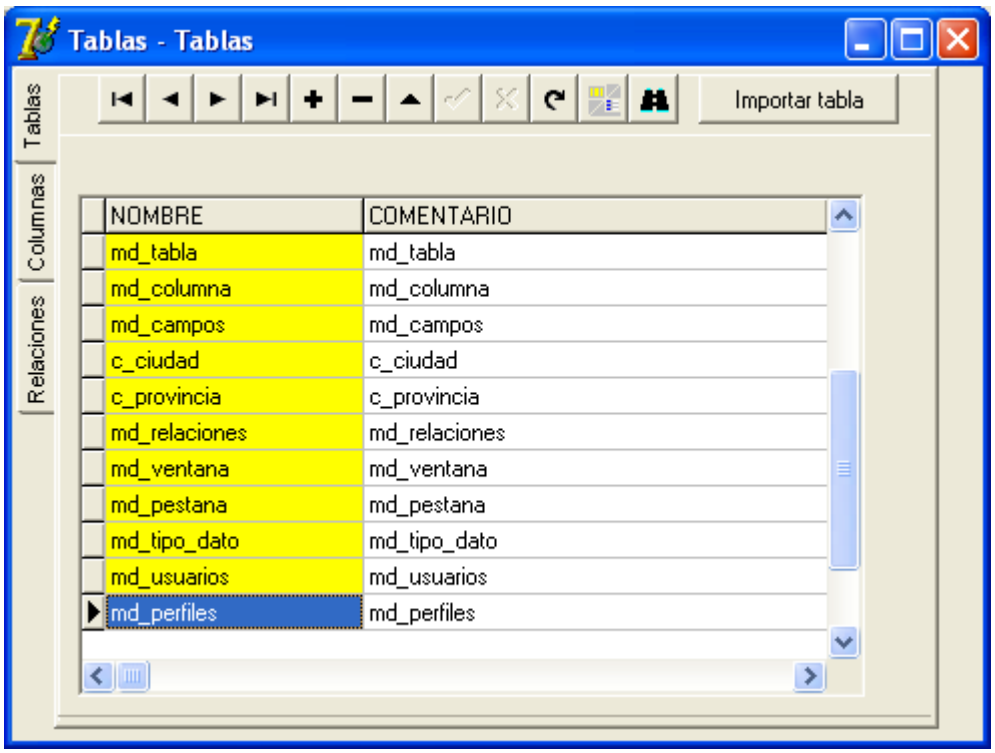
Se creará la tabla de perfiles con el siguiente script:

```
CREATE TABLE `md_perfiles` (
 `md_perfil_id` int(11) NOT NULL auto_increment,
 `nombre` varchar(50) default NULL,
 `activo` tinyint(4) NOT NULL default '0',
 PRIMARY KEY (`md_perfil_id`)
)
```

3.6.1.6. Registro de la tabla de perfiles en los metadatos

Ingresar a la ventana de "Tablas" y agregar un registro con el nombre de la tabla (debe ser exactamente igual).

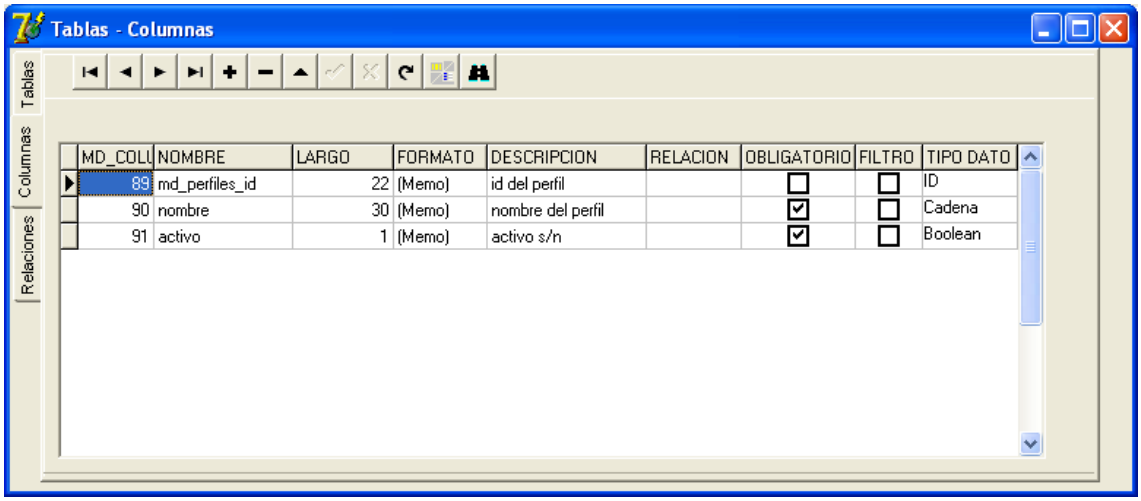
Luego de esto, presionar el botón "Importar tabla". El sistema recuperará de la base de datos la estructura de la tabla, creando las columnas de la misma.



3.6.1.7. Revisión de las columnas de la tabla

Luego de la importación de las columnas se recomienda revisar con detalle las definiciones de las columnas, con el objetivo de asegurar que los tipos de datos y los largos de las columnas sean correctos. Para esto:

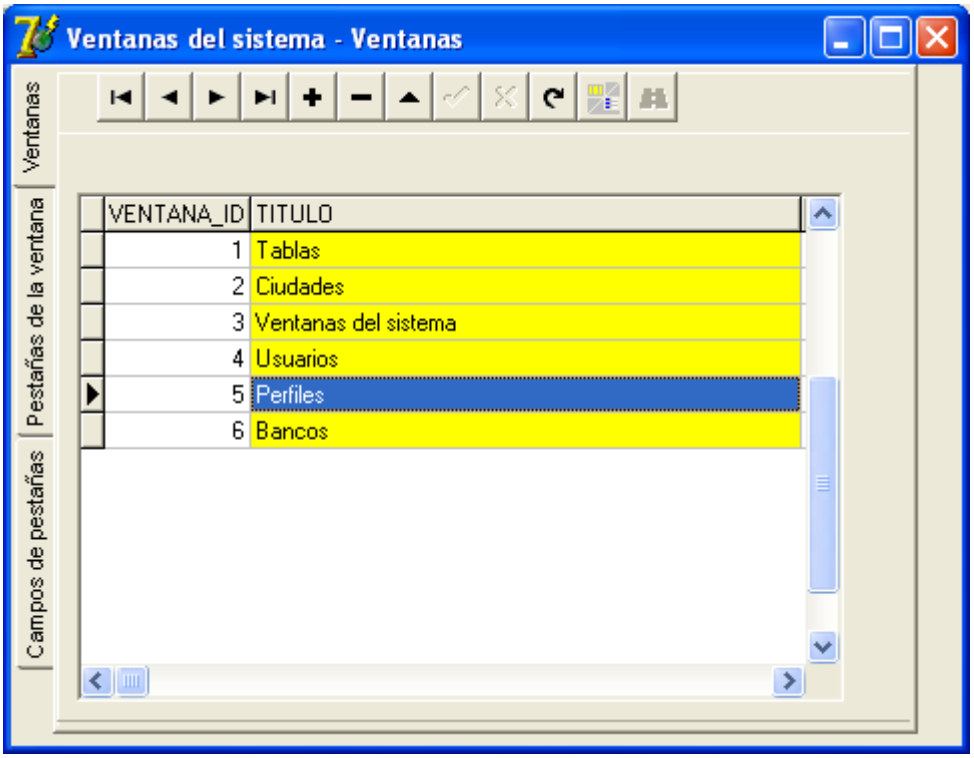
Desde la ventana de tablas del sistema, posicionarse en la tabla "md_perfiles" y cambiar a la pestaña de columnas.



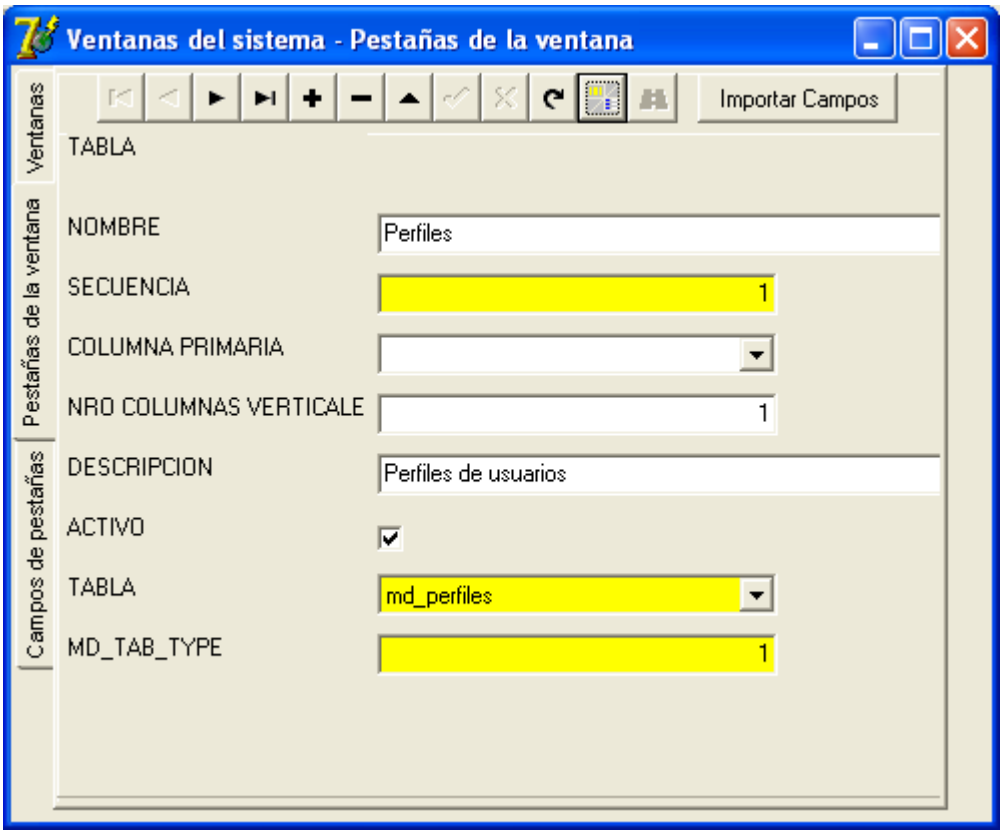
3.6.1.8. Definición de la ventana de perfiles:

Una vez verificados los metadatos de la tabla de perfiles, se definirá una ventana para su visualización.

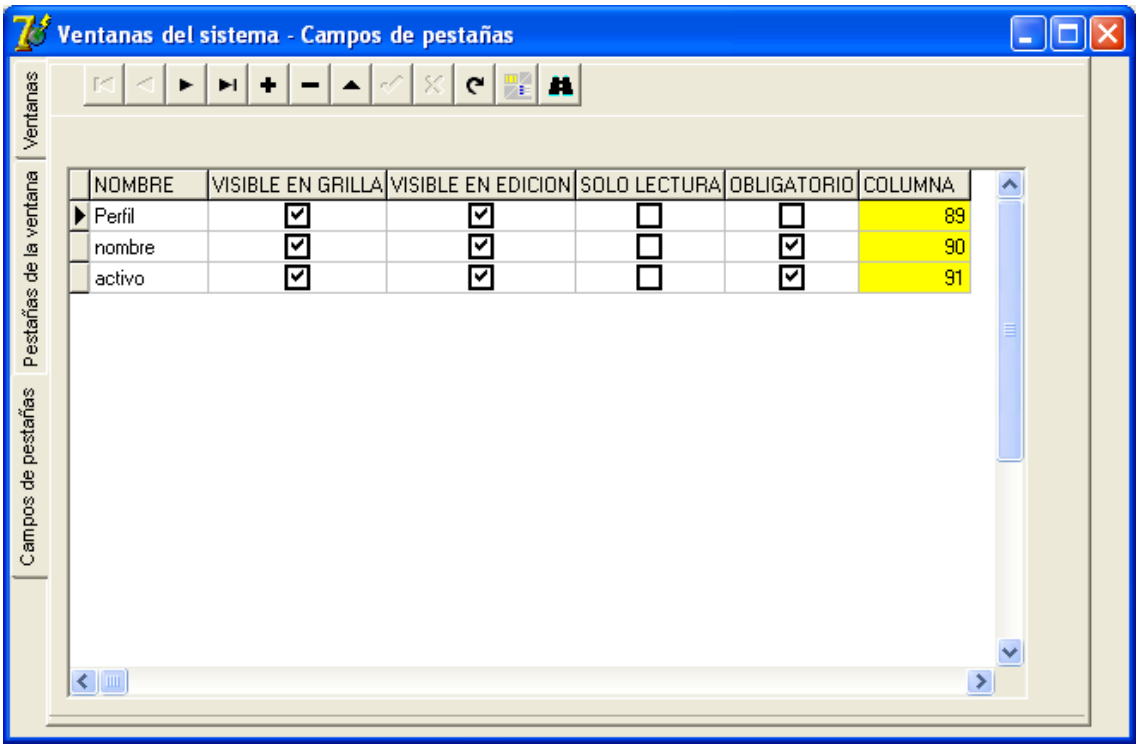
- ✓ Ingresar a la ventana "Ventanas del sistema"
- ✓ Definir una nueva ventana con el nombre "perfiles"



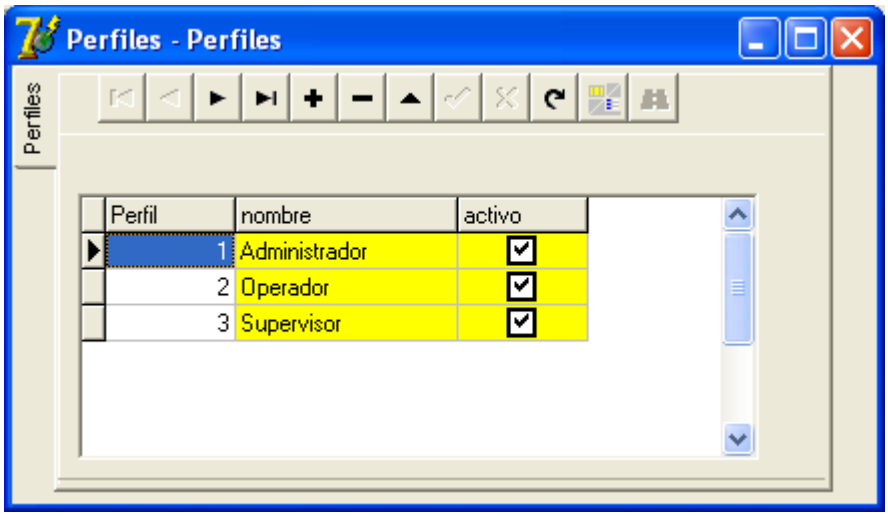
Cambiar a la pestaña de pestañas de la ventana, y definir una nueva pestaña con la tabla de perfiles.



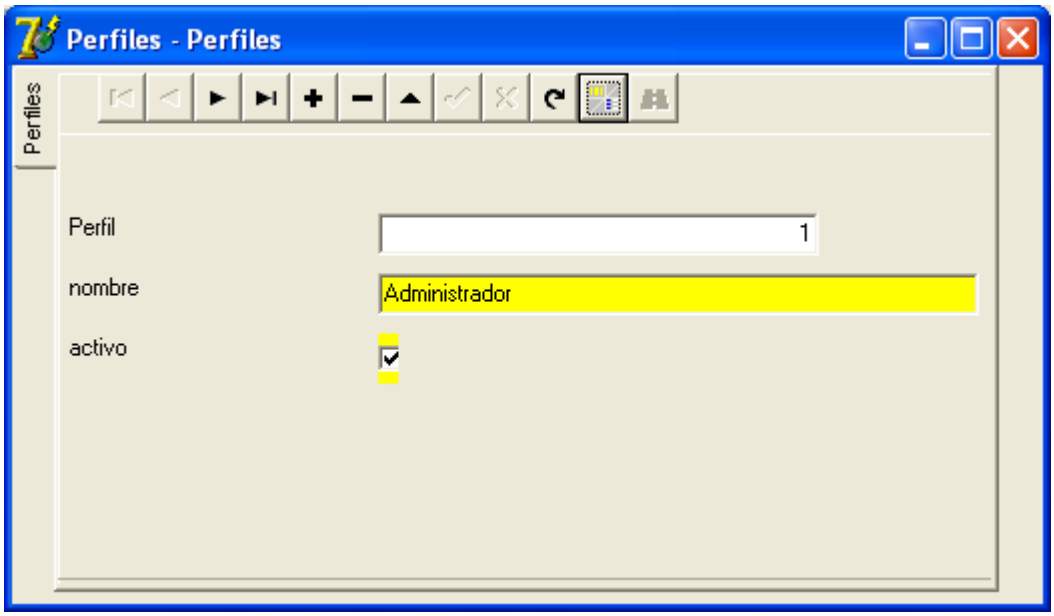
Presionar el botón “Importar campos” para crear los campos en la pestaña con los campos de la tabla. Revisar los campos y acomodarlos a gusto.



El resultado de toda esta configuración se ve en la ventana “Perfiles”:



Cuya vista de edición de registro es:



3.6.1.9. Relación entre usuarios y perfiles

Para culminar el ejemplo, se agregará la tabla de Roles, que asigna perfiles a usuarios. Esta tabla se visualizará como un detalle de la tabla de usuarios, mostrando para el usuario seleccionado los roles que desempeña.

La tabla se crea con el siguiente script:

```
CREATE TABLE `md_rol` (
 `md_rol_id` int(11) NOT NULL default '0',
 `md_perfiles_id` int(11) default NULL,
 `md_usuarios_id` int(11) default NULL,
 `activo` tinyint(4) default NULL,
 PRIMARY KEY (`md_rol_id`),
 KEY `usuario_id` (`usuario_id`)
);
```

3.6.1.10. Registro de la tabla de roles en los metadatos

Ídem Pasos 3.6.1.2 y 3.6.1.3 pero con la tabla md_roles

3.6.1.11. Agregar la pestaña de roles a la ventana de usuarios.

Ingresar a la ventana "Ventanas del sistema" y seleccionar la ventana "Usuarios".

Cambiar a la pestaña "Pestañas", y agregar una nueva pestaña con el nombre "Roles", que tenga asociada la tabla "Roles".

Importar los campos para la pestaña.

En la pestaña Roles, asignar como columna primaria la columna "md_usuarios_id".

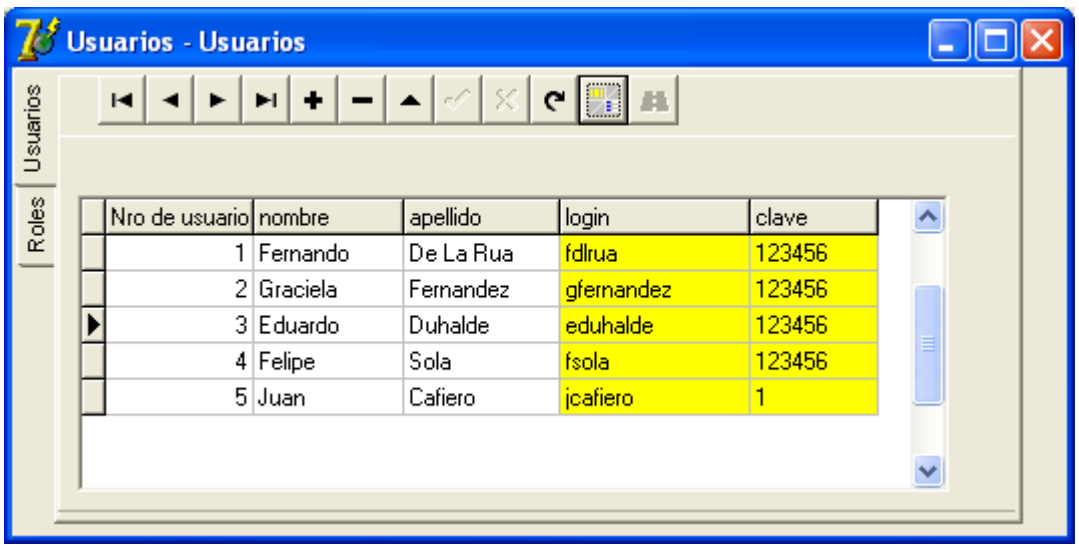
3.6.1.12. Creación de la relación entre usuarios y roles

Con el objetivo que la pestaña de roles muestre sólo los roles que el usuario tiene asignados, se debe crear una relación "que apunte" a la tabla de usuarios, y especificar dicha relación en la columna "md_usuarios_id" del rol.

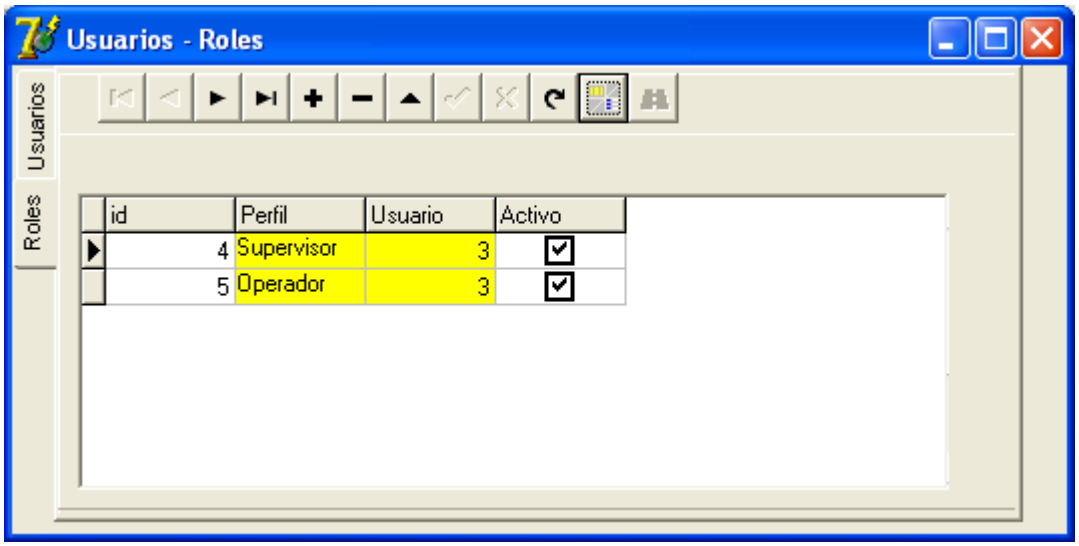
La relación seria la siguiente:

Tabla	Columna	Descripción	Mostrar columna
MD_usuarios	MD_Usuarios_ID	Relacion a usuarios	Login

El resultado final es la ventana:



En la cual cuando se pasa a la pestaña de perfiles se visualizan los perfiles del usuario:

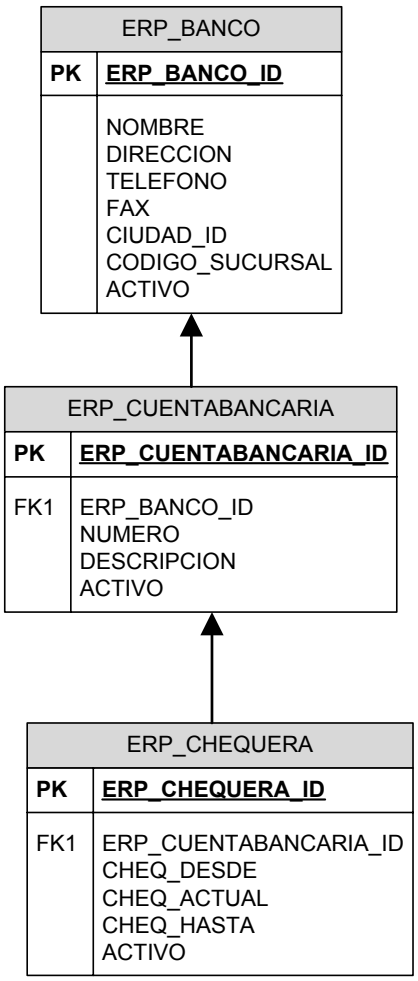


3.6.2. Caso: Reingeniería del circuito bancos de un sistema ERP

A continuación se presenta como caso de estudio la reingeniería de el mantenimiento de las cuentas bancarias de un sistema. Para hacer más interesante este caso, se lo comparará al equivalente en un sistema ERP desarrollado en JDeveloper.

El objetivo del mismo es mantener la información sobre los bancos con los que se opera, las cuentas bancarias y las chequeras que se tienen en uso para los pagos a proveedores.

Configuración de bancos



3.6.2.1. Paso 1: Creación de las tablas en la base de datos

Se procede a crear las tablas del modelo entidad-relación en la base de datos. El script de creación es el siguiente:

```
CREATE TABLE `erp_banco` (
 `ERP_BANCO_ID` int(11) NOT NULL auto_increment,
 `NOMBRE` varchar(25) default NULL,
 `DIRECCION` varchar(50) default NULL,
 `TELEFONO` varchar(20) default NULL,
 `FAX` varchar(20) default NULL,
 `CIUDAD_ID` int(11) default NULL,
 `CODIGO_SUCURSAL` varchar(5) default NULL,
 `ACTIVO` int(11) default NULL,
 PRIMARY KEY (`ERP_BANCO_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `erp_cuenta_bancaria` (
 `ERP_CUENTABANCARIA_ID` int(11) NOT NULL auto_increment,
 `ERP_BANCO_ID` int(11) default NULL,
 `DESCRIPCION` varchar(50) default NULL,
```

```
	`ACTIVO` int(11) default NULL,
	PRIMARY KEY (`ERP_CUENTABANCARIA_ID`),
	KEY `ERP_BANCO_ID` (`ERP_BANCO_ID`),
	CONSTRAINT `erp_cuenta_bancaria_fk` FOREIGN KEY (`ERP_BANCO_ID`) REFERENCES `erp_banco` (`ERP_BANCO_ID`)
	) ENGINE=InnoDB DEFAULT CHARSET=latin1;

	CREATE TABLE `erp_chequera` (
	`ERP_CHEQUERA_ID` int(11) NOT NULL auto_increment,
	`ERP_CUENTABANCARIA_ID` int(11) default NULL,
	`CHEQ_DESDE` int(11) NOT NULL default '0',
	`CHEQ_HASTA` int(11) default NULL,
	`CHEQ_ACTUAL` int(11) default NULL,
	`ACTIVO` int(11) default NULL,
	PRIMARY KEY (`ERP_CHEQUERA_ID`),
	KEY `ERP_CUENTABANCARIA_ID` (`ERP_CUENTABANCARIA_ID`),
	CONSTRAINT `erp_chequera_fk` FOREIGN KEY (`ERP_CUENTABANCARIA_ID`) REFERENCES `erp_cuenta_bancaria` (`ERP_CUENTABANCARIA_ID`)
	) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Se deben observar los siguientes puntos:

- ✓ Las claves primarias de las tablas se componen de "Nombre de la tabla" + "_ID"
- ✓ Las claves primarias son auto-incrementales

3.6.2.2. Paso 2: Incorporación de las tablas a los metadatos

Utilizando el framework, se ingresa a la ventana "Tablas", y se crea un registro con cada tabla. Luego se importarán los campos de la tabla. El siguiente paso es revisar la definición de las columnas que se importaron. Entre los valores a configurar se encuentran:

- ✓ Tipo de dato (recordar que el sistema no puede diferenciar valores enteros de valores lógicos).
- ✓ Longitud del campo
- ✓ Descripciones: Es conveniente completar las descripciones de las columnas, a fin de realizar una documentación de la funcionalidad de la misma.

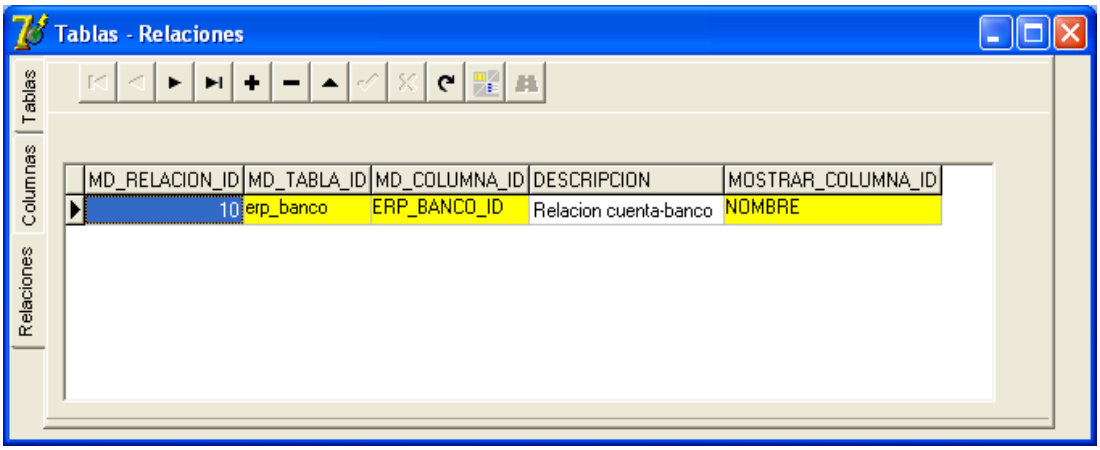
3.6.2.3. Paso 3: Especificación de relaciones entre tablas

El modelo presentado de bancos tiene las siguientes relaciones:

Cuenta Bancaria -> Banco: Una cuenta Bancaria pertenece a un banco.

Chequera -> Cuenta Bancaria: Una chequera pertenece a una cuenta bancaria.

Para especificar las relaciones, se debe seleccionar la columna que actúa como vínculo entre las tablas. Luego ir a la pestaña de relaciones e ingresar los datos de la relación.

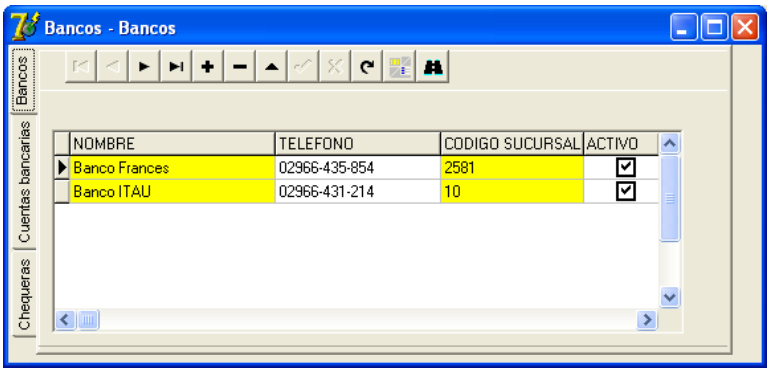
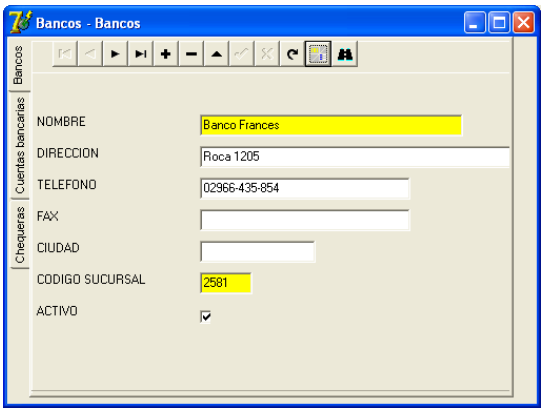
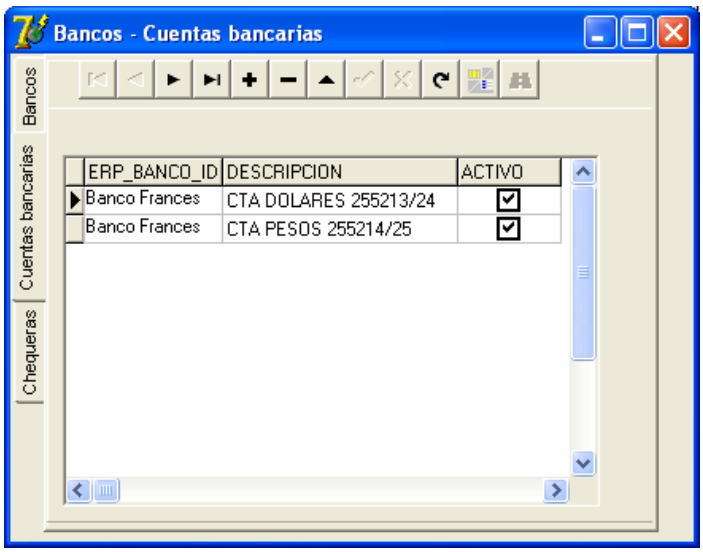
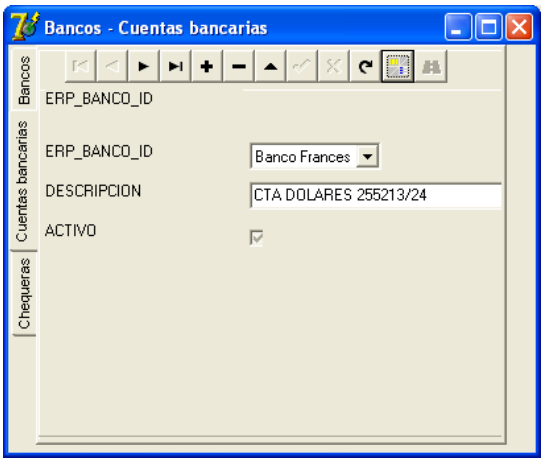
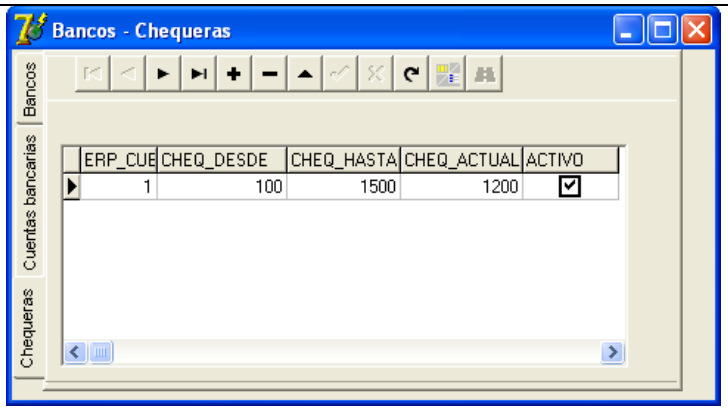
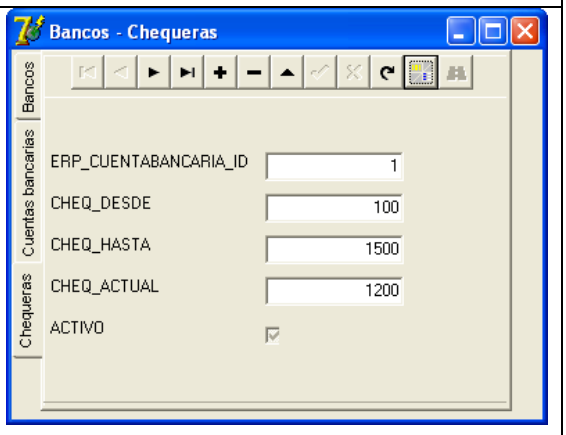


3.6.2.4. Paso 4: Llamado a la ventana

Para invocar la ventana recién creada, desde el menú de la aplicación se debe ejecutar el código:

PrepararVentana('Bancos');

El aspecto final, para la configuración de bancos es el siguiente

	Vista de Grilla	Vista de Detalle
Bancos		
Cuentas		
Chequeras		

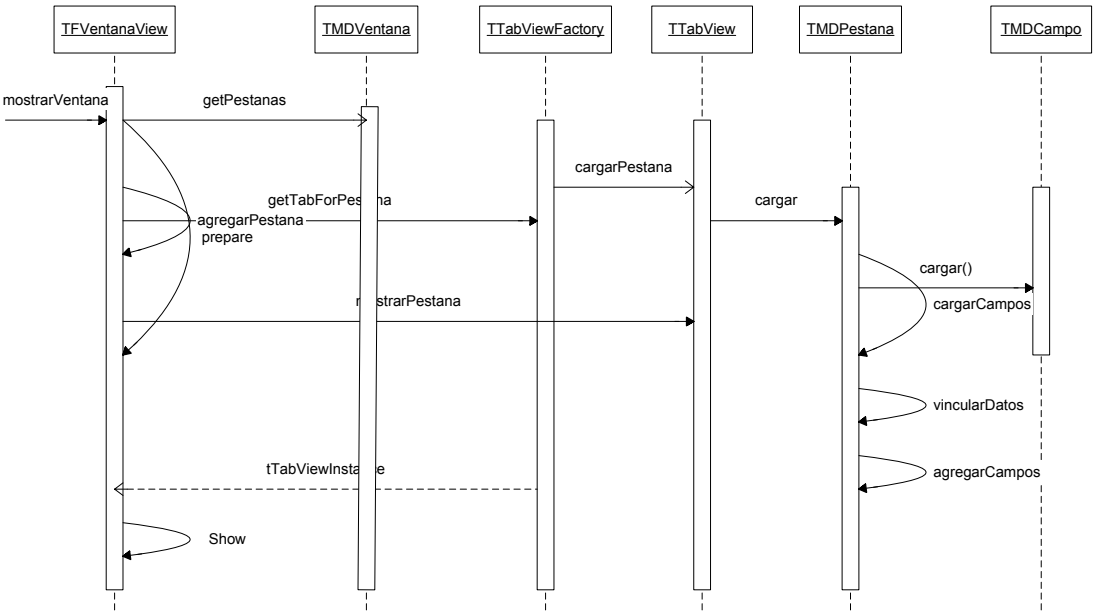
3.6.2.5. Comparación

Sistema	Horas empleadas
Ejemplo	3
ERP	30

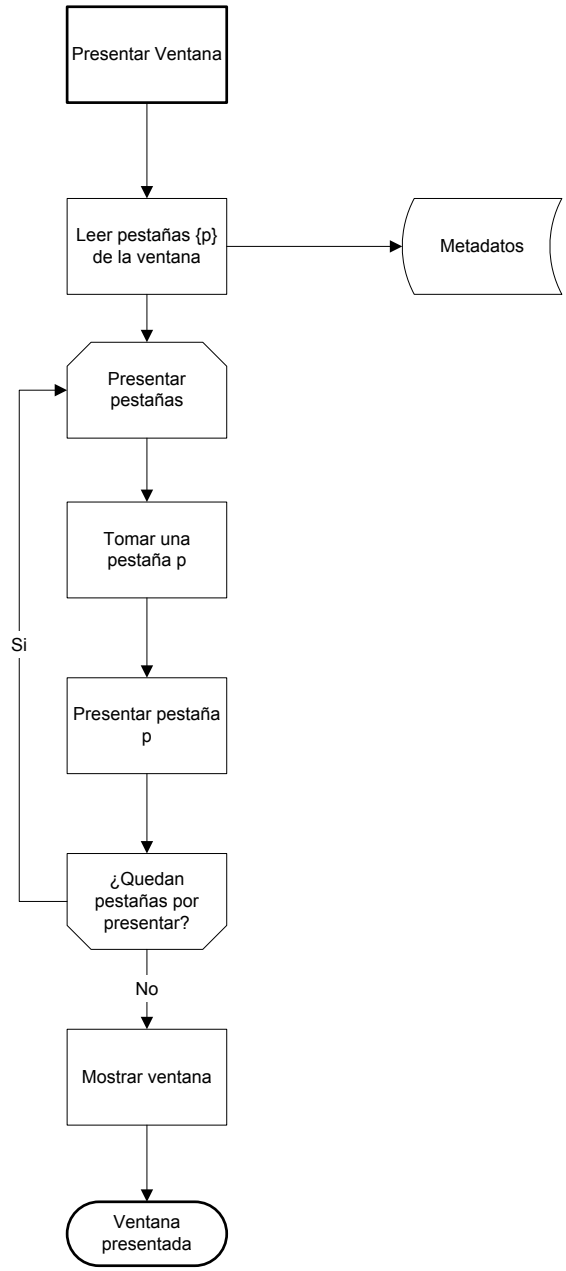
Fuente: Archivos internos Disytel – www.disytel.com

3.7. Desarrollo del framework

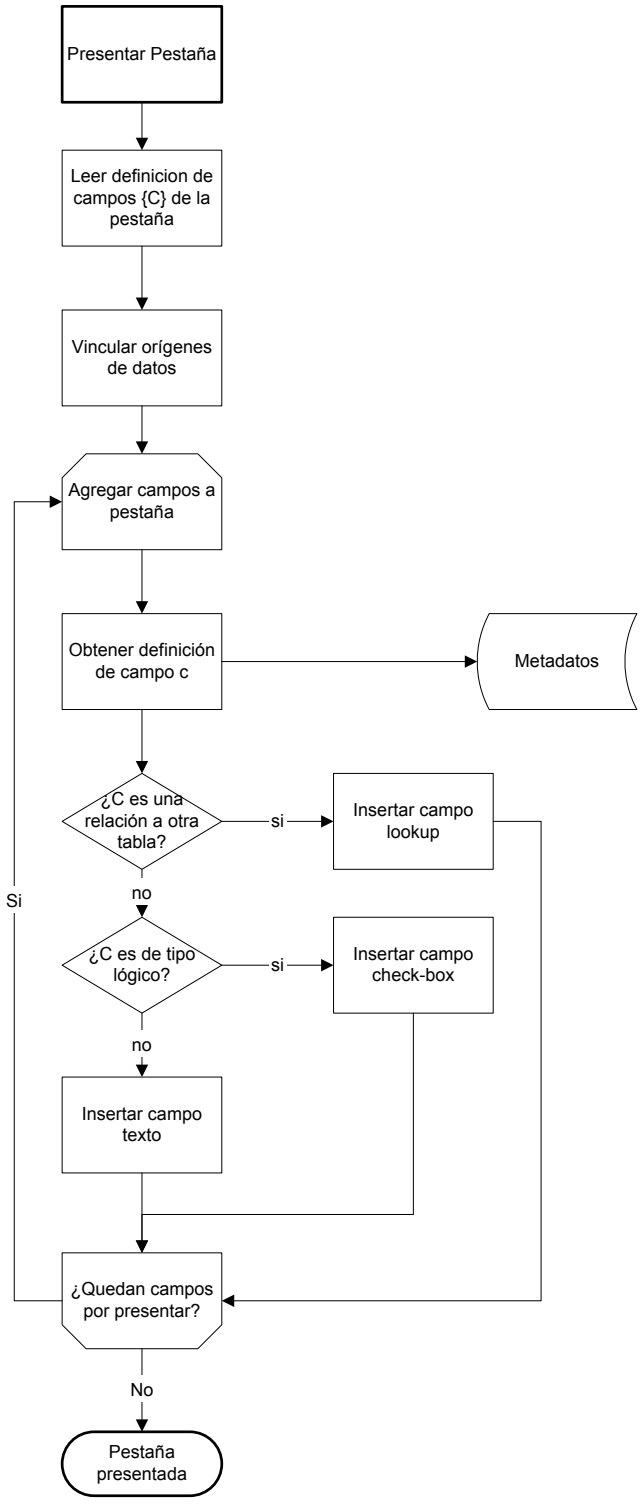
3.7.1. Secuencia de armado de una ventana



3.7.2. Diagrama de flujos de armado de una ventana



3.7.3. Diagrama de flujos de armado de una pestaña



3.8. Restricciones encontradas

- ✓ Claves primarias en tablas: Por necesidades programáticas, se exige que la clave primaria de las tablas del modelo de datos tengan como nombre:

  "nombre de tabla" + "_ID"

- ✓ Manejo de grillas estándar en Delphi: El componente "DBGrid" en Delphi necesitó adiciones importantes para lograr una funcionalidad adecuada al entorno propuesto. Gran parte del tiempo de programación se tuvo que emplear en personalizar dicho componente. Se evaluó la posibilidad de utilizar algún otro componente pero no fue posible al no encontrar uno que fuera gratuito.

4. Conclusiones y trabajo futuro

4.1. Conclusiones

A la hora de revisar el trabajo realizado, se alcanzaron las siguientes conclusiones:

- ✓ Se diseñó un modelo de metadatos que sirva de base para la construcción dinámica de la interfaz de usuario. A su vez, el estudio y la comprensión de los metadatos necesarios sirve como documentación parcial del sistema. Los metadatos de la aplicación terminan definiendo el diccionario de datos según el enfoque clásico de la Ingeniería del Software [PRES93].

- ✓ Se derivó un conjunto de clases o framework que, utilizando los metadatos, generan en forma dinámica la interfaz de usuario. Las modificaciones en el modelo de datos subyacente, y por lo tanto de los metadatos, se reflejan en el sistema sin necesidad de compilar; alcanza con salir y entrar a la ventana modificada.

- ✓ Se plantearon los mecanismos necesarios para la extensión del framework. El método planteado resulta ser sencillo y potente. Sin embargo, cualquier modificación realizada en el código de la aplicación requiere ser compilada.

- ✓ Utilizando los mecanismos del propio framework se creó un esquema de seguridad y usuarios embebido, que queda incorporado a las aplicaciones generadas a partir de aquel.

- ✓ Utilizando el mecanismo previsto de extensión del framework se logró proveer interfaces especiales para el mantenimiento de los metadatos. Fundamentalmente, proveer herramientas para la incorporación en los metadatos del modelo subyacente residente en el motor de bases de datos. Por ejemplo, importar la estructura de una tabla.

4.2. Trabajo futuro

4.2.1. Tablas de alto volumen de datos: En los casos en los que el volumen de registros de una tabla exceda un parámetro a establecer, el sistema siempre presentará un filtro antes de mostrar datos. Con este método se pretende evitar sobrecargar la base de datos y la red con consultas que retornen demasiados registros y tengan un impacto negativo en la performance. Se implementará como un paso previo a mostrar la ventana en el cual se consultará en la base de datos la cantidad de registros existentes (dato que proviene de las estadísticas del motor). Si la cantidad excede el parámetro, no se mostrará ningún registro hasta que el usuario no aplique un filtro.

4.2.2. Paginado: Se definirá un parámetro de cantidad de registros por página, que será la cantidad de registros que se mostrarán por pantalla. El usuario será capaz de navegar a través de los registros cambiando de páginas, disponiendo la aplicación de los controles adecuados para tal fin.

4.2.3. Reportes: Se incluirá un sistema de reportes basado en los metadatos. La estructura del sistema de reportes obtendrá de los metadatos las relaciones del sistema, y permitirá navegar en los datos, a través de los reportes relacionados.

4.2.4. Sitio Web: Utilizando las definiciones disponibles en los metadatos se planea hacer un motor de páginas para poder consultar los datos del sistema. Este enfoque permitiría ampliar el sistema, agregándole propiedades B2B.

Referencias

[LAR98] Craig Largman – Applying UML and patterns, An Introduction to Object Orientes Analysis and Design. 1998 - Prentice Hall

[PRES93] Roger S. Pressman - Ingeniería de software, un enfoque práctico – Mc Graw Hill - 1993

[UCorn1] Tutorial de digitalización de imágenes – Biblioteca de la Universidad de Cornell
http://www.library.cornell.edu/preservation/tutorial-spanish/metadata/metadata-01.html

[UCORN2] Some background on user interfaces:
http://cfg.cit.cornell.edu/cfg/design/bkg.html

[COMP] Compiere ERP: http://www.compiere.org

[OEX] OpenXpertya: http://www.openxpertya.org

[UIML] UMIL Specification:
http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf

[XUL] XUL: http://www.xulplanet.com

[XIML] XIML: A Common representation for interaction data. - Angel Puerta and Jacob Eisenstein.
http://www.ximl.org/documents/XIMLBasicPaperES.pdf

[Walsamakis] Generación Automática de Código a partir del modelo de datos
APU Walsamakis, Máximo, APU Mansutti, Marcos,Lic. Rodolfo Bertone, Lic. Raul Champredonde – III Lidi - UNLP

[Cooper 2001] Presos de la tecnología - Cooper Alan – Pearson Educación.

[Bodart 1994]. Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot,I. & Vanderdonckt, J. (1994) A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype. In Proc. of Eurographics Workshop on Design, Specification, Verification ofInteractive Systems, (p. 25-39).

[Bodart 1995] F. Bodart y J. Vanderdonckt, “Towards a Systematic Building of Software Architectures: The TRIDENT Methodological Guide” Design, Specification and Verification of Interactive Systems, pp. 262–278, Jun. 1995.

[Puerta 1996] The MECANO Project:Comprehensive and Integrated Support for Model-Based Interface Development
http://www.isys.ucl.ac.be/bchi/cadui/96/files96/Puerta-CADUI96.pdf

[RAE] Real academia española
http://www.rae.es/

[FREE1] The free dictionary: framework definition
http://www.thefreedictionary.com/framework

[FREE2] The free dictionary: dinamic definition
http://computing-dictionary.thefreedictionary.com/Dynamic

[Wiki] Wikipedia: Interfaz de usuario
http://es.wikipedia.org/wiki/Interfaz_de_usuario

[Wiki metadato] Wikipedia: Interfaz de usuario
http://es.wikipedia.org/wiki/metadatos

[NIELSEN] Nielsen, J. (1991a) Assessing the Usability of a User Interface Standard
http://www.useit.com/papers/standards.html

Apéndice A: Diagramas de clases

Diagrama de clases – Metadatos

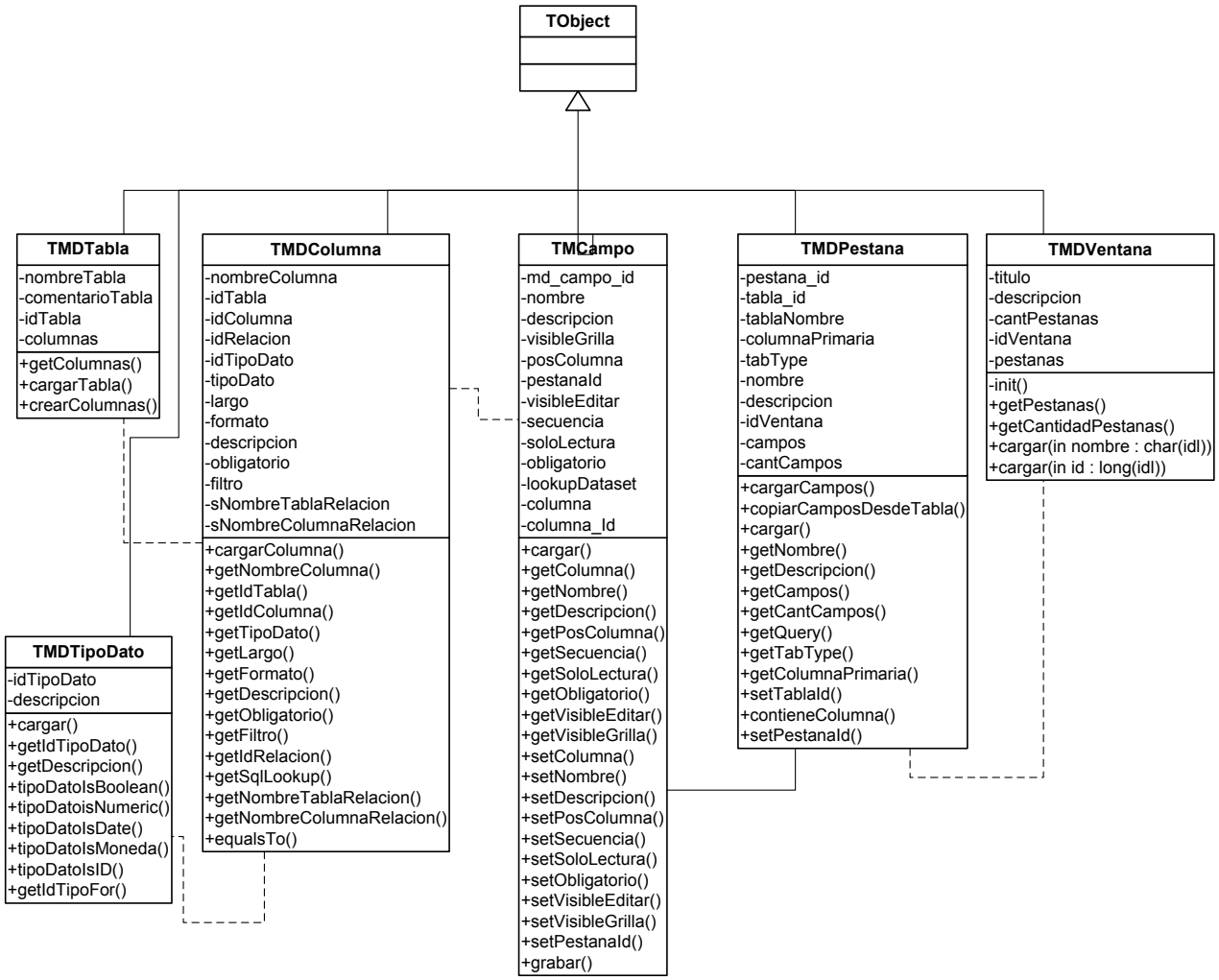
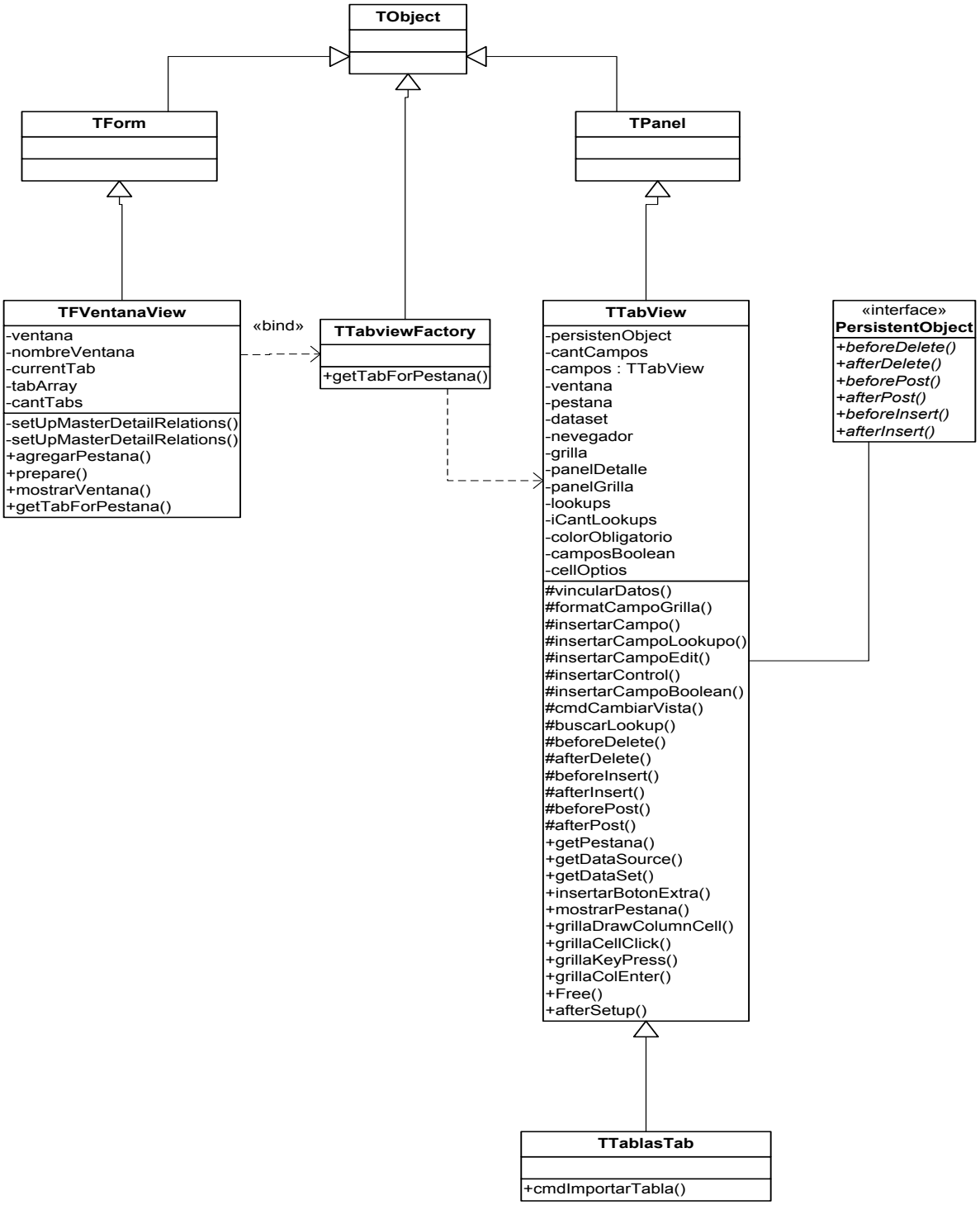
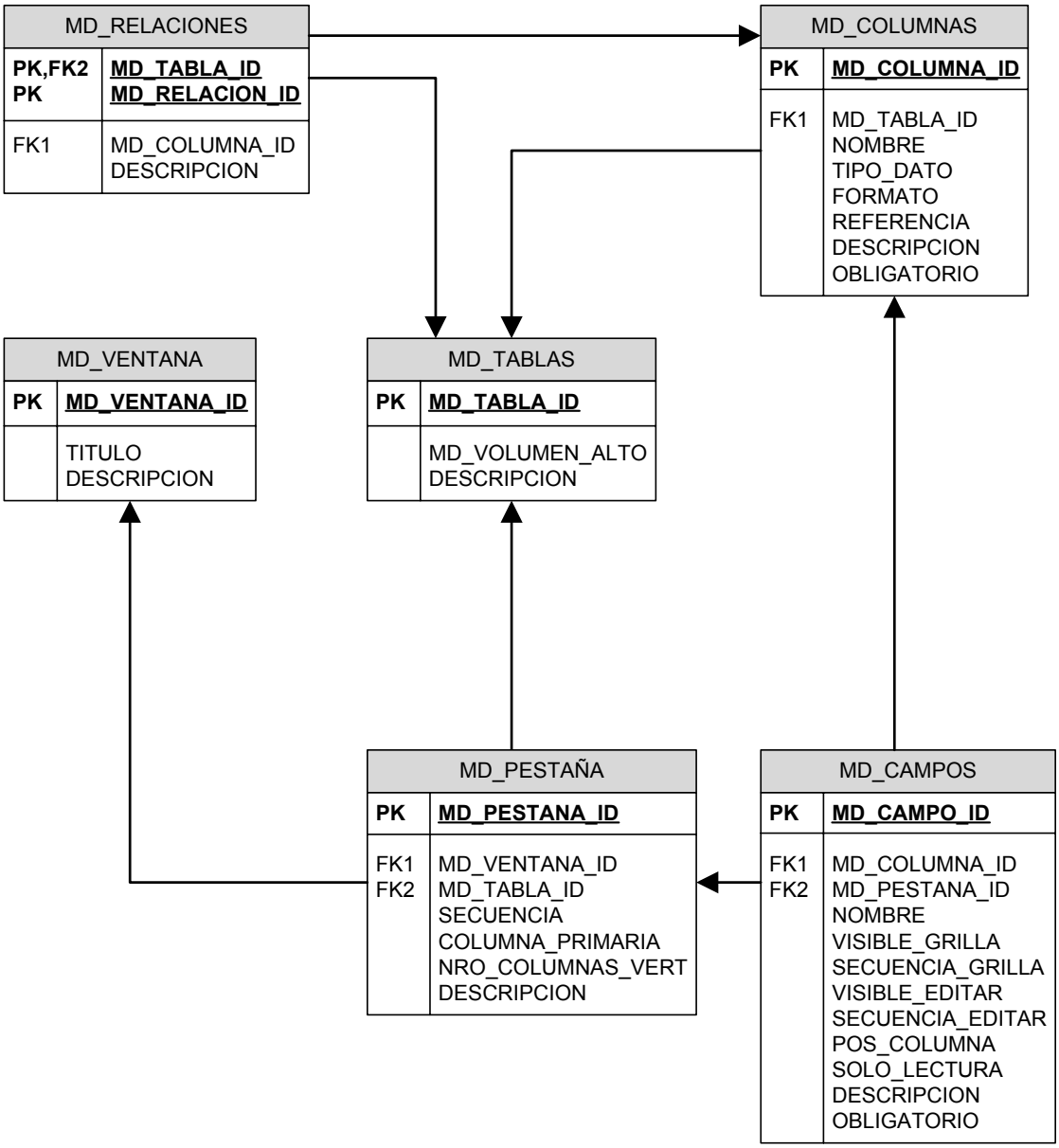


Diagrama de clases – Capa de presentación



Apéndice B: Diagrama de Entidades-Relaciones (Sólo metadatos)



Apéndice C: Configuración de MySQL y conexión ODBC

Para implementar el sistema se utilizó la versión 4.1.11 de MySQL.

El driver utilizado es el oficial de MySQL, Versión 3.51.11.00.

Se adjunta la configuración de ODBC para el driver MySQL. Esta es la configuración necesaria para el ambiente propuesto trabaje adecuadamente.