CONSEJO FEDERAL DE INVERSIONES

PROVINCIA

MENDOZA

TITULO

PROGRAMA DE MODERNIZACIÓN INTEGRAL DEL ESTADO DE MENDOZA. "INTEGRACIÓN DE BASE DE DATOS"

EXPERTO

A.U.S. JULIO MONETTI

Informe Final. Octubre de 2003

Autoridades

PROVINCIA DE MENDOZA	CONSEJO FEDERAL DE INVERSIONES
GOBERNADOR DE LA PROVINCIA Ing. Roberto Iglesias **Secretario Administrativo Legal y Técnico** Dr. Claudio Romano	**SECRETARIO GENERAL** Ing. Juan José CIÁCERA **Directora de Coordinación** Ing. Marga VELÁSQUEZ CAO **Jefa de Área Red de Información** Lic. Alicia Noemí Rapaccini

Autor

A.U.S. Julio César Monetti

Colaboradores

Dr. Diego Enzo Amprino

Sr. Guillermo Arroyo

Índice

Resumen

Ya sobre el final de nuestro trabajo, nos aprestamos a realizar las conclusiones pertinentes y programadas para la presente etapa.

Con el ánimo de poder transmitir claramente el trabajo realizado, como así también de documentar los procedimientos utilizados con el fin de poder emprender nuevos desarrollos, elevamos el presente informe. El mismo contiene la descripción de los desarrollos llevados a cabo por el personal técnico en la última etapa del proyecto "Integración de Bases de Datos" y la conciliación con las tareas de las etapas anteriores para exponer el resultado obtenido.

Se ha realizado un trabajo de relevamiento y análisis de la información durante el periodo de trabajo del proyecto, que consideramos desde nuestro punto de vista, adecuado para poder definir un esquema común de trabajo en la delineación de nuevas estructuras de datos que abarquen las preexistentes.

Para este fin creemos conveniente ejercitar el cruzamiento de datos con pequeñas estructuras, como el trabajo realizado en la segunda etapa de nuestro proyecto. Este ejercicio permitió, y permitirá en trabajos similares detectar distintos problemas técnicos y operativos en el cruzamiento de la información.

Como se expresó anteriormente, la necesidad de reutilización y lectura de información por parte de las distintas áreas hace necesaria la creación de un

esquema dinámico a corto plazo. Más allá de la reunión total de la información en un almacén común de datos (*datawarehousing*), situación que no consideramos factible en el corto plazo, es posible la generación de **rutinas de recolección y transformación** de datos utilizables por cada uno de los administradores con la finalidad de acceder fácilmente a los diferentes formatos.

En la sección de desarrollos exponemos un modelo análisis, diseño y homogeinización de datos/información en el estándar XML.

Metodología Actual de Trabajo

Se ha mantenido la metodología utilizada en las anteriores etapas, la cual comprende como actividades globales el permanente relevamiento en las áreas con el objeto de perfeccionar datos (metadatos) e información sobre la metodología de trabajo interna, la investigación en el campo tecnológico y conjuntamente el desarrollo de procedimientos que permitan, por intermedio de la tecnología observada poder palear las deficiencias de información observadas en el relevamiento.

Objetivos del Presente Informe

- Determinación de un estándar de traspaso y reutilización de datos.
- Conciliación de las etapas del trabajo.
- Transmitir la Conclusión del Trabajo a las diferentes áreas de gobierno participantes.

Actividades

Integración de las Bases de Datos

Hemos analizado a lo largo de nuestro trabajo las distintas opciones técnicas y operativas para la integración de Bases de Datos, no obstante ello preferimos esperar hasta este momento para formalizar un modelo (el cual hemos tenido en mente desde el primer momento).

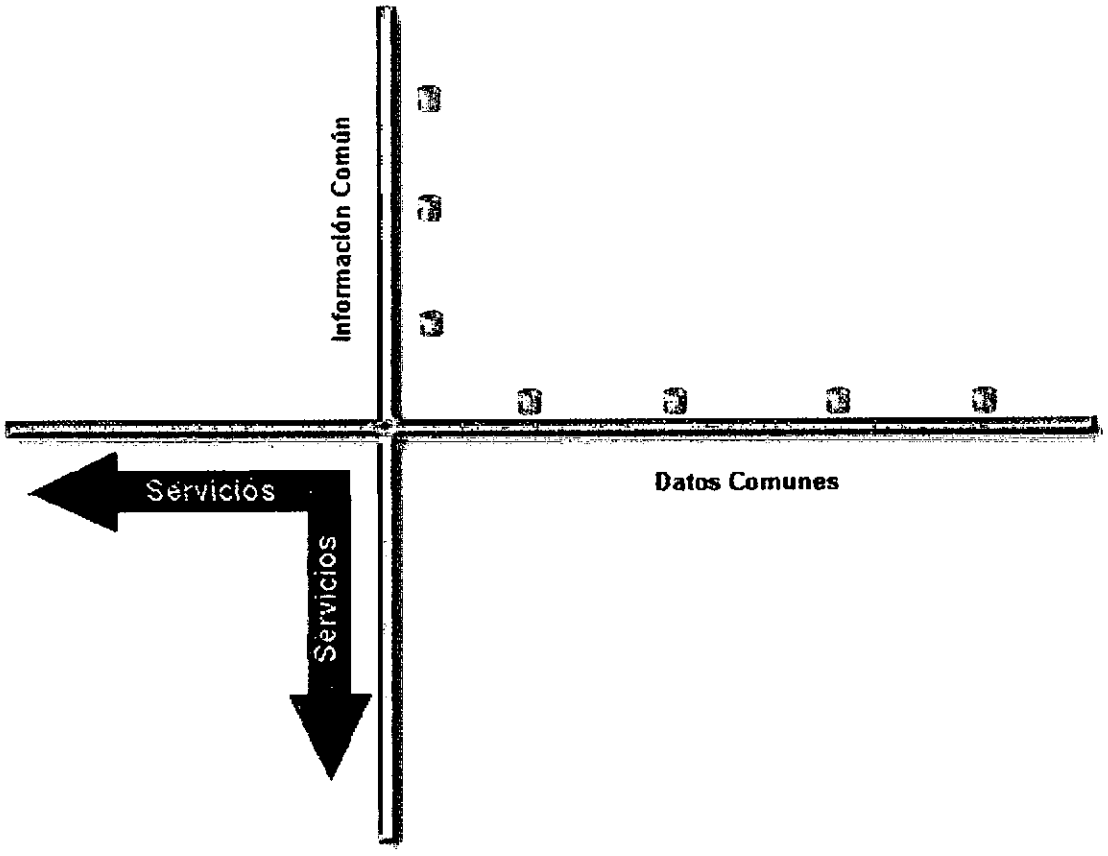
Para comenzar, utilizaremos un sistema de ejes para modelar la integración de las distintas áreas, ubicando en el eje de abcisas todas aquellas áreas que proveen información, y en el eje de coordenadas todas aquellas áreas que son usuarias de información provistas por las primeras.

Tal sistema de ejes permitirá ubicar en forma gráfica y precisa cada una de las relaciones provisión/utilización de información.

Otro modelo a utilizar permitirá ubicar en el eje de abcisas aquellas áreas que por su carácter presentan una dependencia jerárquica una de otra. Y en el eje de coordenada áreas que intercambian información, pero se encuentran en un mismo nivel jerárquico.

En el primer caso podremos observar **flujos de información** que existen entre áreas afines y con dependencia jerárquica. En este caso modelaremos el **flujo**

de información. En el segundo caso se modelará el intercambio de datos (y eventualmente intercambio de información).



Como base de este modelo consideramos, como lo muestra la figura anterior, dos tipos distintos de integración: **una horizontal, y una vertical**.

Esta estructura utilizada para modelar datos se encuentra intimamente relacionada con la **matriz de usuarios** presentada en la primera etapa de nuestro trabajo.

Brinda información a	Secretaría Administrativa, Legal y Técnica	Dirección General de Escuelas	Rentas	Ministerio de Seguridad y Justicia
Secretaría Administrativa, Legal y Técnica	–	R1	R3	R2
Dirección General de Escuelas	–	–	R4	–
Rentas	R5	R6	–	R7
Ministerio de Seguridad y Justicia	–	–	R8	–

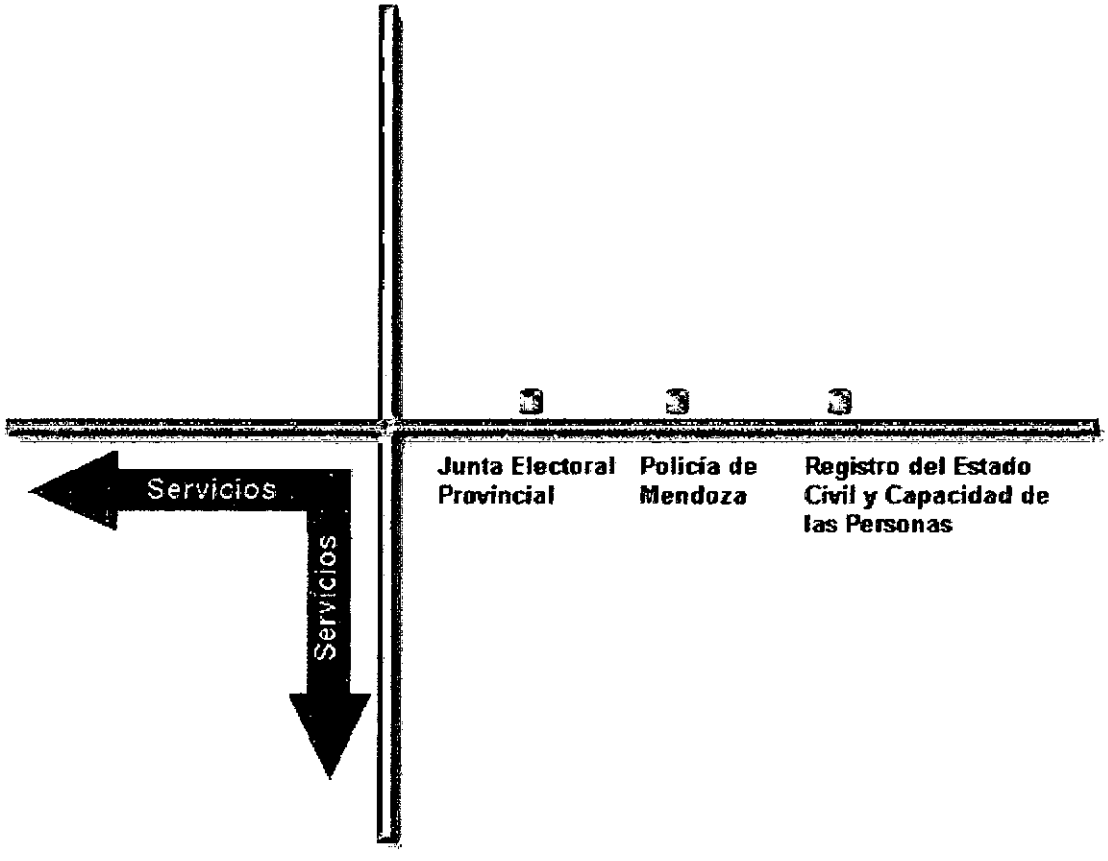
Recordemos también que las necesidades de información han sido establecidas de acuerdo a los comentarios realizados por personal técnico y jerárquico de cada área, donde los mismos explicitan la información que requieren de otros organismos, la cual le puede ser brindada o no.

Utilizaremos en este modelo también la notación propuesta en la primer parte de nuestro trabajo para clasificar los datos de acuerdo a:

Datos Elementales	Son los datos que permiten identificar una persona
Datos de la Aplicación	Son los datos que identifican una persona pero dentro del contexto de la aplicación que lo utiliza.
Metadatos	Son los datos que no tienen relevancia para identificar una persona. Son datos suplementarios que permiten que la aplicación actúe de acuerdo a los requerimientos.
No Relevante	

Integración Horizontal

La integración horizontal ha sido observada teniendo en cuenta todas aquellas áreas (con distintos fines) que contienen datos (no información) comunes.



Ya que la existen **diferentes motivos** de explotación de información, no analizamos los fines de la misma, sino solamente los datos en común y los diferentes formatos de los mismos.

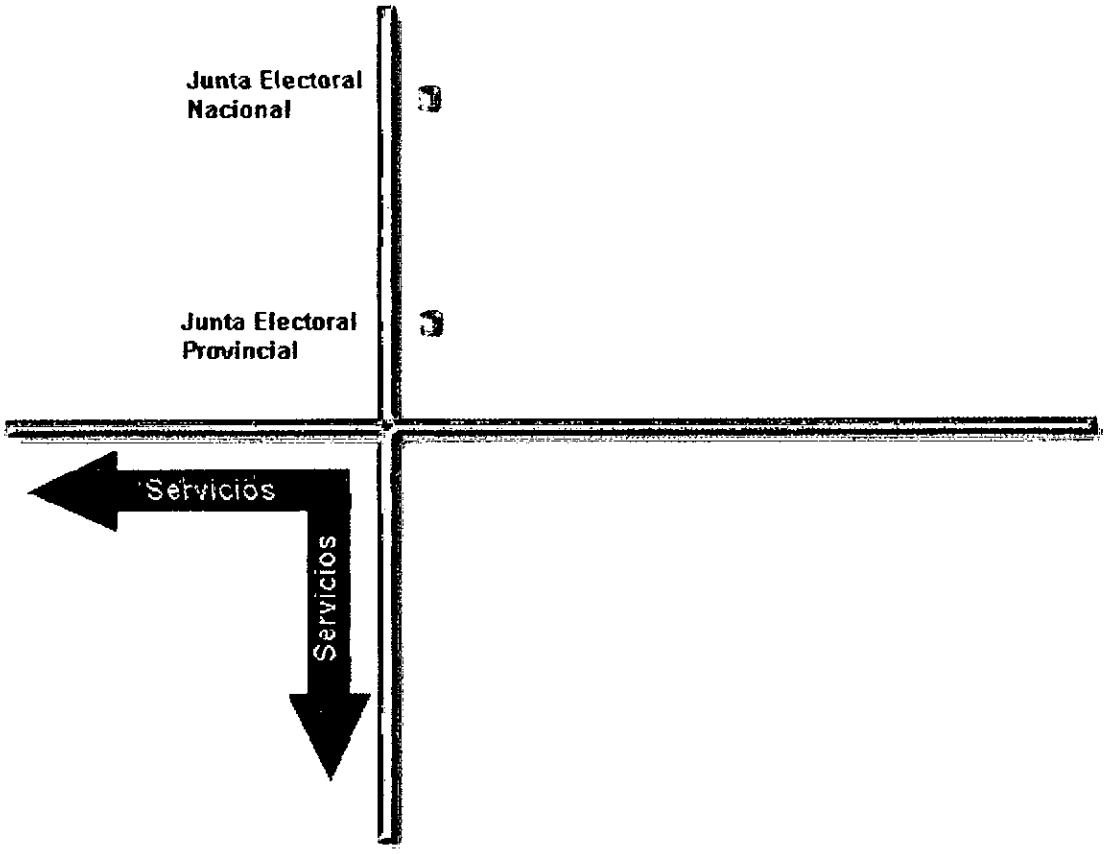
Este es el momento donde comenzaremos a utilizar el **diccionario de datos** propuesto en las primeras etapas de nuestro trabajo, puesto que no resulta

suficiente el **cruzamiento y análisis de estructuras y elementos de datos** llevados a cabo en etapas anteriores.

Con el objeto de realizar un cruzamiento completo y preciso, utilizaremos los sistemas de explotación programados desde la etapa anterior y finalizados en la presente, basándonos en el relevamiento de estructuras de datos y posterior volcado en el diccionario de datos común.

Integración Vertical

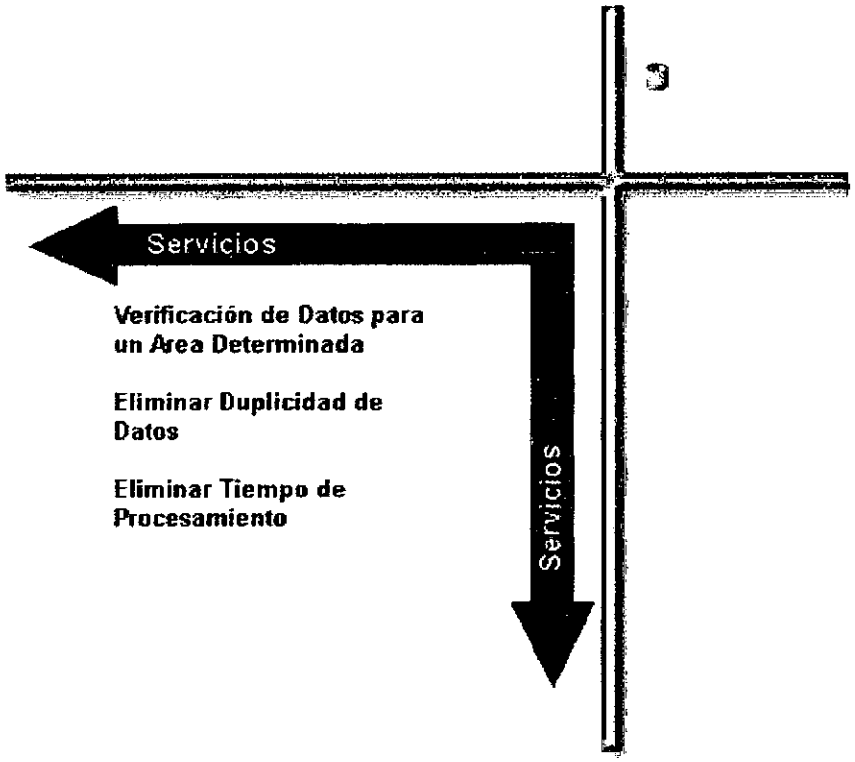
Análogamente, definimos la integración vertical de datos como la utilización de datos comunes pertenecientes a áreas públicas cuya función exige la manipulación de datos e información en común.



En contraposición a la integración horizontal definida anteriormente, en este tipo de integración, las distintas áreas necesitan explotar los mismos datos para generar la misma información. Esta situación permite un mejor acceso a las

bases de datos por medio del personal técnico, puesto que pertenecen a una misma rama de gobierno, aunque jerárquicamente distinta.

De lo anterior, definiremos un cuadrante de servicios, donde suponemos la integración ya realizada.

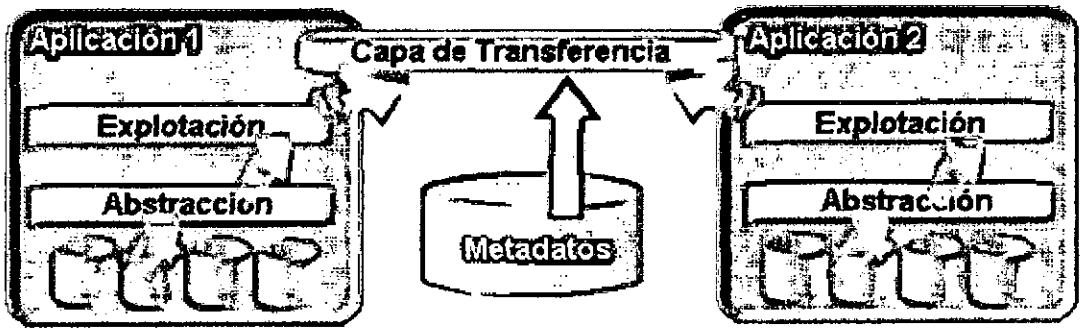


Para los fines de prestar un servicio de comunicación de información, no tendremos en cuenta si la integración es horizontal o vertical. Técnicamente las tareas que se puedan llegar a realizar para homogeneizar los datos serán las mismas.

Relaciones entre Sistemas de Explotación

La relación lograda entre sistemas de explotación (muchas veces a nivel de prototipo) ha resultado satisfactoria para los fines del presente trabajo.

La misma permitió analizar flujos de información redundante, carencia de la misma, etc.



El gráfico anterior, muestra el flujo de información entre dos aplicaciones distintas de acuerdo a los siguientes pasos:

1. Cada aplicación realiza una abstracción de sus propios datos físicamente almacenados. Esto permite obtener elementos de datos complejos y vistas globales de la información.
2. La capa de abstracción de datos provee los mismos a la capa de explotación, la cual tiene por objeto satisfacer los fines del software mismo a través de la información generada.

3. Se agregará una nueva capa de abstracción de datos (la capa de transferencia) que tendrá por objeto la transformación de formatos de almacenamiento (físico o lógico) de una aplicación a otra. Este proceso se llevará a cabo basándonos en la base de metadatos (o diccionario común de datos).

Relación entre las Áreas de Trabajo

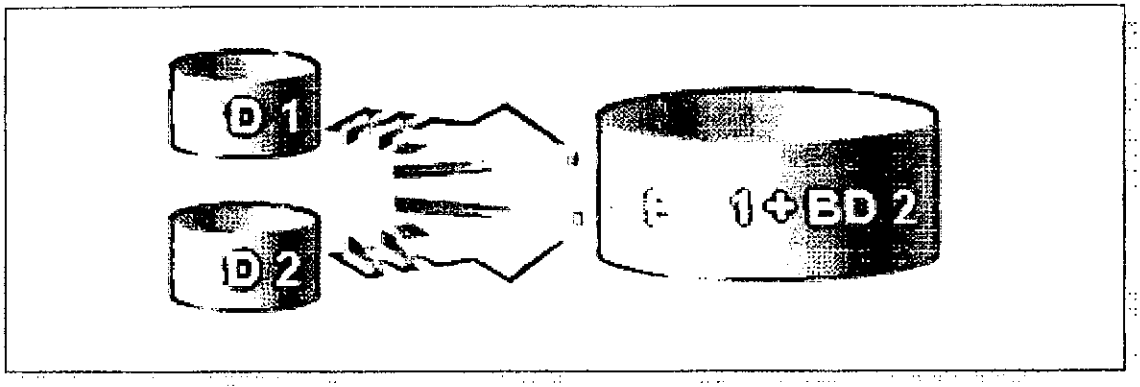
Se ha logrado un muy buen ambiente de trabajo con algunas oficinas técnicas de la Policía de Mendoza. Se han considerado vías alternativas al almacenamiento común de datos (*Datawarehousing*) a la hora de reutilizar datos, aceptando cada una de las oficinas técnicas esta metodología de trabajo.

Una de las principales preocupaciones del grupo de trabajo del proyecto, fue y es aún la buena predisposición de técnicos y personal jerárquico de cada una de las áreas a la hora de comprender el fin de nuestro proyecto.

Si bien se ha encontrado una gran predisposición a colaborar en el cruce datos por parte de las distintas áreas, aún se toma muy difícil plantear la unificación total (físicamente hablando), por lo tanto ha sido muy bien aceptada la propuesta de crear **funciones de transferencia** entre una base de datos y otra.

En todo momento intentamos comunicar al personal técnico de la forma más clara posible las principales posibilidades técnicas de reutilización de datos:

1. Creación de un almacenamiento común.



Este modelo se consideraría el más adecuado para la integración de datos, ya que permite reunir en un contenedor común (físicamente) la totalidad de la información, minimizando costos de administración.

2. Creación de funciones de transferencia.

Al no ser posible reunir los datos en un contenedor común, según nuestras observaciones por problemas operativos a la hora de realizar la implementación final; se tornó necesario realizar una propuesta de reutilización y cruzamiento de datos. Para tal fin definimos funciones de lectura y transferencia de datos entre ambas bases de datos.

Transmisión de la Conclusión del Trabajo

Recomendaciones

Creación de un Espacio de Almacenamiento Común

Se han realizado reuniones varias con personal técnico y operativo de las distintas áreas de trabajo, analizando los estándares ya diseñados en el marco de nuestro proyecto de Integración de Bases de Datos.

Con ellos se ha llegado a la conclusión de que es sumamente importante la integración futura de las bases de datos. No obstante ello, una solución rápida para la reutilización de información en tiempo real, sin tener en cuenta las barreras que nos presenta la variedad de plataformas y metodologías de trabajo, es la de la utilización de funciones de transferencia/lectura entre las diferentes bases de datos.

Mantenimiento del Diccionario de Datos

Más allá de lograr el fin propuesto para nuestro proyecto, nos avocamos en todo momento a crear una necesidad de **documentación de datos** entre el personal técnico de áreas relacionadas. Esto es, en otras palabras, la permanente sugerencia a mantener normalizada y en condicionas las carpetas de sistemas

de información, y en forma particular el diccionario de datos, modelos de entidad y relaciones, y toda documentación que permita visualizar en forma gráfica las estructuras de datos utilizadas por sus sistemas de explotación.

Estándares XML

Principios del Lenguaje XML

El XML proviene de un lenguaje que ideó y desarrolló la empresa IBM allá por los años 70s. El lenguaje original ideado por la empresa IBM se llama GML (*General Markup Language*) y surgió por la necesidad que tenían en la empresa de almacenar grandes cantidades de información de temas diversos, haciendo uso de estándares comunes.

Imaginemos por un momento la cantidad de documentación que generaría IBM sobre todas las áreas en las que trabajaba e investigaba, y la cantidad de información que habrá generado hasta hoy. Así pues, la empresa necesitaba una manera de almacenar y distribuir la información. Para ello los expertos informáticos de IBM inventaron GML, un lenguaje con el que poder clasificarlo todo y escribir cualquier documento para que se pueda luego procesar adecuadamente.

Este lenguaje gustó mucho a la gente de ISO, una entidad que se encarga de crear estándares de normalización para los procesos del mundo actual, de modo

que allá por el año 1986 trabajaron para normalizar el lenguaje, creando el SGML, que no era más que el GML estandarizado.

SGML es un lenguaje muy trabajado, capaz de adaptarse a un gran abanico de problemas y a partir de él se han creado los siguientes sistemas para almacenar información.

Por el año 1989, para el ámbito de la red Internet, un usuario que había conocido el lenguaje de etiquetas (*Markup*) y los hipervínculos creo un nuevo lenguaje llamado HTML, que fue utilizado para un nuevo servicio de Internet, la Web. Este lenguaje fue adoptado rápidamente por la comunidad y varias organizaciones comerciales crearon sus propios visores de HTML y riñeron entre ellos para hacer el visor más avanzado, inventándose etiquetas como su propia voluntad les decía. Desde 1996 hasta hoy una entidad llamada W3C ha tratado de poner orden en el lenguaje HTML y establecer sus reglas y etiquetas para que sea un estándar. Sin embargo el HTML creció de una manera descontrolada y no cumplió todos los problemas que planteaba la sociedad global de Internet. Este problema es sumamente importante que sea observado en nuestro proyecto, y podría ser resumido como el siguiente principio:

El aumento de información en forma descontrolada genera un caos tal que su explotación se puede volver imposible.

El mismo W3C en 1998 empezó y continúa, en el desarrollo de XML (*Extended Markup Language*). En este lenguaje se ha pensado mucho más y muchas personas con grandes conocimientos en la materia están trabajando todavía en su gestación. Pretendían solucionar las carencias del HTML en lo que respecta al tratamiento de la información. Problemas del HTML como:

- El contenido se mezcla con los estilos que se le quieren aplicar.
- No permite compartir información con todos los dispositivos, como pueden ser ordenadores o teléfonos móviles.
- La presentación en pantalla depende del visor que se utilice.

(...Y una vez crearemos una analogía de estos problemas con los encontrados a lo largo de nuestro trabajo).

Imagínese, una persona que conoce el HTML y lo difícil que puede llegar a ser entender su código, que tuviese que procesarlo para extraer datos que necesite en otras aplicaciones. Sería muy difícil saber dónde está realmente la información que busca, siempre mezclada entre etiquetas <FONT>, <TABLE>, <TD>, etc... Esto es una mala gestión de la información y el XML la soluciona.

Objetivos y usos del XML

El XML se creó para que cumpliera varios objetivos.

- **Que fuera idéntico a la hora de servir, recibir y procesar la información que el HTML, para aprovechar toda la tecnología implantada para este último.**

- **Que fuera formal y conciso desde el punto de vista de los datos y la manera de guardarlos.**

- **Que fuera extensible, para que lo puedan utilizar en todos los campos del conocimiento.**

- **Que fuese fácil de leer y editar.**

- **Que fuese fácil de implantar, programar y aplicar a los distintos sistemas.**

El XML se puede usar para infinidad de trabajos y aporta muchas ventajas en amplios escenarios. Veamos algunas ventajas del XML en algunos campos prácticos.

- **Comunicación de datos**. Si la información se transfiere en XML, cualquier aplicación podría escribir un documento de texto plano con los datos que estaba manejando en formato XML y otra aplicación recibir esta información y trabajar con ella.

- **Migración de datos**. Si tenemos que mover los datos de una base de datos a otra sería muy sencillo si las dos trabajasen en formato XML.

- **Aplicaciones Web**. Hasta ahora cada navegador interpreta la información a su manera. Con XML tenemos una sola aplicación que maneja los datos y para cada navegador o soporte podremos tener una hoja de estilo o similar para aplicarle el estilo adecuado.

Algunas ventajas sobre la utilización del XML como formato de datos para el transporte entre distintos sistemas en una red:

- **Optimización de la Información Buscada**

  Mediante la posibilidad de dividir los datos en elementos discretos, se convierte en fácil para los individuos el extraer la información realmente relevante desde distintas fuentes y reensamblarla en cualquier formato requerido. Esto ayuda a disminuir la sobrecarga de información existente en las búsquedas de algo específico, ya que el usuario solo recibe la información relevante.

- **Optimización del Ancho de Banda**

  Un tema importante en la actualidad es la optimización del uso del ancho de banda, como así también el tema de asignar prioridades sobre el ancho de banda.

Tecnologías Asociadas

Definición de Tipo de Documento (DTD)

Los DTDs son los que definen la estructura de los documentos XML, describen las etiquetas que aparecen y su disposición. Sirven para validar los documentos. En el DTD, plasmamos reglas que deben cumplir los documentos para que sean válidos.

Un ejemplo de todo esto podía ser el de un mensaje de correo electrónico. Para que éste sea correcto debe tener un destinatario, remitente, mensaje, adicionalmente puede, o no, tener otros campos como copia, *attachment* y prioridad de envío. La definición de un *mail* en un DTD sería **E.mail.dtd**.

Los DTDs pueden ser externos y hacer referencia a ellos dentro del XML, o pueden ir en un mismo documento el DTD y el XML. Los DTDs pueden también hacer referencias a entidades externas.

Schemas (Esquemas)

Los *Schemas* vienen a sustituir a los DTDs, el W3C no ha creado todavía ningún estándar al respecto, pero si existe una recomendación generada por dicha organización. También hay iniciativas como **SOX () de Commerce One´s**, de Microsoft.

Los *Schemas* sirven para lo mismo que los DTDs, pero estos son mucho más potentes.

Hojas de Estilo Extensibles (XSL)/XSLT

Las hojas de estilo sirven para dar formato (gráficamente) a la información. Es la forma de poder visualizar el contenido de los documentos XML en distintos formatos. A un mismo documento XML se le pueden aplicar las hojas de estilo que queramos e incluso podemos transformar un documento XML en otro distinto (XSLT). Se pueden aplicar filtros a los datos para visualizar únicamente los que nos interesen. El lenguaje XSL, dispone de estructuras condicionales y estructuras repetitivas para ser utilizadas en su código.

Modelo de Objetos de Documento (DOM)

EL DOM es una *API*, una librería de funciones para manipular el árbol DOM que sirve para tratar los documentos XML. Por medio de esta *API* podemos acceder a toda la información contenida en los documentos, podemos añadir, modificar y eliminar etiquetas. Las etiquetas son tomadas como nodos del árbol.

Se tratan simplemente de unas normas que indican a los desarrolladores la manera de acceder a los documentos. Estas normas incluyen una jerarquía de objetos que tienen unos métodos y atributos con los que tendremos que trabajar y que nos simplificarán las tareas relativas al recorrido y acceso a las partes del documento.

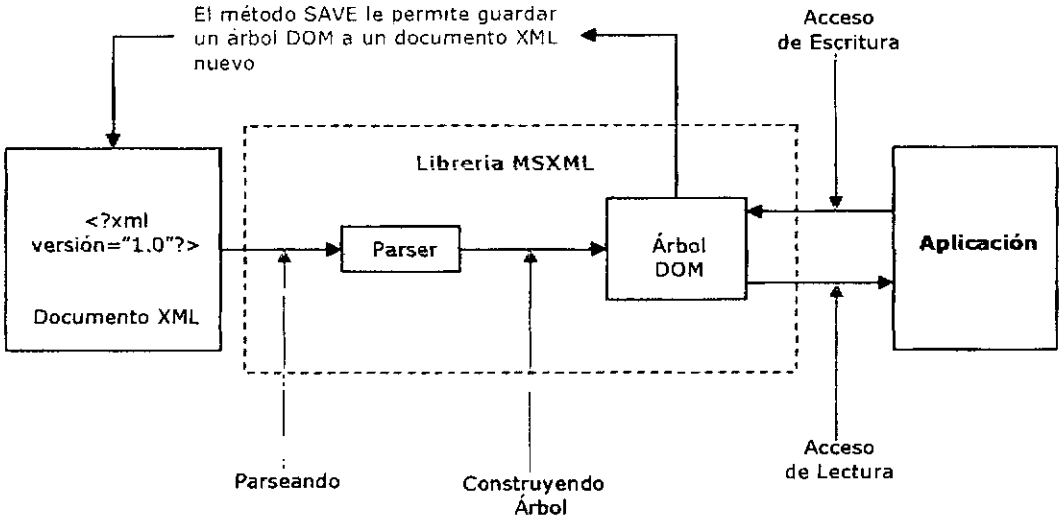
DOM implica la creación de una estructura arborescente en memoria que contiene el documento XML, y con él en memoria podemos hacer cualquier tipo de recorrido y acciones con los elementos que queramos.

Se puede programar con el lenguaje de programación que se desee para acceder a un documento XML. Los creadores del lenguaje son los responsables de crear unas *API* que cumplan las especificaciones de XML para que luego los desarrolladores de cada lenguaje las encuentren y puedan trabajar con ellas.

El Modelo de Objeto Documento (DOM) presenta una forma fácil y estandarizada de procesar un documento XML para aplicaciones o *scripts*. La implementación de Microsoft de DOM, el **MSXML *parser*** nos permite leer o crear un documento, obtener los errores si los hubiere, acceder y manipular la información y la estructura contenida en el documento y guardar nuevamente el documento en un documento XML.

DOM y MSXML

La implementación de DOM es simplemente una parte del *parser* MSXML. El diagrama siguiente muestra las tareas o pasos que se llevan a cabo en el proceso de *parsear* un documento y presentar la información a una aplicación o *script*.



El procesamiento de DOM crea un objeto árbol que es administrado por el *parser* MSXML, esto trae como ventaja que los programadores o usuarios dejan la administración del contenido de un documento a la lógica interna de MSXML en vez de tener que crear la propia.

Metodología de Conversión de un Documento en un Árbol de Nodos

Cuando el *parser* MSXML carga un documento utilizando DOM, lee el mismo desde el principio hasta el final creando un modelo lógico de nodos de la estructura y del contenido dentro del documento XML. El documento en si mismo es considerado un nodo que contiene todos los demás nodos, incluyendo el nodo raíz, elementos, atributos y texto.

Una vez *parseado* el documento los dos nodos superiores de la estructura que representan este documento tendrían esta estructura:

```
Document (DOMDocument )
    XML Declaration (XMLDOMProcessingInstruction )
    xml-stylesheet processing instruction ( XMLDOMProcessingInstruction )
    DOCTYPE declaration (XMLDOMDocumentType )
    catalog (XMLDOMElement )
```

El nodo superior es el documento completo, que contiene a todos los demás nodos. Inmediatamente abajo están los nodos representando la declaración XML, el procesador de instrucciones de hojas de estilos, la declaración de la definición de documentos (DTD), y finalmente el nodo raíz para este documento, en este ejemplo **catalog**.

El nodo **catalog**, tiene el contenido real del documento.

SAX Parsers

SAX significa Simple API for XML. Un parser SAX tiene una interfaz del estilo:

SaxParse(documento, f_inicio_elemento, f_fin_elemento, f_texto).

En este modelo los principales argumentos son punteros a funciones. Estas funciones serán ejecutadas por el *parser* cuando él se encuentre con un elemento inicial, con uno final, o con texto.

Por ejemplo, en el documento Hola mundo , si ejecuto SaxParse(documento, fstart, fend, ftext), se produce la siguiente secuencia de invocaciones:

fstart(http://www.w3.org/TR/REC-xml-names/"p")

ftext("Hola")

fstart("b")

ftext("mundo")

fend("b")

fend("p")

SAX2 (*API* simple para XML) es un parser tipo "push-model". Cuando SAX2 *parsea* un documento, el SAXXMLReader (reader) lee el documento y pasa una

serie de eventos al manejador de eventos implementados. El reader de SAX2 genera una gran cantidad de eventos, entre los cuales se incluyen:

- **Los que ocurren en el contenido del documento**
- **Los que ocurren en la definición de tipo de documento (DTD)**
- **Los que ocurren como errores.**

Para manejar tales eventos, se debe implementar la clase del manejador correspondiente, que contiene métodos para procesar dichos eventos. Solo se deben implementar manejadores de eventos de aquellos eventos que uno desee procesar. Si no implementa un controlador para un evento, el *Reader* simplemente ignora dicho evento.

Los siguientes son los componentes que en la mayoría de los casos se deberían implementar en cualquier aplicación que maneje archivos XML con SAX2:

Componente	Descripción
ContentHandler	Implementa la interfaz **IVBSAXContentHandler**. Este componente es la clase que provee los métodos para procesar el contenido principal de un documento XML. Cuando SAX2 parsea un documento, el Reader pasa una serie de eventos al componente ContentHandler. Por ejemplo, por cada elemento en un documento, el Reader, pasa los eventos StartElement, Character, y EndElement. Para manejar estos eventos, se debe agregar código a los métodos en la clase ContentHandler para procesar la información pasada por el Reader.
ErrorHandler	Implementa la interfaz IVBSAXErrorHandler. El ErrorHandler es una clase que implementa métodos para procesar los eventos generados por errores pasados por el Reader.
DTDHandler	Implementa la interfaz IVBSAXDTDHandler. Esta clase implementa metodos para eventos relacionados con las DTD.

Comparación entre Parseadores DOM y SAX

La *API* simple para XML, conocida como SAX, es una interfaz que nos permite escribir aplicaciones que lean datos en un documento XML. SAX2 es la última versión de esta API (Interfaz de Programación de Aplicaciones).

La implementación de SAX2, que provee interfaces para Visual Basic y C++, nos ofrece una alternativa simple, rápida y de poca sobrecarga para procesar información a través del Modelo de objeto Documento (DOM). Cuando utilizamos DOM para manipular un archivo XML, el DOM lee el archivo, lo separa en objetos individuales (como elementos, atributos y comentarios), y entonces crea una estructura de árbol del documento en memoria. Los beneficios de utilizar DOM es que podemos hacer referencia y manipular cada objeto, llamados nodos, individualmente. Sin embargo, crear una estructura de árbol, sobre todo de un documento grande, requiere mucha cantidad de memoria.

A diferencia de DOM, SAX2, es basado en eventos, lo que significa que genera eventos cada vez que encuentra símbolos específicos en un documento XML. Una ventaja de SAX2 es que lee una sección del documento, genera un evento, y se mueve a la siguiente sección. A causa de que SAX2 procesa los documentos de forma serial, utiliza menos memoria que el DOM, y es mejor para

procesar grandes documentos. SAX2 puede crear aplicaciones que aborten el procesamiento cuando encuentran una pieza de información en particular.

A continuación se mencionan las alternativas de cuando conviene utilizar SAX2:

- **Documentos Grandes:** Quizás la mayor ventaja de SAX es que requiera mucha menos memoria al procesar un documento que DOM. Con SAX el consumo de memoria no crece junto con el tamaño del archivo a procesar (con DOM un documento de 100 KB requiere aproximadamente 1 MB de memoria para ser procesado), por lo que para grandes documentos la mejor alternativa es SAX, particularmente si no se tiene que cambiar el contenido del archivo.

- **Necesidad de Abortar el Procesamiento en Cualquier Momento:** SAX le permite abortar el procesamiento en cualquier momento, se pueden crear aplicaciones que busquen cierta información específica dentro del documento, la regresen y aborten el procesamiento.

- **Cuando desea obtener pocas cantidades de información:** Para una gran cantidad de soluciones basadas en XML, no es necesario leer el documento entero para obtener los resultados deseados. Por ejemplo con SAX podría buscar un documento o apellido específico en un archivo del Registro Civil y Capacidad de las Personas para luego

pasar los resultados a un nuevo servicio, ahorrando recursos al leer una porción especifica del archivo entero.

- **Cuando se Desea Crear una Nueva Estructura de Documento:** En alguno casos, se necesitaría utilizar SAX para crear una estructura de datos utilizando solo objetos de alto nivel como por ejemplo "Actas de Matrimonio" y "Defunciones", y luego combinar los datos de este archivo XML con otras fuentes de datos. En lugar de construir la estructura con DOM utilizando objetos de bajo nivel como elementos, atributos y procesando instrucciones, SAX permite construir la estructura del documento más eficiente y rápidamente.

- **Cuando los Recursos (de hardware) no Alcanzan:** Para documentos grandes y para grandes cantidades de documentos, SAX provee métodos más eficientes para *parsear* datos XML. Por ejemplo considere que un procedimiento regresa 10 MB de datos al servidor para ser pasados a un cliente. Utilizando SAX los datos pueden ser procesados usando un pequeño buffer de entrada, un pequeño buffer de trabajo y finalmente un pequeño buffer de salida, utilizando DOM, la estructura de datos es construida en memoria, requiriendo 10 MB de buffer de trabajo y al menos 10 MB de buffer de salida par los datos formateados a XML.

Las siguientes aclaraciones son las limitaciones actuales de la tecnología SAX:

- **No Existe el Acceso Aleatorio a los Documentos:** Por la razón que el documento no esta en memoria, se debe manipular la información en el orden en que es procesada, como resultado SAX es muy difícil de utilizar cuando en documento contiene muchas referencias cruzadas internas (como por ejemplo los atributos ID e IDREF).

- **Dificultad para Implementar Búsquedas Complejas:** Es responsabilidad del programador el mantener la información de contexto dentro de la estructura de datos, como por ejemplo el atributo del "padre" (nodo superior) del elemento actual.

- **No hay Implementación de SAX en los Navegadores Actuales:** El soporte para SAX no esta implementado dentro del Internet Explorer de Microsoft.

A continuación se mencionan los escenarios adecuados para utilizar DOM para *parsear* un documento:

- **Cuando se necesita realizar una transformación XSL:** El DOM trabaja mejor para las transformaciones XSL (XSLT), donde el documento es transformado basado en la plantilla XSLT aplicada. Por ejemplo para crear distintas vistas de una misma información, se debe transformar aplicando una o dos plantillas de estilos, para llevar a cabo dicha transformación se deben crear dos instancias del objeto DOM, una para almacenar el archivo de origen XML, y la otra para almacenar el contenido transformado.

- **Cuando se necesitan filtrados XPath complejos:** Si se necesita realizar filtrados complejos de XPath y retener en una compleja estructura de datos la información de contexto, al utilizar DOM la información de contexto es mantenida automáticamente en la estructura de árbol generada, mientras que utilizando SAX la información de contexto debe ser retenida o mantenida por el programador manualmente.

- **Cuando se necesita modificar y guardar archivos XML:** DOM permite crear o modificar un documento en memoria, tanto como leer un documento desde un archivo XML. SAX fue diseñado para lectura, no para escritura de documentos XML. DOM es la mejor opción para

modificar un archivo XML y guardar los cambios realizados en memoria.

- **Cuando se necesita acceso aleatorio a los datos:** Si el acceso aleatorio a la información es algo crucial, la mejor opción es utilizar DOM para crear la estructura de árbol en memoria.

Lenguaje Extensible de Consultas (XQL)

Es un lenguaje de consultas a sistemas de bases de datos, SQL incrustado en los documentos XML. Va incrustado dentro de los documentos XML, y sirve para hacer consultas contra bases de datos y obtener el resultado en un nuevo documento XML. Se pueden añadir y borrar items. No hay un estándar pero ya se han realizado varios desarrollos que podemos ver en el enlace indicado.

Lenguaje de Enlaces Extensible (XLL: XLINK\XPOINTER)

El lenguaje de enlaces extendidos presenta tres partes

XLINK es un enlace bidireccional que nos permite navegar entre páginas. Un enlace puede tener múltiples enlaces destino. Estos enlaces nos llevan al documento completo.

XPOINTER son enlaces bidireccionales que nos llevan no al documento completo, sino a una parte concreta dentro de éste.

XPATH

Es una especificación que nos indica como navegar por los nodos que forman el árbol de un documento XML.

XML y Bases de Datos

Antes de comenzar por hablar de XML y las bases de datos, es necesario contestar una pregunta muy frecuente: **¿es XML una base de datos?**

La respuesta es **si** en un sentido estricto de la definición del término: es una colección de datos, como posee un formato tipo base de datos, tiene ciertas ventajas, como por ejemplo: es auto descriptiva (los *tags* describen la estructura y el tipo de datos), es portable (utiliza unicode) y puede describir la información almacenada como un grafico de árbol o de estructura. Como principal desventaja es que el acceso a la información es lento debido al proceso de *parseo*.

También se debería preguntar si las tecnologías que acompañan a XML se pueden comparar con las de los sistemas de bases de datos:

- **Almacenamiento: los documentos XML en si.**
- **Estructuras: DTD, XML Schemas, etc.**
- **Lenguajes de Consultas: XQuery, XPath, XQL, XML-QL, etc.**
- **Interfaces de Programación: SAX, DOM, JDOM**

Por otro lado tiene ciertos aspectos que se encuentran en las bases de datos reales y no están presentes en XML:

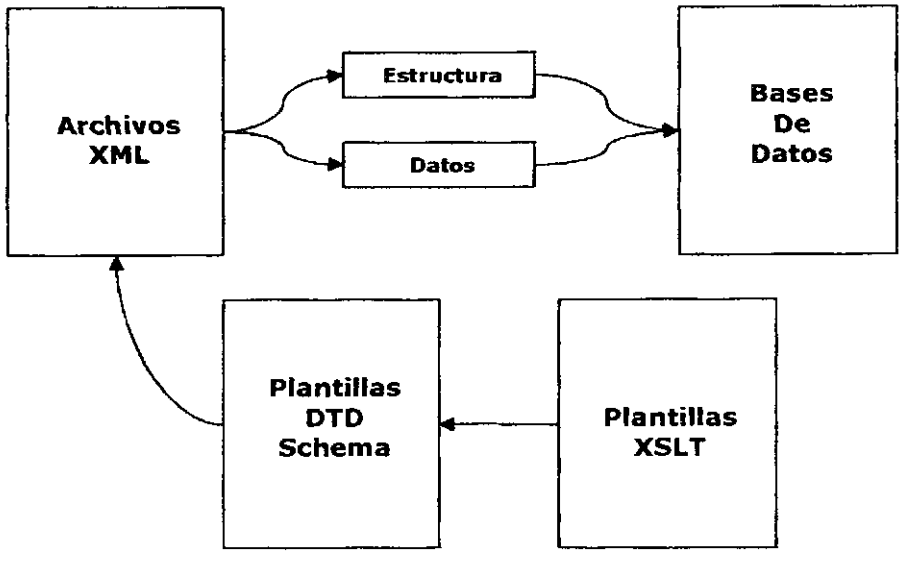
- **Almacenamiento eficiente.**
- **Índices.**
- **Seguridad.**
- **Transacciones.**
- **Control de integridad.**
- **Control de accesos concurrentes.**
- **Desencadenantes.**
- **Consultas a múltiples documentos.**

Por consiguiente, nosotros en este proyecto no recomendamos el utilizar XML como un sistema de base de datos, si no el utilizar las ventajas de XML al momento de transportar la información entre orígenes de datos heterogéneos.

Almacenando y Recuperando la Información

Al momento de transferir información entre los documentos xml y las bases de datos, es necesario mapear la estructura del documento (DTD, Schema, etc.) a la estructura de la base de datos.

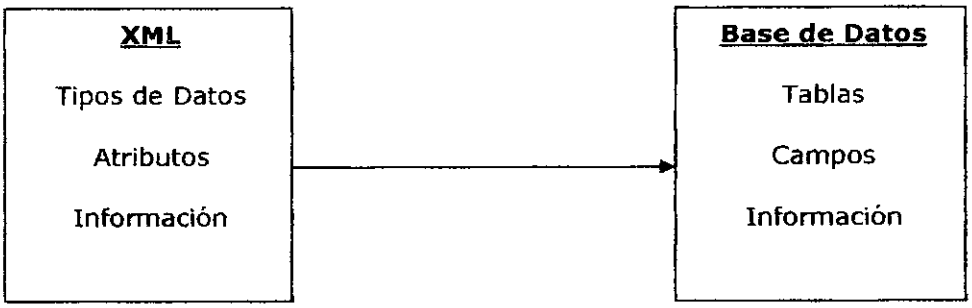
En el caso ideal la estructura del documento coincidirá con la de la base de datos, por lo general estos casos son los menos frecuentes, por lo que en muchos casos, la manera de realizar la transferencia es utilizar una plantilla XSLT, para llevar la estructura del documento a la estructura requerida por el destino (la base de datos) y luego se realiza la transferencia.



De forma similar al transferir información de una base de datos a un documento y luego se utiliza el documento en una aplicación, la estructura del documento será transformada a la forma necesaria para ser utilizada por la aplicación.

Mapeo de Documentos a Bases de Datos

El mapeo de un documento a una base de datos se realiza sobre los tipos de elementos, los atributos y la información (o texto).



Por lo general se omite cierta estructura física (como entidades, CDATA, e información de codificación) y ciertas estructuras lógicas (como instrucciones de procesamiento y comentarios), esto suena razonable si pensamos que las bases de datos y las aplicaciones en si, solo se interesan en la información almacenada en los documentos XML.

Son dos los tipos de mapeos más utilizados actualmente:

- **Mapeo basado en tablas.**
- **Mapeo Objeto – Relacional.**

Mapeo Basado en Tablas

Es el tipo de mapeo más utilizados por programas comerciales que hacen de vínculos entre distintas aplicaciones y las bases de datos utilizadas.

En este tipo de mapeo los documentos XML tienen una estructura prácticamente estática que se debe respetar:

```
<Database>
        <Table>
                <Row>
                        <column1></Column1>
                        <column2></Column2>
                        <.....></.....>
                </Row>
                <Row>
                .....
                </Row>
        </Table>
        <Table>
        .....
        </Table>
<Database>
```

En los casos de tratarse de un documento que represente una base de datos con una sola tabla, los *tags* <table> y <Row> adicionales se omiten.

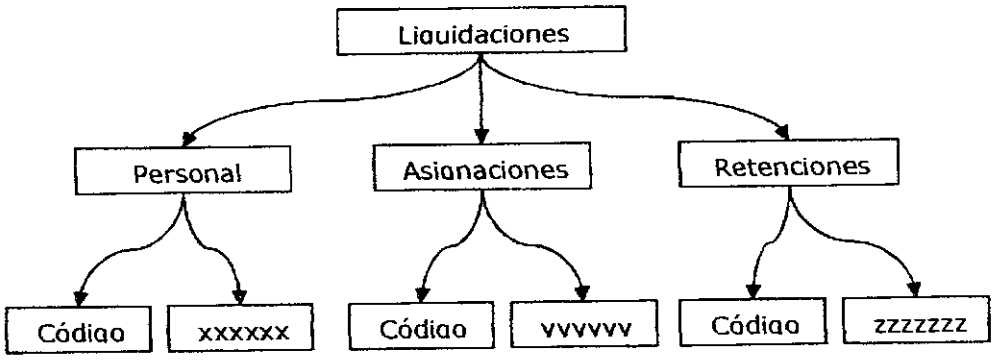
Este método de mapeo es especialmente útil cuando se trata de serialización de datos relacionales, como por ejemplo cuando se transfiere información entre dos bases de datos relacionales.

La principal desventaja es que no puede ser utilizada para cualquier documento XML que no respeten el formato arriba mencionado.

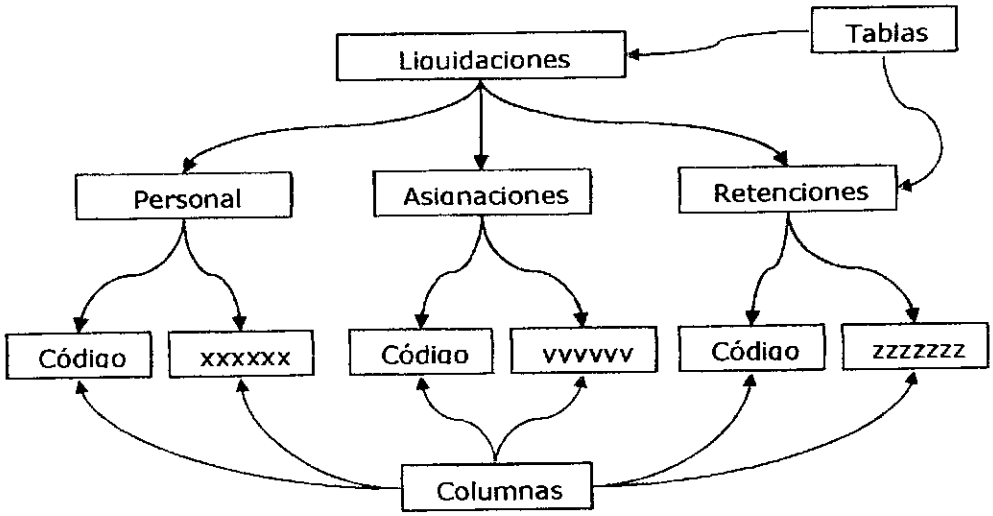
Mapeo Objeto – Relacional

Este proceso de mapeo modela los datos contenidos en el documento XML como un árbol de objetos que es específico a la información en el documento en si.

En este modelo los tipos de elementos con atributos, contenido o contenido mixto (elementos complejos) son generalmente modelados como clases. Los tipos de elementos con contenido únicamente tipo PCDATA son modelados como propiedades escalares.



Este modelo es luego mapeado a la base de datos relacional utilizando por ejemplo SQL, esto significa que las clases son mapeadas como tablas, las propiedades escalares son mapeadas como columnas en dichas tablas y los valores son mapeados como claves primarias o claves foráneas.

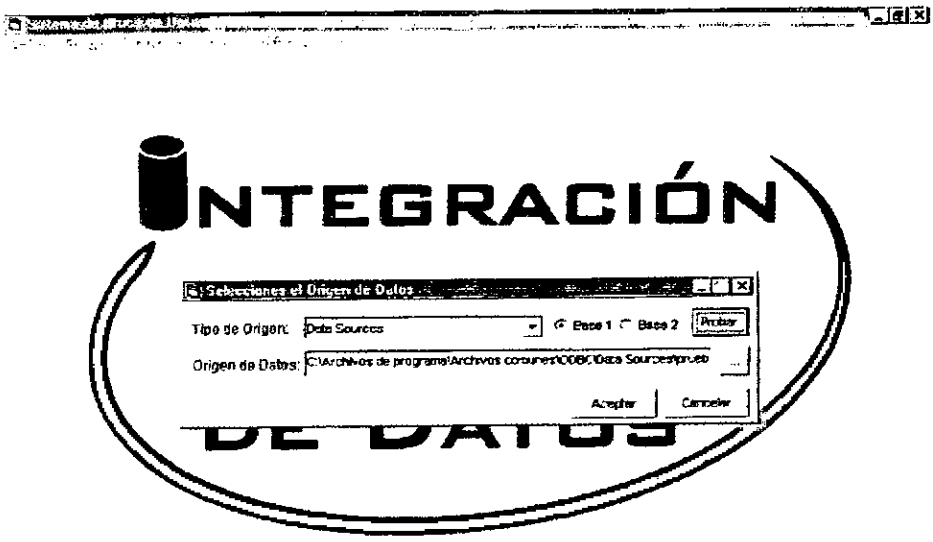

Tablas
Liquidaciones
Personal
Asignaciones
Retenciones
Código
xxxxxx
Código
vvvvvv
Código
zzzzzzz
Columnas

Desarrollo

Sistema General de Cruces

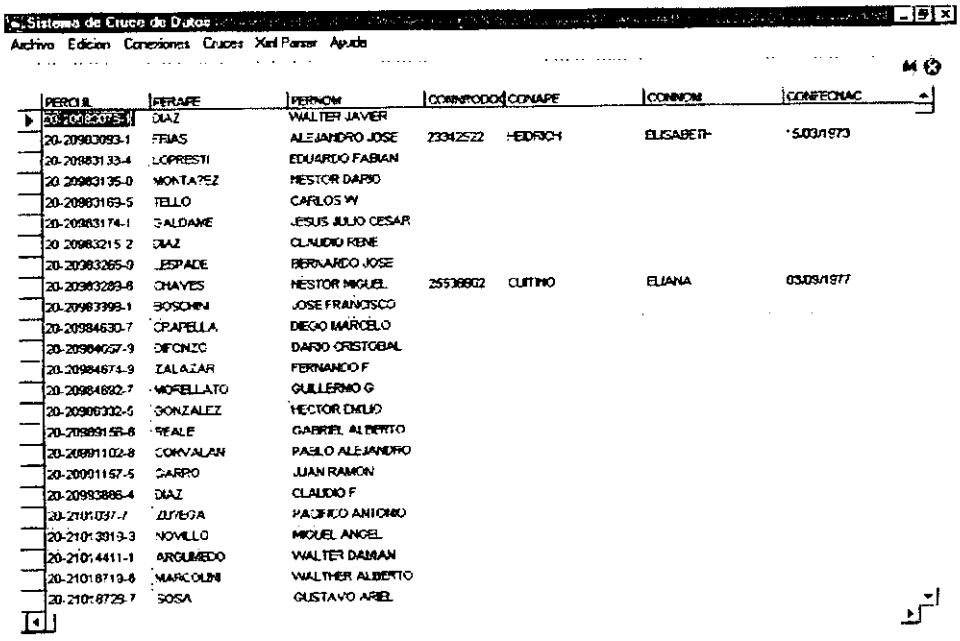
Para la etapa final del proyecto, intentamos conciliar la totalidad de los desarrollos realizados en el proyecto. Los nuevos módulos implementados contemplan una aplicación que permite seleccionar los orígenes de datos (DSN) ha utilizar para el cruce, la configuración previa (parámetros de accesos a los datos) y el cruce en si, y finalmente el conjunto de opciones para exportar los resultados del cruzamiento o cualquiera de los orígenes de datos a formato XML, contando con la posibilidad también de utilizar distintas metodologías relacionadas con este estándar para llevar a cabo dicha tarea.


Sistema de Cruce de Datos
Archivo Edicion Conexiones Cruces XmlParser Ayuda
INTEGRACIÓN
DE BASES
DE DATOS



Como se ve en la pantalla anterior, el primer paso consiste en establecer los distintos orígenes de datos para realizar los cruces. En este caso la selección de orígenes de datos se realiza a través de conexiones ODBC. Este paso se lleva a cabo en el menú conexiones y Nueva Conexión, luego se debe seleccionar un DSN (Data Source Name) que es quien tiene las características preestablecidas necesarias para conectarse con una base de datos de cualquier origen reconocido por ODBC (dbf –database file format-, mdb –archivo de Microsoft Access-, etc..). Si este es valido, le asigna la consulta que el usuario ingresó y la ejecuta, luego de verificar que la misma retornó al menos un registro. En caso contrario, informa que no se podrán realizar cruces si uno de los dos *recordset* está vacío. Posteriormente se debe realizar la misma tarea para el otro *recordset,* una vez finalizada esta etapa, al presionar el botón aceptar el sistema se

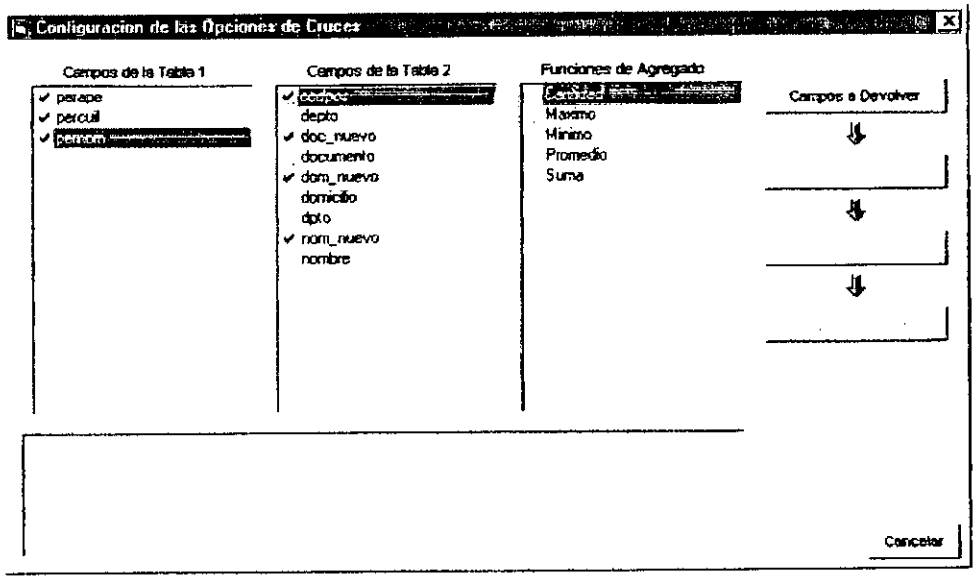
solicitará el ingreso de un valor numérico para determinar si actualiza el primer *recordset*, el segundo *recordset*, ambos o ninguno y regresar a la pantalla anterior, todo esto es necesario para el caso que necesite modificar uno de los dos orígenes de datos que ya fueron cargados, para no tener que repetir siempre dos veces la misma tarea cuando realmente no es necesario.



Una vez configurados los orígenes de datos se pueden visualizar los datos seleccionados en cualquiera de los dos Recordset.

Todos estos procedimientos se realizan mediante el menú **Conexiones**, **Ver Datos 1** y **Ver Datos 2**, lo que producirá como se ve en la pantalla anterior la

aparición de una grilla con las columnas y los registros seleccionados en las consultas introducidas en los pasos anteriores

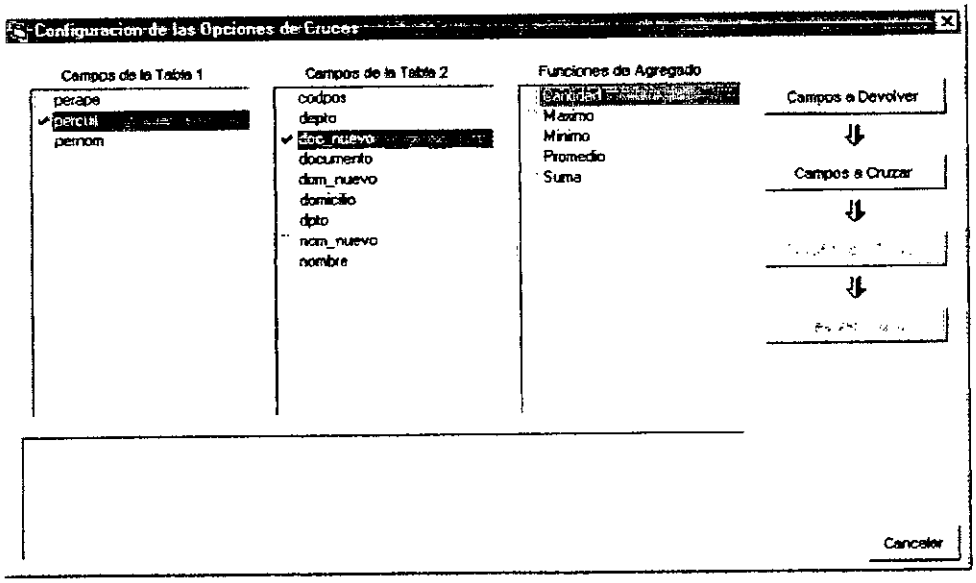


Una vez seleccionados los orígenes de datos, se puede seleccionar la opción de generar los cruces. Este proceso consiste en seleccionar como primer paso los campos que serán devueltos por la consulta final; esto se realiza marcando las casillas de verificación que figuran a la izquierda de cada nombre de campo, los cuales están separados en dos listas, representando cada una de las tablas seleccionadas en el paso anterior. Una vez seleccionados se debe presionar el botón marcado como **Campos a Devolver**.

Completado el paso anterior se habilita la segunda etapa, que consiste en seleccionar los campos por los cuales se desea realizar el cruce. Solo se puede

seleccionar un campo por tabla (el cruce es directo **campo1 = campo2**, la operación de comparación puede variar). Si ya se ha seleccionado un campo previamente y posteriormente se selecciona otro de la misma lista se desmarcará automáticamente el primero. Se debe tener en cuenta las variaciones en cuanto a tipos de datos dentro de los campos que se seleccionarán para el cruce; debido a que cabe la posibilidad de incompatibilidades entre ellos (tipos de datos, longitud, etc.) por lo que se deberán aplicar las funciones de transferencia mencionadas en los informes anteriores para llevar uno de los campos al formato necesario para poder ser cruzado con el otro campo seleccionado.

Finalizada la selección de los campos a cruzar, se debe presionar el botón **Campos a Cruzar**.

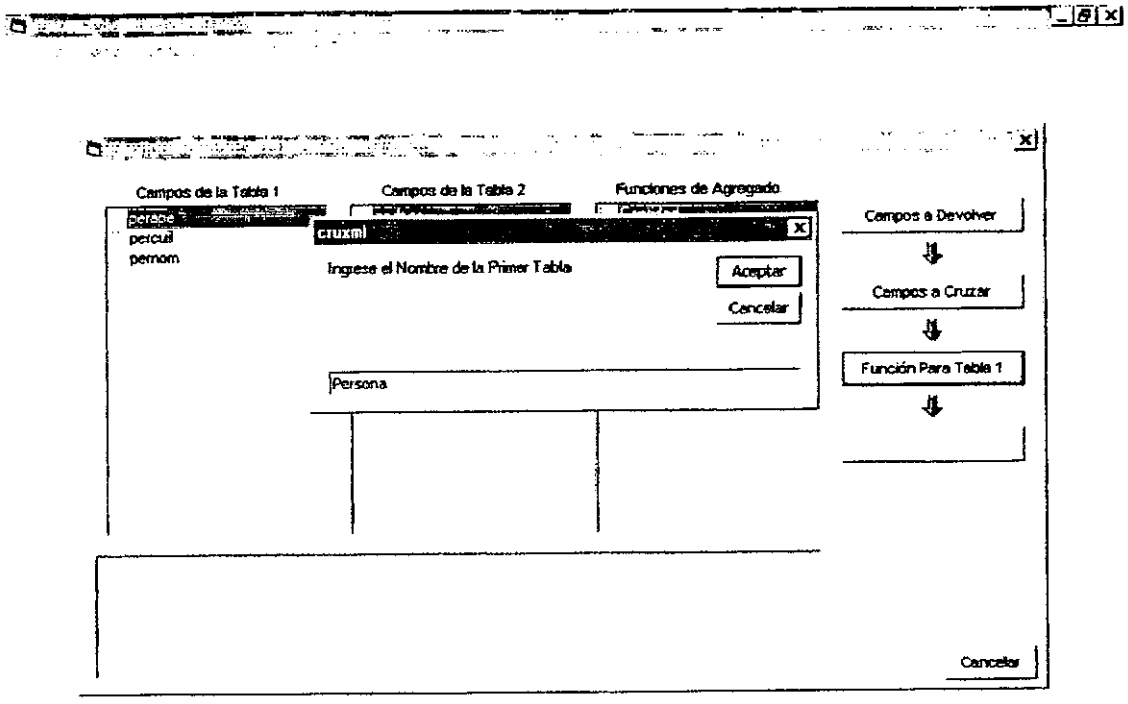


El tercer paso consiste en seleccionar las **funciones de agregado** que se pueden utilizar con los campos de la primera tabla (primer origen de datos). Estas funciones son útiles para realizar operaciones varias, como cálculos, sumatorias, promedios, etc. Algunas de las funciones disponibles son:

- **Suma**: devuelve la suma de todos los registros de una columna numérica
- **Cantidad**: devuelve la cantidad de registros retornados por la consulta.
- **Promedio**: devuelve el promedio de una columna numérica.
- **Máximo**: devuelve el máximo valor presente en una columna numérica.
- **Mínimo**: devuelve el mínimo valor presente en una columna numérica.

Se debe aclarar que si se incluyen campos adicionales a los campos que se les aplique alguna función antes mencionada, estos campos adicionales deberán estar incluidos en una cláusula Group By al final de la sentencia SQL generada.

También en este paso se le solicitará al usuario que introduzca el nombre de las dos tablas; esto es requerido para poder distinguir entre campos con el mismo nombre en ambas tablas (esto es necesario porque en caso de que existieran dos o más campos con el mismo nombre y sin ser distinguidos por el nombre de la tabla a la cual pertenecen, el motor de la base de datos -quien interpreta la consulta-, generara un error en el que explica que "**existen campos con el mismo nombre y se deberán calificar**").



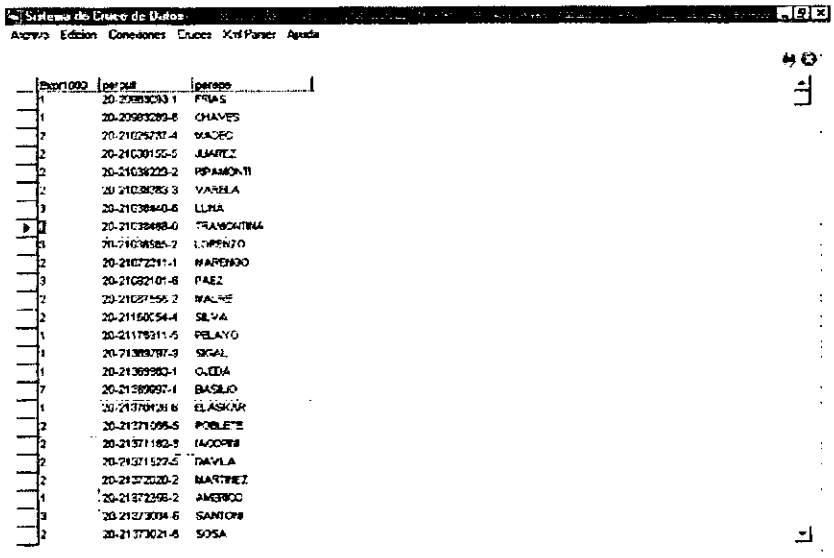
Para finalizar este paso se debe presionar el botón **Funciones para la Tabla 1**.

Por último se habilita la opción de **Realizar Cruce**, previo a realizar el mismo se debe verificar la consulta que se va ejecutar, la cual aparece en el panel inferior de la pantalla mostrada anteriormente.

Esta consulta SQL es la generada por el sistema en base a todos los pasos anteriores, la misma puede ser modificada por cualquier usuario con conocimientos básicos del lenguaje de consulta SQL.

Una vez realizada la consulta, en caso de no haber ocurrido ningún error con el motor de la base de datos, finalizamos la ejecución con el botón **Cerrar**.

Con la opción **Ver Resultados** del menú **Cruces**, podemos desplegar una grilla con los resultados del último cruce



Con los iconos ubicados en la esquina superior derecha de la grilla, podemos consultar la cantidad de registros retornados por la consulta ejecutada, y con el último cerramos la grilla (no borramos los resultados, entrando nuevamente al menú **Ver Resultados**, los visualizamos nuevamente). Estos iconos están presentes también en las grillas de los orígenes de datos cargados al principio de del programa.

En la pantalla anterior se puede visualizar un cruce realizado entre bases de la Dirección General de Escuelas, para obtener la cantidad de hijos que tiene registrado en las bases de la repartición pública el personal de la misma.

Como se menciono en el informe anterior los campos por los cuales se realizan los cruces (solo se puede seleccionar uno por tabla), no necesariamente pueden coincidir en el formato, tanto de tipo de datos, como en el de presentación, por lo que es el lugar adecuado para aplicar las funciones con las cuales forzamos a uno de los dos campos a cumplir con los requerimientos del otro para llevar a cabo el cruce: las llamadas **funciones de transferencia**.

Como ejemplo de cruzamiento podemos mencionar el caso de la DGE con las bases del Registro del Estado Civil y Capacidad de las Personas.

El primero guarda en sus tablas los documentos de identidad de sus empleados como numero de C.U.I.T. o C.U.I.L., el mismo tiene formato de texto y su presentación respeta el siguiente formato: "**xx-xxxxxxxx-x**", mientras que el segundo también tiene formato texto, pero su presentación es la de un número de documento normal, sin puntos ni comas, con una longitud máxima de 12 caracteres.

Por tales motivos es necesario llevar el formato DNI a formato C.U.I.L. o viceversa. Para tal acción tenemos que agregar funciones de transferencia que trabajen sobre los guiones de los DNI y completen con comodines los 2 dígitos al

principio y el dígito final del número de C.U.I.L. o bien, en este caso más sencillo, la ejecución de la función que retira los dígitos antes mencionados al igual que los guiones del número de C.U.I.L.

Para luego poder realizar el cruce con los datos "**modificados**", se debe crear en una base de datos temporal al menos una tabla nueva para guardar temporalmente los datos cambiados para poder realizar el cruce.

Para poder realizar este paso, se utiliza código que genera dichas tablas temporales, el código que realiza dicha tarea se encuentra desarrollado en el lenguaje *Visual Basic*. El código en si genera dos tablas llamadas **tabla1** y **tabla2**, y los campos que las componen son los mismos campos de las tablas originales, pero los mismos están en formato texto para evitar cualquier conflicto a la hora de realizar los cruces.

Una vez finalizado los cruces que sean necesarios, el sistema nos da la posibilidad de exportar el contenido de la tabla resultante a formato XML.

Tenemos 3 posibilidades para elegir al momento de realizar la exportación:

1. XML Manual
2. Metodología DOM.
3. Recordset ADO.

La primera opción es la codificación manual de un archivo XML, la cual se lleva a cabo por el código desarrollado en *Visual Basic.* Las principales acciones que realiza esta aplicación es: en primer lugar tomar los nombres de los campos en el origen de datos a transformar y los convertirlos en los "Tags" del archivo final; a continuación realizar un bucle o ciclo por todos y cada uno de los registros del origen de datos agregándolos a un archivo, hasta ese momento de texto sin formato, formado las cadenas <campo1>valor registro</campo1>... hasta finalizar el archivo; finalmente cerrar el *tag* principal, el cual indica que se trata de un archivo XML y lo graba al disco con un nombre previamente seleccionado por el usuario.

Este proceso en forma aislada genera un Archivo XML No Valido. Para validar el mismo se le debe adjuntar un DTD, el cual le indica al explorador o cualquier otro visor valido de archivos XML, el orden y la cardinalidad que posee cada *tag* en el archivo.

El sistema genera un DTD al momento de crear el archivo XML de salida. Existe en la actualidad dos posibilidades con los DTD: una es de crear un archivo externo y adjuntar en el archivo XML una referencia al otro, y la segunda, la cual es la utilizada por nuestro sistema, la de incluir en la cabecera del archivo XML la definición completa del esquema DTD que validara nuestro archivo.

La metodología DOM es una librería de codificación implementada por Microsoft, que permite representar en memoria y luego "Bajar" a un archivo un esquema XML.

El mayor problema que presenta es que si la estructura a representar (el resultado de nuestros cruces), es muy grande (en cantidad de registros), requerirá una gran cantidad de recursos del sistema en el que está corriendo, pudiéndose dar el caso de que se suspendiera la ejecución debido a la falta de memoria libre u ocasionando la caída del sistema completo.

La tercer opción es la implementada por el lenguaje de programación *Visual Basic*, la que nos permite guardar el contenido de un *recordset* (en el cual tenemos guardado de forma temporal el resultado de nuestros cruces, o la información original de cada tabla) en un archivo en el disco rígido u otra ubicación que seleccione el usuario.

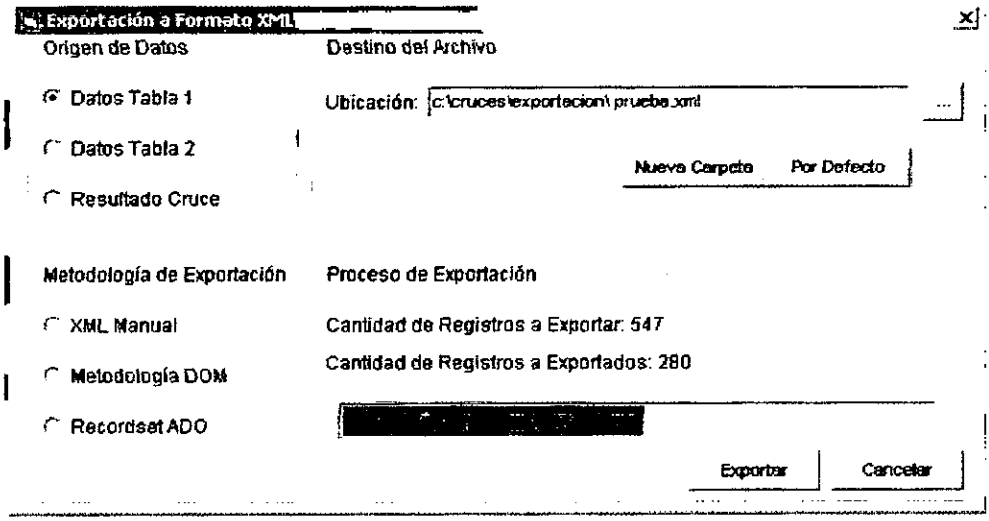
Esta opción es la más rápida de implementar e incluye en el mismo archivo la descripción de los campos que guardamos.

Como esta herramienta esta pensada para guardar en forma momentánea el *recordset* en el disco para luego volver a cargarlo, sería la forma ideal de transportar la información, para luego volver a cargarla en otra máquina con el procedimiento inverso, que también se incluye con el objeto *recordset*.

Para la utilización en si de los archivos XML generados, se les debe aplicar alguna plantilla de estilos al momento de visualizar los datos en el explorador de Internet, ya que por defecto tiene la visualización secuencial en la forma en que fueron grabados los datos, solamente individualizados por los **tag** creados.

A continuación se explica el proceso de exportación en el sistema de cruces:

En la imagen inferior se observa la pantalla a la que se accede luego de presionar el menú **XmlParser** en la barra de menú del sistema.



En la parte superior izquierda, el usuario selecciona que es lo que desea exportar. Las opciones son los datos originales de la **tabla 1**, los datos originales de la **tabla 2** o los datos devueltos por el proceso de cruces. Esta última opción solo esta habilitada si se realizó un cruce antes de ingresar a esta pantalla.

En la parte inferior izquierda están las opciones de exportación antes explicadas, las cuales incluyen la exportación manual, la metodología DOM y guardar el *recordset* ADO.

Una vez realizada esta acción, se deberá seleccionar el destino de los datos para el archivo a crear, podemos escribir directamente el camino del archivo en el disco rígido más el nombre del archivo en la caja de texto si estamos seguro de que existe dicho camino, o bien podemos buscar uno con el botón con los tres puntos ubicado a la derecha de la caja de texto, lo que nos presentará una ventana de **Guardar Como...**. Otra opción es el botón **Nueva Carpeta** que nos permite crear una nueva ubicación en la cual luego guardaremos el archivo creado.

Una vez que tengamos decidido donde guardaremos nuestro archivo, podemos presionar el botón **Por Defecto**, lo que ocasionara que la próxima vez que tengamos que exportar algún archivo simplemente tendremos que ingresar el nombre del nuevo archivo, pero la ubicación siempre será la misma. La ubicación por defecto puede ser cambiada las veces que sea necesario, es solo como una herramienta para ahorrar tiempo si siempre se van a guardar los datos resultantes en el mismo lugar (carpeta del disco rígido).

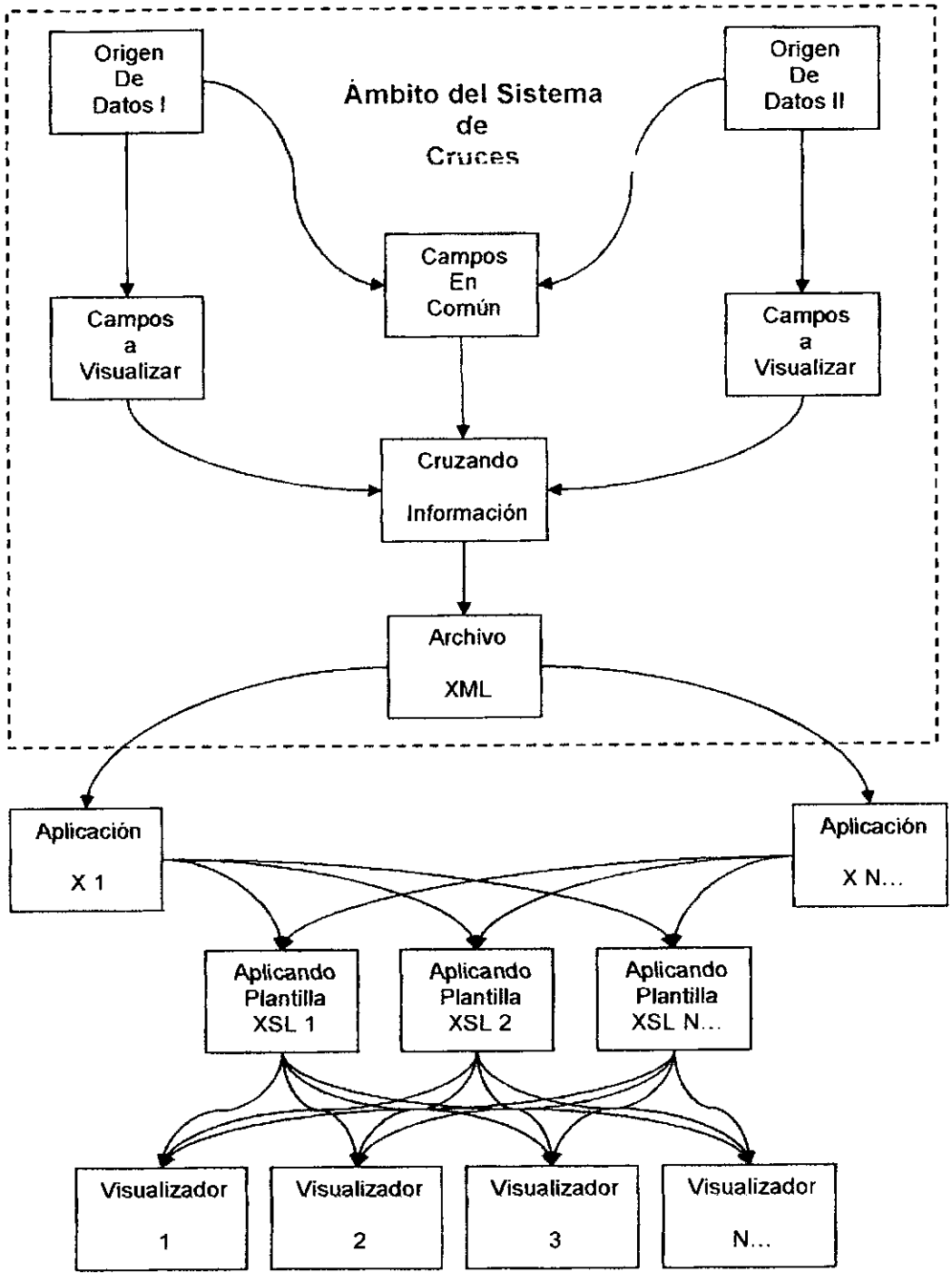
Finalmente, cuando se presiona el botón **Exportar**, aparece el panel derecho inferior, el cual nos informa cuantos registros vamos a exportar, cuantos llevamos exportados y una barra de progreso gráfica.

Este sistema es de gran utilidad al momento de generar cruces de grandes cantidades de información, como es el caso ya mencionado de la Junta Electoral (la cual tiene más de 1.070.000 de registros) y las bases del Registro Civil y Capacidad de las Personas, por ejemplo.

En tales casos el sistema podría realizar un cruce por el número de documento y detectar por ejemplo personas fallecidas que aun permanezcan en el padrón, para una depuración del mismo.

Una vez detectado los problemas se podrá fácilmente exportar la información realmente necesaria (las personas que realmente deben estar en el padrón) a formato XML para transportar la información a través de la red en un formato que sea liviano para la transmisión (que ocupe poco ancho de banda) y que pueda ser interpretado e implementado en cualquier sistema que requiera dicha información, como podría ser la publicación en una pagina *WEB* del padrón para consulta de los votantes o para una auditoria más exhaustiva por parte de otras oficinas.

El siguiente es el esquema de trabajo con el sistema:


Origen De Datos I
Origen De Datos II
Ámbito del Sistema de Cruces
Campos En Común
Campos a Visualizar
Campos a Visualizar
Cruzando Información
Archivo XML
Aplicación X 1
Aplicación X N...
Aplicando Plantilla XSL 1
Aplicando Plantilla XSL 2
Aplicando Plantilla XSL N...
Visualizador 1
Visualizador 2
Visualizador 3
Visualizador N...

Lo que esta fuera del ámbito del sistema de cruces se refiere al manejo que le de cada sistema (cada dependencia que maneje los datos entregados por los cruces) estas dependencias de acuerdo a sus necesidades aplicaran plantillas de estilos a los archivos XML para luego visualizarlos o publicarlos en algún navegador o en sus propios sistemas, o los procesaran con sus propios métodos, lo que se debe dejar en claro que el sistema de cruce luego de realizar las tareas de cruzamiento genera un archivo que es universalmente entendido por los sistemas y al que se le puede procesar de distintas maneras para obtener distintos resultados, con la ventaja de la portabilidad y de la eficiencia al momento de la transmisión de un sistema a otro.

Anexos

Glosario

ADO Activex Data Object. Tecnología de acceso a datos creada por Microsoft.

Ancho de Banda Es una magnitud que sirve para medir la capacidad de transmitir información de un sistema informático. Se mide en kbps.

API **Aplication Programming Interface**. Librerías y rutinas de programación. Existen APIs para distintos temas, como por ejemplo TAPI que es la API para programar aplicaciones relacionadas con telefonía, MAPI, similar pero referida a la programación de mail.

Attachment Son archivos que se adjuntan a un mensaje de correo electrónico.

CDATA Es una sección dentro de un archivos XML que se utiliza para introducir bloques de texto, en los cuales existen caracteres que de estar fuera de estas secciones serian interpretados como markup o tags.

Cardinalidad Magnitud utilizada para especificar la cantidad de relaciones que posee un registro en un entorno dado.

Cruce Es la acción de obtener resultados en base a la unión de

	dos orígenes de datos por un campo en común.
Datawarehousing	Es un conjunto de técnicas, por las cuales se almacena la información en repositorios tridimensionales, los cuales son óptimos para la obtención de datos resumidos, estadísticas y para la optimización del almacenaje de la información.
DBF	Formato de archivo de base de datos utilizado por Dbase, Fox Pro, etc.
DOM	Modelo de Objeto Documento
DSN (DataSource Name)	Nombre de orígenes de datos. Son unos archivos en los cuales se almacenan todas las propiedades necesarias para conectarse con un origen de datos especifico
DTD	Definición de Tipos de Documentos: es un documento que define las normas que debe cumplir un archivo XML para ser valido. Puede estar dentro del mismo archivo XML o ser un documento separado.
MDB	Formato de archivo de bases de datos propietario de Microsoft Access.
HTML	Lenguajes de Marcado de Hipertexto. Es el lenguaje utilizado por las páginas WEB para presentar información de forma estática. Esta compuesto por tags o marcas preestablecidas.

Parser Librería que transforma un documento de texto a una representación interna, entendida por alguna aplicación.

Recordset Conjunto de datos en memoria. Son los objetos en los cuales se almacenan temporalmente la información recuperada tras la apertura de un origen de datos por medio de una consulta SQL o directamente trabajando sobre una tabla

Schemas Representa el esquema de un archivo XML, los schemas le indican a los programas la estructura que deben respetar los archivos al ser procesados.

SAX API simple de XML. Funciones para *parsear* documentos XML en la que se destaca le procesamiento secuencial del archivo y su gran utilidad en documento muy grandes.

SGML Lenguaje General de Marcado Estandarizado. Es un lenguaje creado por IBM en los años 70, del cual surgió el XML

SQL Lenguaje Estructurado de Consulta: Lenguaje desarrollado para manipular bases de datos, tanto la información contenida en ella, como la estructura de la misma. (Este lenguaje está destinado al usuario final con escasos conocimientos de informática y programación de bases de datos).

Tabla	Estructura lógica de almacenamiento de información. La información se almacena organizada por medio de columnas y cada fila representa un registro
Tags (XML)	Marcas. Son las etiquetas que forman los archivos XML, al contrario de los tags de HTML, estos son definidos por el usuario que crea el archivo.
XLINK	Es un enlace bidireccional que nos permite navegar entre páginas. Un enlace puede tener múltiples enlaces destino. Estos enlaces nos llevan al documento completo.
XML	Lenguaje de Marcas Extendido. Es un lenguaje que extiende las posibilidades del HTML, permitiendo el uso de tags propios, uno de las grandes ventajas es la universalidad de la información contenida en un archivo XML, la cual puede ser interpretada por cualquier sistema.
XPOINTER	Son enlaces bidireccionales que nos llevan no al documento completo, sino a una parte concreta dentro de éste.
XSLT	Es el lenguaje para transformar un archivo XML en otro mediante la aplicación de distintos filtros y hojas de estilos extensibles.

Código

A continuación se realizará una adaptación de las rutinas trabajadas al lenguaje Visual Basic, con el objeto de permitir una lectura clara.

- **Tablas Temporales:**

```
Private Sub Temporales_□omp()
On Error GoTo err_crear
Dim cadena1 As String
Dim cadena2 As String
Dim lista1 As String
Dim lista2 As String

conex.Mode = adModeReadWrite
conex.Open

lbl_pasos.Caption = "Eliminando Tablas..."
lbl_pasos.Visible = True

consulta.ActiveConnection = conex4
consulta.CommandType = adCmdText
consulta.CommandText = "Drop Table tabla1"
Set rs4 = consulta.Execute
consulta.CommandText = "Drop Table tabla2"
Set rs4 = consulta.Execute

lbl_pasos.Caption = "Creando Tablas..."

For I = 0 To List1.ListCount – 1
   If I < List1.ListCount – 1 Then
```

```
        cadena1 = cadena1 + Lcase(List1.List(i)) + " TEXT,"
        lista1 = lista1 + List1.List(i) + ","
    Else
        cadena1 = cadena1 + Lcase(List1.List(i)) + " TEXT"
        lista1 = lista1 + List1.List(i)
    End If
Next I

consulta.CommandText = "Create Table tabla1 (" + cadena1 + ")"
Set rs4 = consulta.Execute

For I = 0 To List2.ListCount - 1
    If I < List2.ListCount - 1 Then
        cadena2 = cadena2 + Lcase(List2.List(i)) + " TEXT,"
        lista2 = lista2 + List2.List(i) + ","
    Else
        cadena2 = cadena2 + Lcase(List2.List(i)) + " TEXT"
        lista2 = lista2 + List2.List(i)
    End If
Next I

consulta.CommandText = "Create Table tabla2 (" + cadena2 + ")"
Set rs4 = consulta.Execute
Set rs4 = Nothing
conex.Close

conex.Open
lbl_pasos.Caption = "Transfiriendo Datos..."

rs1.MoveFirst
For I = 0 To rs1.RecordCount - 1
    rs4.AddNew
```

```
    For j = 0 To rs1.Fields.Count – 1
      rs4(j) = CStr(rs1(j))
    Next j
    rs4.Update
    rs1.MoveNext
Next I

rs2.MoveFirst
For I = 0 To rs2.RecordCount – 1
    rs4.AddNew
    For j = 0 To rs2.Fields.Count – 1
      rs4(j) = CStr(rs2(j))
    Next j
    rs4.Update
    rs2.MoveNext
Next I

conex.Close
Set rs4 = Nothing
Set conex = Nothing

MsgBox "Estructuras de Tablas Creadas Exitosamente", vbInformation +
vbOKOnly, "□omproba de Tablas"
Exit Sub
err_crear:
        MsgBox "Ocurrio el Siguiente Error: " + Err.Description,
vbExclamation + vbOKOnly, "Error al Crear Tablas"
        If conex.State <> 0 Then
          conex.Close
        End If
        Exit Sub
End Sub
```

El código anterior toma los nombres de las tablas originales cargas en los □omprobac rs1 y rs2 y crea dos tablas con los mismos campos en formato texto, luego procede a cargar los registros contenidos en los dos □omprobac, transformando los campos en formato texto mediante la función **Cstr**.

- **Selección de □omproba de Datos**

```
Private Sub btn_Origenes_□omp()
On Error GoTo err_probar
Select Case cbo_tipo.ListIndex
    Case 0
      If opb1.Value = True Then
        If Len(Trim(cmd_ds.FileName)) > 0 Then
          b_□ompr.b1_tipo = 0
          b_□ompr.b1_origen = cmd_ds.FileName
          b_□ompr.b1_sql = InputBox("Ingrese una consulta sql valida",
"Origen de Datos")
          If b_temp.b1_sql = "" Then
            MsgBox "Debe Ingresar una Consulta Valida",
            Exit Sub
          Else
            If conex1.State <> 0 Then
              conex1.Close
            End If
            conex1.ConnectionString = "File Name=" + b_temp.b1_origen
            conex1.CursorLocation = adUseClient
            conex1.Open
            comando1.ActiveConnection = conex1
            comando1.CommandType = adCmdText
```

```
            comando1.CommandText = b_temp.b1_sql
            rs1.CursorType = adOpenDynamic
            Set rs1 = comando1.Execute
            MsgBox "Registros Devueltos por la Consulta: " +
CStr(rs1.RecordCount), vbInformation + vbOKOnly, "□omprobación de
Consulta"
            If rs1.RecordCount > 0 Then
              b_temp.b1_tienedatos = True
            Else
              MsgBox "La consulta no contiene datos, no se podran
realizar cruces.", vbInformation + vbOKOnly, "Consulta sin Datos"
              b_temp.b1_tienedatos = False
            End If
          End If
        Else
          MsgBox "Debe Seleccionar un Origen de Datos Valido",
vbExclamation + vbOKOnly, "Origen de Datos No Valido"
          Exit Sub
        End If
      Else
        If Len(Trim(cmd_ds.FileName)) > 0 Then
          b_□ompr.b2_tipo = 0
          b_□ompr.b2_origen = cmd_ds.FileName
          b_□ompr.b2_sql = InputBox("Ingrese una consulta sql valida",
"Origen de Datos")
          If b_temp.b2_sql = "" Then
            MsgBox "Debe Ingresar una Consulta Valida", vbInformation
+ vbOKOnly, "Falta Consulta SQL"
            Exit Sub
          Else
            If conex2.State <> 0 Then
              conex2.Close
```

```
                End If
                conex2.ConnectionString = "File Name=" + b_temp.b2_origen
                conex2.CursorLocation = adUseClient
                conex2.Open
                comando2.ActiveConnection = conex2
                comando2.CommandType = adCmdText
                comando2.CommandText = b_temp.b2_sql
                rs2.CursorType = adOpenDynamic
                Set rs2 = comando2.Execute
                MsgBox "Registros Devueltos por la Consulta: " +
CStr(rs2.RecordCount), vbInformation + vbOKOnly, "□omprobación de
Consulta"
                If rs2.RecordCount > 0 Then
                  b_temp.b2_tienedatos = True
                Else
                  MsgBox "La consulta no contiene datos, no se podran
realizar cruces.", vbInformation + vbOKOnly, "Consulta sin Datos"
                  b_temp.b2_tienedatos = False
                End If
              End If
            Else
              MsgBox "Debe Seleccionar un Origen de Datos Valido",
vbExclamation + vbOKOnly, "Origen de Datos No Valido"
              Exit Sub
            End If
          End If
      Case 1
End Select
btn_aceptar.Enabled = True
Exit Sub
err_probar:
        MsgBox "Ocurrio el Siguiente Error: " + Err.Description
```

```
        btn_aceptar.Enabled = False
        Exit Sub
End Sub
```

- **XML Manual**

```
Public Sub genera_xml()
 Dim fso  As New FileSystemObject
 Dim archivo
 Set archivo = fso.OpenTextFile(App.Path + "\xml_prueba.xml",
ForAppending, True)
 archivo.WriteLine "<variable>"
 Form1.Data1.Recordset.MoveFirst
 For i = 0 To Form1.Data1.Recordset.RecordCount - 1
        For j = 0 To UBound(campos) - 1
       If Not IsNull(Form1.Data1.Recordset(j)) Then
       archivo.WriteLine "<" + campos(j) + ">" +
LCase(Form1.Data1.Recordset(j)) + "</" + campos(j) + ">"
       Else
       n = "nulo"
       archivo.WriteLine "<" + campos(j) + ">" + n + "</" + campos(j)
+ ">"
       End If
    Next j
    Form1.Data1.Recordset.MoveNext
    Form1.lbl_r.Caption = "Registro " + Str(i) + "de " +
Str(Form1.Data1.Recordset.RecordCount - 1)
    DoEvents
 Next i
 archivo.WriteLine "</variable>"
 archivo.Close
 Set archivo = Nothing
 Set fso = Nothing
```

```
End Sub
```

- **Genera DTD**

```
Public Sub genera_dtd()
ReDim campos2(Form1.Data1.Recordset.Fields.Count)
ReDim camp_nul(Form1.Data1.Recordset.Fields.Count)
With Form1.Data1
   For i = 0 To .Recordset.Fields.Count - 1
      campos2(i) = LCase(.Recordset.Fields(i).Name)
      camp_nul(i) = .Recordset.Fields(i).Required
   Next i
End With
Dim fso As New FileSystemObject
Dim ar_dtd
Dim cad_dtd As String
Dim cad_dtd2 As String
Dim cad_dtd3 As String
Dim x As Integer
Dim def_dtd() As String
x = UBound(campos2)
ReDim def_dtd(x)
cad_dtd = "<!DOCTYPE variable ["
cad_dtd2 = Join(campos2, ",")
cad_dtd2 = "<!ELEMENT variable (" + Left(cad_dtd2, Len(cad_dtd2)
- 1) + ")>"
For i = 0 To UBound(campos2) - 1
   def_dtd(i) = "<!ELEMENT " + campos2(i) + " (#PCDATA)>"
Next i
cad_dtd3 = "]>"

Set ar_dtd = fso.CreateTextFile(App.Path + "\xml_prueba.xml",
True)
```

```
ar_dtd.WriteLine cad_xml
ar_dtd.WriteLine cad_dtd
ar_dtd.WriteLine cad_dtd2
For j = 0 To UBound(campos2) - 1
 ar_dtd.WriteLine def_dtd(j)
Next j
ar_dtd.WriteLine cad_dtd3
ar_dtd.Close
Call lista_campos
End Sub
```

- **XML Manual 2**

```
Public Enum XML_ERRS
   ERR_INVALID_OBJ = 30000
   ERR_NO_RECORDSET = 30010
   ERR_NOT_OPEN = 30020
   ERR_NOFILENAME = 30030
   ERR_FILENAME_INVALID = 30040
   ERR_INVALID_VALUE = 30050
End Enum

Private Const ERR_INVALID_OBJ_DESC = "se Requiere un
Recordset ADODB"
Private Const ERR_NO_RECORDSET_DESC = "Debe establecer
la propiedad CurrentRecordset a un recordset ADODB abierto y
conectado"
Private Const ERR_NOT_OPEN_DESC = "El recordset debe estar
abierto y conectado a un origen de datos"
Private Const ERR_NOFILENAME_DESC = "Debe especificar un
nombre de archivo especificando la propiedad OutputFile"
Private Const ERR_FILENAME_INVALID_DESC = "Nombre de
archivo no valido"
```

```
Private Const ERR_INVALID_VALUE_DESC = "El valor no puede
ser escrito al archivo XML"

Private psCurrentTag As String
Private poRs As ADODB.Recordset
Private psOutputFile As String
Private psOutputDirectory As String
Private psXML As String
Private pbCData As Boolean
Private psOutStandingTags() As String
Private pDictAttributes As Scripting.Dictionary
Private pbWriting As Boolean

Public Property Set CurrentRecordset(oRS As Object)
On Error Resume Next
If TypeOf oRS Is ADODB.Recordset Then
   Set poRs = oRS
Else
   Err.Raise ERR_INVALID_OBJ, , ERR_INVALID_OBJ_DESC
End If
If poRs.State <> adStateOpen Then poRs.Open
If poRs.State <> adStateOpen Then Err.Raise ERR_NOT_OPEN, ,
ERR_NOT_OPEN_DESC
End Property

Public Property Let OutputDirectory(ByVal NewValue As String)
   psOutputDirectory = NewValue
End Property

Public Property Let OutputFile(ByVal NewValue As String)
   psOutputFile = NewValue
End Property
```

```
Public Property Get OutputFile() As String
    If InStr(psOutputFile, "\") > 0 Or Len(psOutputDirectory) = 0
Then
        OutputFile = psOutputFile
    Else
        OutputFile = psOutputDirectory &
IIf(Right(psOutputDirectory, 1) = "\", _
          psOutputFile, "\" & psOutputFile)
    End If
End Property

Public Property Get OutputDirectory() As String
  OutputDirectory = psOutputDirectory
End Property

Public Function AddRow(Optional ByVal RowName As String)
As Boolean
Dim lFieldCount As Long, lCtr As Long
Dim sRowName As String
If RecordSetReady = False Then Exit Function
If poRs.EOF Or poRs.BOF Then Exit Function
lFieldCount = poRs.Fields.Count
sRowName = TrimWithoutPrejudice(RowName)
If sRowName <> "" Then TagStart sRowName
For lCtr = 0 To lFieldCount - 1
  AddField poRs.Fields(lCtr).Name
Next
If sRowName <> "" Then TagEnd
AddRow = True
End Function
```

```
Public Function AddRecordset(Optional ByVal StartTag As
String, Optional ByVal RowTag As String,
  Optional IncrementRowTag As Boolean) As Boolean
  Dim lRowCtr As Long
  Dim sRowName As String
  Dim dictTemp As New Scripting.Dictionary
  If RecordSetReady = False Then
    AddRecordset = False
    Exit Function
  End If
With poRs
On Error Resume Next
 .MoveFirst
 On Error GoTo 0
 If .EOF Then
   AddRecordset = False
   Exit Function
End If
If StartTag <> "" Then TagStart StartTag
Set dictTemp = TempAttributes
Set pDictAttributes = New Scripting.Dictionary
 Do While Not .EOF
   lRowCtr = lRowCtr + 1
   sRowName = RowTag & IIf(IncrementRowTag, lRowCtr, "")
   AddRow sRowName
 .MoveNext
 Loop
Set pDictAttributes = dictTemp
End With
If StartTag <> "" Then TagEnd
End Function
```

```
Public Sub Clear()
psXML = ""
End Sub

Public Sub TagStart(TagName As String)
Dim sTag As String, I As Long
Dim iCount As Integer, iCtr As Integer
sTag = TagName
If TrimWithoutPrejudice(psOutStandingTags(0)) = "" Then
  I = 0
Else
   I = UBound(psOutStandingTags) + 1
   ReDim Preserve psOutStandingTags(I)
End If
psOutStandingTags(I) = RemoveAllSpaces(sTag)
sTag = "<" & sTag
iCount = pDictAttributes.Count
For iCtr = 0 To iCount - 1
  sTag = sTag & " " & pDictAttributes.Keys(iCtr) & "=" &
pDictAttributes.Items(iCtr)
Next
sTag = sTag & ">" & vbCrLf
psXML = psXML & sTag
End Sub

Public Sub TagEnd()
Dim sEndTag As String
Dim I As Long
If UBound(psOutStandingTags) = 0 And
TrimWithoutPrejudice(psOutStandingTags(0)) = "" Then Exit Sub
I = UBound(psOutStandingTags)
```

```
sEndTag = "</" & TrimWithoutPrejudice(psOutStandingTags(l)) &
">" & vbCrLf
contador = contador + 1
inicial.txt_xml.Text = "Linea XML N°: " + CStr(contador)
DoEvents
psXML = psXML & sEndTag
If l = 0 Then
   psOutStandingTags(0) = ""
Else
   ReDim Preserve psOutStandingTags(l - 1) As String
End If
End Sub

Public Property Get CData() As Boolean
CData = pbCData
End Property

Public Property Let CData(ByVal NewValue As Boolean)
   pbCData = NewValue
End Property

Public Function AddField(FieldName As String, Optional
AllValues As Boolean = False) As Boolean
Dim oField As ADODB.Field
Dim vValue As Variant, sValue As String
If RecordSetReady = False Then
   AddField = False
   Exit Function
End If
If AllValues = True Then
   With poRs
      On Error Resume Next
```

```
        .MoveFirst
        On Error GoTo 0
        Do While Not .EOF
        AddField FieldName, False
        .MoveNext
        Loop
    End With
End If
On Error Resume Next
    Set oField = poRs.Fields(FieldName)
    If FieldCanBeString(oField) Then
        psXML = psXML & ADOFieldtoXMLField(oField)
    Else
        AddField = False
    End If
End Function

Public Function AddDisconnectedField(FieldName As String,
Value As Variant) As Boolean
Dim sValue As String
Dim sAns As String
Dim sFieldName As String
sFieldName = RemoveAllSpaces(FieldName)
If IsNull(Value) Then
    sValue = ""
ElseIf Value = vbEmpty Then
    sValue = ""
Else
    On Error Resume Next
    sValue = CStr(Value)
    If Err.Number <> 0 Then
```

```
        Err.Raise ERR_INVALID_VALUE, ,
ERR_INVALID_VALUE_DESC
        Exit Function
        AddDisconnectedField = False
    End If
End If
On Error GoTo 0
sAns = "<" & sFieldName & ">"
If pbCData = True Then
    sAns = sAns & "<![CDATA[" & sValue & "]]>"
Else
   sAns = sAns & sValue
End If
sAns = sAns & "</" & sFieldName & ">" & vbCrLf
psXML = psXML & sAns
AddDisconnectedField = True
End Function

Public Property Get XML() As String
XML = "<?xml version=""1.0"" encoding=""ISO-8859-2""?>" & vbCrLf
& psXML
End Property

Public Function PersistXML() As Boolean
Dim iFileNum As Integer
Dim sFileName As String
Dim lCtr As Long
iFileNum = FreeFile
sFileName = OutputFile
If sFileName = "" Then
    Err.Raise ERR_NOFILENAME, , ERR_NOFILENAME_DESC
    Exit Function
```

```
End If
On Error Resume Next
Open sFileName For Output As #iFileNum
If Err.Number <> 0 Then
   Err.Raise ERR_FILENAME_INVALID, ,
ERR_FILENAME_INVALID_DESC
   Exit Function
End If
On Error GoTo 0
If psOutStandingTags(0) <> "" Then
   For lCtr = 0 To UBound(psOutStandingTags)
      TagEnd
   Next
End If
Print #iFileNum, XML
Close #iFileNum
pbWriting = False
PersistXML = True
Exit Function
End Function

Private Function RecordSetReady() As Boolean
RecordSetReady = False
If poRs Is Nothing Then
   Err.Raise ERR_NO_RECORDSET, ,
ERR_NO_RECORDSET_DESC
   Exit Function
End If
If poRs.State = 0 Then
   On Error Resume Next
   poRs.Open
   If Err.Number <> 0 Then
```

```
        Err.Raise ERR_NOT_OPEN, , ERR_NOT_OPEN_DESC
        Exit Function
    End If
End If
RecordSetReady = True
End Function

Private Function FieldCanBeString(FieldObj As ADODB.Field)
As Boolean
 If IsObject(FieldObj.Value) Then
    FieldCanBeString = False
 Else
    Select Case FieldObj.Type
       Case adBinary, adIDispatch, adIUnknown, adUserDefined
          FieldCanBeString = False
       Case Else
          FieldCanBeString = True
    End Select
 End If
End Function

Private Function ADOFieldtoXMLField(FieldObj As
ADODB.Field) As String
Dim sAns As String
Dim vValue As Variant
Dim sName As String
If FieldObj.Type = adEmpty Then
   vValue = ""
ElseIf IsNull(FieldObj.Value) Then
   vValue = ""
Else
   vValue = FieldObj.Value
```

```
End If
sName = RemoveAllSpaces(FieldObj.Name)
sAns = "<" & sName & ">"
If pbCData = True Then
   sAns = sAns & "<![CDATA[" & CStr(vValue) & "]]>"
Else
  sAns = sAns & CStr(vValue)
End If
sAns = sAns & "</" & sName & ">" & vbCrLf
ADOFieldtoXMLField = sAns
End Function

Private Sub Class_Initialize()
ReDim psOutStandingTags(0) As String
Set pDictAttributes = New Scripting.Dictionary
End Sub

Private Function TrimWithoutPrejudice(ByVal InputString As
String) As String
Dim sAns As String
Dim sWkg As String
Dim sChar As String
Dim lLen As Long
Dim lCtr As Long
sAns = InputString
lLen = Len(InputString)
If lLen > 0 Then
   For lCtr = 1 To lLen
      sChar = Mid(sAns, lCtr, 1)
      If Asc(sChar) > 32 Then Exit For
   Next
sAns = Mid(sAns, lCtr)
```

```
lLen = Len(sAns)
 If lLen > 0 Then
      For lCtr = lLen To 1 Step -1
        sChar = Mid(sAns, lCtr, 1)
        If Asc(sChar) > 32 Then Exit For
      Next
   End If
   sAns = Left$(sAns, lCtr)
End If
TrimWithoutPrejudice = sAns
End Function

Public Sub AddAttribute(AttName As String, AttValue As
String)
   On Error Resume Next
   pDictAttributes.Add AttName, """ & AttValue & """
End Sub

Private Sub Class_Terminate()
   Set pDictAttributes = Nothing
End Sub

Public Sub ClearAttributes()
   Set pDictAttributes = New Scripting.Dictionary
End Sub

Private Property Get TempAttributes() As Scripting.Dictionary
   Dim dictTemp As New Scripting.Dictionary
   Dim iCtr As Integer
   For iCtr = 0 To pDictAttributes.Count - 1
      dictTemp.Add pDictAttributes.Keys(iCtr),
pDictAttributes.Items(iCtr)
```

```
    Next
   Set TempAttributes = dictTemp
End Property

Private Function RemoveAllSpaces(ByVal InputString As
String) As String
Dim sAns As String
Dim lLen As String
Dim lCtr As Long, lCtr2 As Long
Dim sChar As String
lLen = Len(InputString)
sAns = InputString
lCtr2 = 1
For lCtr = 1 To lLen
   sChar = Mid(InputString, lCtr, 1)
   If sChar <> " " Then
      Mid(sAns, lCtr2, 1) = sChar
      lCtr2 = lCtr2 + 1
   End If
Next
If lCtr2 > 1 Then
   sAns = Left(sAns, lCtr2 - 1)
Else
   sAns = ""
End If
RemoveAllSpaces = sAns
End Function
```

Este código representa una clase que genera un archivo XML. La diferencia principal con le **Código Manual 1** es el manejo más estructurado de lo que es una línea bien formada de XML, nos referimos a que permite agregar atributos a cada *tag*. También nos permite agregar campos desconectados, lo que significa que podemos exportar los datos de un *recordset* y también agregarle campos adicionales con registros adicionales sin la necesidad de tenerlos incorporados en el *recordset* original. Nos permite especificar así también el directorio por defecto de salida en el que se guardarán los archivos XML creados.

- **Metodología DOM**

```
dim documento as domdocument   'todo el documento xml
dim raiz as ixmldomelement     'el nodo raiz
dim registro as ixmldomelement 'el tag por cada fila
dim elem1 as imxldomelement    'las distintas columnas
dim elem2 as ixmldomelement    'el contenido de cada columna

set documento = new domdocument 'crea el documento DOM en
memroria

set raiz = documento.createelemnt(txt_raiz) 'crea el nodo raiz
documento.appendchild raiz 'agrega el nodo raiz al documento

for j = 0 to rs.recordcount-1
set registro = documento.createelement(txt_registro)
raiz.appendchild regitro

for i = 0 to ubound(arr_columnas(0)) 'arr_columnas trae el nombre
de cada columna de la tabla
        set elem1 = documento.createelement(arr_columnas(i))
        registro.appendchild elem1
        elem1.text = rs(arr_columnas(i)).value
next i
```

```
rs.movenext
next j
```