Un Generador Automático de Planes de Migración de Datos*

J. A. Carsí, I. Ramos, J. Silva, J. Pérez, V. Anaya
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46071 Valencia - España
{pcarsi, iramos, jsilva, jeperez, vanaya}@dsic.upv.es

Abstract.– This paper presents a solution for the data migration problem in information systems that must evolve to adapt to new requirements. The solution uses the information provided by the conceptual schemas, which represent the evolution that the information system has undergone, in order to automatically generate data migration plans. The plan offers the maximal flexibility permitting the user to act always with the maximum possible freedom in its construction and execution. Thus, a solution to a seldom addressed problem in Software Engineering is offered, and a tool that supports it is presented. This tool has been developed by the Information Systems and Computing Department of the Universidad Politécnica de Valencia in collaboration with the software development firm CONSOFT.

Keywords: data migration plan, migration language, patterns, migration expressions, automatic generation, comparison criteria, comparison algorithm.

1. INTRODUCCIÓN

La primera ley de la ingeniería de sistemas de Bersoft [4] dice *"Sin importar en qué momento del ciclo de vida del sistema nos encontremos, el sistema cambiará y el deseo de cambiarlo persistirá a lo largo de todo el ciclo de vida."*. Es un hecho destacado por numerosos investigadores y profesionales que los sistemas informáticos deben evolucionar para adecuarse a los siempre cambiantes requisitos del entorno. Las estadísticas indican que entre el 65% y el 75% de las fuerzas vivas relacionadas con el mundo de la informática [7] y aproximadamente el 80% de los gastos totales del software [23] se dedican al mantenimiento del software existente.

Idealmente, a la hora de introducir una modificación en una aplicación informática se debe seguir un proceso (por ejemplo el ISO/IEEE 12207 [5]) en el cual sea identificada la razón del cambio, se analice cómo afecta el cambio a introducir en la aplicación[1], para posteriormente realizar los cambios allí donde sean necesarios, finalizando con una exhaustiva fase de pruebas que demuestre que la modificación no ha introducido errores. Todo el proceso anterior debe estar bajo un estricto control de cambios que recoja todos los documentos que se generan y almacene las diferentes versiones.

Artículo recibido el 25 de febrero de 2002.

* Este trabajo ha sido financiado con fondos del Proyecto CICYT del programa FEDER con ref. TIC 1FD97-1102.

[1] Tanto en los documentos de análisis, diseño, como en el código.

Lamentablemente, muchas empresas simplifican el proceso anterior reduciéndolo, en la mayoría de los casos, a un análisis de la modificación para posteriormente aplicarla directamente al código sin excesivas pruebas y menor documentación.

En la actualidad, numerosas herramientas CASE son capaces de generar todo o parte del código que implementa una aplicación a partir de la información que se introduce en los modelos del método que se esté utilizando. Así por ejemplo, herramientas como Rational Rose [13], Together [20] o System Architect [19], son capaces de generar esqueletos de programas en lenguajes de programación como Java, C++, VBasic y *scripts* en SQL para generar las bases de datos necesarias a partir de la información estructural que se introduce mediante dichas herramientas en los diagramas de clases. Otras herramientas CASE como OO-Method/CASE [10] u Oblog/CASE [14], son capaces de generar aplicaciones completas a partir de la información de modelado que se introduce en sus modelos gracias a su sólido soporte formal. En el caso de OO-Method es el lenguaje de especificación formal OASIS, actualmente por su versión 3.0 [6].

Todas las herramientas CASE anteriores no ofrecen un buen soporte al paso del tiempo. La introducción de un nuevo requisito en las aplicaciones en funcionamiento, siguiendo algún método de gestión de cambios, consiste en introducirlo en los modelos pertinentes para posteriormente y aprovechando las capacidades generadoras de las herramientas CASE, regenerar la aplicación y el esquema de la base de datos acorde con el modelo del sistema actualizado. Finalmente, existen dos esquemas conceptuales del sistema de información, uno inicial con la definición del sistema antes de introducir los nuevos requisitos y uno final en el que ya se han introducido. Además, existen dos bases de datos, una inicial con toda la información que se ha generado mientras la aplicación ha estado en funcionamiento, y una final que satisface los nuevos requisitos del sistema, sin información. El problema es ¿cómo trasvasar la información de la base de datos inicial a la final?

La solución ofrecida en la siguiente, a partir de la información disponible en los modelos generados con las herramientas CASE, por comparación de los esquemas que definen el sistema original y el sistema evolucionado que recoge los nuevos cambios, es posible generar de manera automática planes de migración de datos cuya ejecución migre y transforme la información almacenada en la base de datos inicial a la final.

De esta forma, se da solución al problema de la evolución de los sistemas de información que han de cambiar para adaptarse a los nuevos requisitos de una manera sencilla y elegante. Dicha solución genera de manera automática los planes de migración de los datos y toda la documentación necesaria para documentar el cambio, agilizando en gran medida el tiempo necesario para adaptarse a los cambios.

El artículo está dividido en cuatro secciones. Tras la introducción, en el segundo apartado se presentan las diferentes fases en las que se ha dividido la herramienta de generación de planes de migración de datos. En el tercer apartado se presenta algunos trabajos relacionados con el presente. Y finalmente, en el cuarto apartado se presentan las conclusiones y los trabajos futuros.

2. FASES DEL GENERADOR AUTOMÁTICO DE PLANES DE MIGRACIÓN DE DATOS

La Figura 1 muestra las fases en las que se ha divido el generador de planes de migración:

1. Un comparador de esquemas *conceptuales aprovecha la información generada por la herramienta CASE para comparar la estructura de los modelos y calcular las diferencias que hay entre ambos.*
2. Un generador de planes *utiliza las semejanzas y diferencias detectadas por la primera fase para construir de manera automática por aplicación de diversos patrones un plan de migración en un lenguaje OO independiente del sistema gestor de bases de datos (SGBD) que se esté utilizando.*
3. *Finalmente, el plan de migración que se ha construido en la fase anterior debe ser ejecutado para que se realice la migración de los datos. Un* traductor *realiza esta tarea compilando el plan a paquetes DTS*[2] *que pueden ser ejecutados en SQL Server.*

2.1. Comparador de Esquemas Conceptuales

La primera fase de la migración de datos [17] consiste en determinar los orígenes y los destinos de los datos para realizar la migración. Para ello, se realiza un proceso comparativo a través del cual se determina qué modificaciones han sufrido los elementos de los esquemas conceptuales; y de esta manera, conocer entre qué elementos debe realizarse el trasvase de información. La comparación de esquemas conceptuales tiene como entrada los dos esquemas entre los que se va a realizar la migración; y como salida, qué elementos de los esquemas han sido eliminados, qué elementos han sido añadidos, y finalmente qué elementos han sufrido modificaciones pero deriva uno del otro.

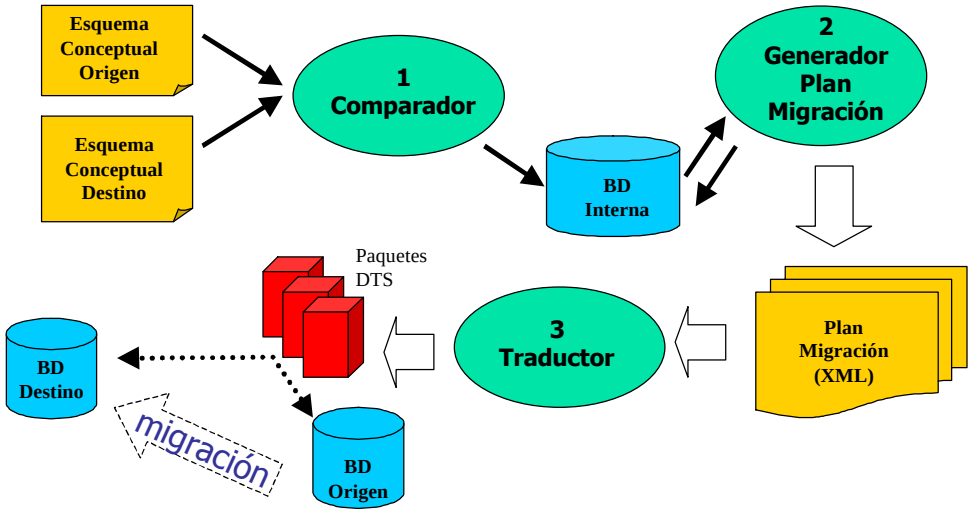

Figura 1. Fases del generador automático de planes de migración de datos.

El trabajo realizado para obtener un algoritmo capaz de automatizar la comparación de esquemas conceptuales, se ha dividido en dos secciones independientes:

1. Un estudio para determinar qué tipo de algoritmos son adecuados y presentan un acercamiento mayor al problema tratado. Finalmente, se ha diseñado un algoritmo específico de comparación de esquemas conceptuales.

[2] Data Transformation Services.

2. La determinación de qué información deben utilizar los algoritmos para comparar. Dicha información constituye lo que se ha denominado ‘criterio de comparación’, el cual es parcialmente independiente del algoritmo utilizado.

El análisis algorítmico realizado ha obtenido como resultado un algoritmo capaz de generar automáticamente las diferencias existentes entre dos esquemas conceptuales OASIS [18]. Dichas diferencias vienen dadas en términos de inserciones, modificaciones y borrados. El algoritmo está basado en técnicas de programación dinámica y de teoría de grafos, debido a que aprovecha la estructura de árbol de los esquemas conceptuales. Para procesar los esquemas conceptuales, se divide el espacio de comparación en cuatro clases distintas de elementos (clases, atributos, relaciones de agregación y relaciones de especialización) como se muestra en la Figura 2. Dicho fraccionamiento reduce el espacio de búsqueda limitando el número de comparaciones y consecuentemente, el tiempo de procesamiento del algoritmo.

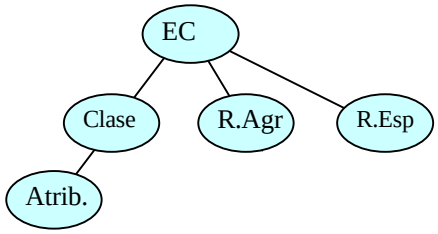

Figura 2. Estructura de árbol para la representación de esquemas conceptuales.

El algoritmo considera durante el proceso de comparación información semántica, siendo capaz de detectar modificaciones complejas como pueden ser *Inline* (unión de varias clases en una sola) o *Duplicate* (duplicado de la definición de una clase en varias) [16].

El algoritmo de comparación es capaz de utilizar información de distinta naturaleza para comparar los elementos de los esquemas conceptuales. Dicha información puede ser el nombre de los elementos, su identificador, fechas de creación, etc. Al aplicar el algoritmo de comparación utilizando diferentes criterios de comparación se pueden obtener resultados distintos dependiendo del contexto en el que sea utilizado.

Se llama *criterio de comparación* a la información utilizada para determinar si dos elementos de dos esquemas diferentes proviene uno del otro o no. El algoritmo propuesto es capaz de considerar en la actualidad cinco criterios de comparación distintos junto con las posibles combinaciones entre ellos. Para llegar a dicha selección de criterios de comparación se ha llevado a cabo un experimento utilizando siete criterios distintos para comparar una muestra de veinte esquemas conceptuales. A partir de dicho experimento se han obtenido estadísticas representativas del comportamiento de cada criterio ante una gran diversidad de situaciones.

El algoritmo propuesto junto con el diseño de los criterios de comparación que es capaz de utilizar son la primera fase de la herramienta que ha sido desarrollada para la generación de planes de migración. Se han elaborado los modelos conceptuales de la comparación de esquemas y a partir de los mismos se ha generado automáticamente el código de la aplicación utilizando la herramienta CASE OO-Method, la misma a la que da soporte de evolución. El resultado es una herramienta capaz de obtener de manera automática las diferencias existentes entre dos esquemas conceptuales cualesquiera.

Para la validación de la herramienta se han realizado diversas pruebas de comparación con esquemas conceptuales industriales, obteniéndose tasas medias de acierto cercanas al 98%. La

herramienta genera como salida el conjunto de correspondencias entre los elementos del esquema destino y origen así como aquellos para los que no se ha encontrado su correspondiente.

2.2. Generación de un Plan de Migración de Datos

A partir de las correspondencias generadas en la fase inicial del proceso de migración, de los esquemas conceptuales y del *orden de migración* [1] calculado a partir de las dependencias (relaciones) entre elementos del esquema conceptual destino, la segunda fase [11] genera de forma automática la primera versión de un *plan de migración de datos*.

Un plan de migración está formado por un conjunto de cambios expresados en un lenguaje de migración, cuya posterior ejecución permite migrar los datos de una BD origen a una BD destino de forma adecuada. La creación de un plan de migración completo debe realizarse incorporando cada uno de los cambios necesarios para que pueda hacerse dicha migración.

Cada uno de los posibles cambios que pueden realizarse en las bases de datos, se expresa mediante un lenguaje pivote declarativo. Este *lenguaje de migración* especifica los cambios basándose en el modelo orientado a objetos, debido a que la información que maneja son esquemas conceptuales orientados a objetos OASIS. De esta forma, se permite especificar el plan de migración de forma correcta y próxima al usuario de la aplicación, independientemente del SGBDR en el que se vaya a ejecutar finalmente.

El plan de migración se construye gracias a la composición de diferentes elementos: expresiones de migración, cambios y módulos de migración. El nivel de abstracción más elevado de esta composición de elementos lo constituyen los módulos de migración, mientras que el gránulo más pequeño son las expresiones de migración.

Una *expresión de migración* es el concepto general para hacer referencia a las diferentes expresiones que permite especificar el lenguaje de migración diseñado. Cada una de ellas ayuda a especificar unos determinados elementos de un esquema conceptual, tiene una connotación semántica diferente y unas restricciones sintácticas claramente definidas.

Un *cambio* contiene el conjunto de expresiones de migración necesarias para expresar las modificaciones de un determinado elemento del esquema conceptual origen y los filtros que se le han de aplicar a su población, para poderlo migrar correctamente.

Un *módulo de migración* esta asociado a una clase, una relación de agregación o una relación de especialización del esquema conceptual destino. Los módulos representan unidades de migración, cuya composición mediante agregaciones y especializaciones constituyen el plan de migración. Un ejemplo de dicha composición es el módulo de migración de una agregación, que está compuesto por los módulos de migración de clase de sus clases compuesta y componente.

El proceso de generación de un plan de migración de datos se ha dividido en dos partes, cada una de ellas genera y parte de una información distinta:

1. La primera parte consiste en generar automáticamente tanto la estructura del plan de migración de datos, como cada una de las expresiones que lo componen. Para ello, se define un módulo de migración vacío por cada elemento del esquema conceptual destino con el orden y la composición que indique el orden de migración. Después, se calculan automáticamente las expresiones de cada

uno de los cambios asociados a un módulo de migración mediante el uso de *patrones de expresiones de migración*. Posteriormente, cada una de las expresiones generadas se incluye en el módulo adecuado y en el lugar correcto basándose en los *patrones de migración*. El resultado de este subproceso automático es una primera versión de un plan de migración completo.

2. La segunda parte tiene como objetivo la validación del plan de migración que se ha generado automáticamente por la primera parte. Para ello, el analista necesita interaccionar con la herramienta de migración de datos, no solo para validar el plan de migración propuesto, sino también para modificarlo cuando no exprese lo que él desee.

Por este motivo, se ha diseñado una interfaz gráfica de usuario en la que se muestran las correspondencias entre los elementos del esquema conceptual final (EC') y los del esquema conceptual original (EC). Se han contemplado varias representaciones visuales para mostrar los elementos que forman parte de ambos esquemas y finalmente se ha optado por una visualización en forma arborescente. Por otro lado, mediante un código de colores y de símbolos se muestran las diferencias entre ambos esquemas, además de indicar los cambios, mediante expresiones asociadas a los elementos (ver Figura 3).

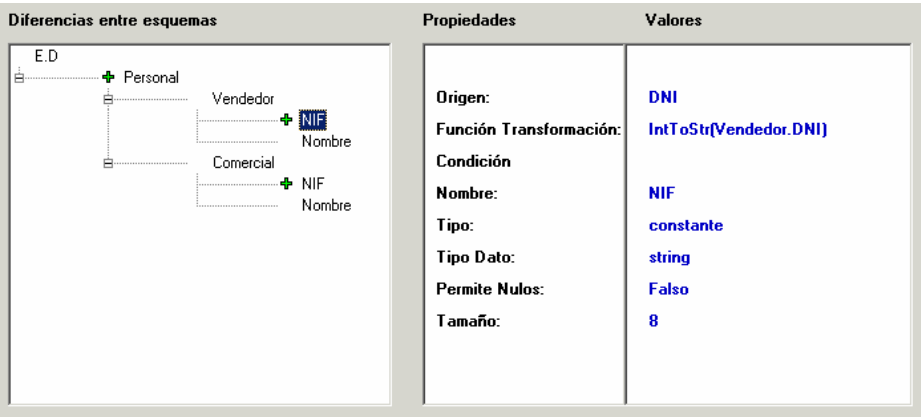

Figura 3. Visualización de diferencias entre dos esquemas conceptuales.

Los cambios se muestran desde la perspectiva del esquema conceptual final, ya que el objetivo es llenar de información la base de datos final a partir de la base de datos inicial. Esta visualización muestra en el primer nivel del árbol las clases y relaciones del esquema final en el orden en que se han citado. Si se despliegan cualquiera de estos elementos, se encuentra a un segundo nivel de anidamiento el elemento o elementos del esquema conceptual origen del que provienen sus datos, que serán de su mismo tipo o compatibles. Finalmente, en el tercer nivel de anidamiento del árbol se encuentran los atributos de la clase destino de la que cuelgan, la cual aparece en el primer nivel del árbol. Se ha de destacar que en este último nivel del árbol aparece en la parte derecha de la pantalla de exploración el atributo origen del cual se extraen los datos para el atributo destino seleccionado. Dicho atributo origen pertenece a la clase del esquema conceptual inicial del que cuelga el atributo destino. La representación arbórea de la figura 3 muestra que los objetos de la clase *Personal* provienen de los objetos de las clases *Comercial* y *Vendedor*. También da a conocer que los valores de los atributos *NIF* y *Nombre*, de la clase *Personal* provienen de los atributos *DNI* y *Nombre* de las clases *Comercial* y *Vendedor*, cuya especificación no aparece en la estructura arborescente, sino en las propiedades que aparecen en la ventana de exploración asociada al árbol de visualización.

Por otro lado, siempre que se siga la estructura del árbol a la hora de validar la información, la complejidad de ésta se reduce, ya que primero se validan las clases y luego los atributos. De esta manera, se tiene la certeza de que los atributos que se asocien a un atributo del esquema final, han de pertenecer a la clase del esquema inicial que se ha asociado a la clase del esquema final a la que pertenece dicho atributo.

Otro punto a considerar es la visualización de las operaciones realizadas en el esquema conceptual inicial que lo hicieron evolucionar al esquema conceptual final. Las posibles operaciones que pueden sufrir los elementos de un esquema conceptual son: inserción, modificación y borrado. No obstante, al visualizar las diferencias desde la perspectiva del esquema conceptual final, solamente se consideran la inserción y modificación de elementos. Los borrados no se visualizan en el árbol, por no existir en el esquema conceptual destino.

Finalmente, una vez completadas las dos partes de esta fase del proceso de migración se obtendrá un plan de migración de datos validado por el analista. Dicho plan será compilado por la tercera fase de la herramienta que hará efectiva la migración. Con el objetivo de que la fase de ejecución tenga el mayor conocimiento posible del proceso de migración, en lugar de proporcionarle el plan de migración de forma textual, se estructura en formato XML. De este modo, se facilita la lectura del plan de migración y el proceso de traducción.

2.3. Traductor del Plan de Migración

Una vez validado el plan de migración sintáctica y semánticamente, se procede a traducirlo a paquetes de código [2]. El objetivo de la ejecución de dichos paquetes es la migración de la información entre bases de datos relacionales. Dicha migración puede realizarse entre bases de datos heterogéneas (Oracle, Access, dBase, SQLServer, etc). Para realizar dicha tarea no basta con la simple generación de sentencias SQL, ya que la expresión de transformaciones complejas sobre los datos origen no es viable al no disponerse de un lenguaje imperativo.

La solución adoptada es la generación de código interpretado por herramientas específicas de migración. Entre las distintas posibilidades que existen, se ha optado por DTS (Data Transformation Services de Microsoft®), herramienta que acompaña a SQL Server. Entre las ventajas que proporciona generar código para DTS está la capacidad de migrar información entre multitud de fuentes de datos (ficheros, bases de datos jerárquicas, bases de datos relaciones heterogéneas, etc), la posibilidad de permitir expresar transformaciones complejas sobre los datos haciendo uso de lenguajes *script*, y las opciones de copia rápida (Bulk Copy) y de carga rápida (Fast Load Option), que optimizan el rendimiento de la migración. Pero a pesar de todas las facilidades ofrecidas, un proceso de migración necesita de la implementación de miles de líneas de código. El traductor implementado se encarga de ahorrar ese trabajo.

La entrada principal del traductor la constituye el plan de migración. Junto a éste, para traducirlo a paquetes de código se necesita de información adicional que no está disponible en él por cuestiones de rendimiento y evitar duplicidad. En el plan de migración pueden aparecer expresiones de navegación y atributos derivados origen, ya sea en los filtros poblacionales, en las condiciones o en las funciones de transformación, que hacen necesaria la consulta de los esquemas conceptuales para obtener dicha información.

La salida del traductor es un conjunto de paquetes DTS. Un paquete DTS [15, 22] contiene un conjunto de conexiones asociadas a fuentes de datos de dónde leer y salvar la información. A su vez el

paquete posee un conjunto de tareas que pueden ser de distinto tipo. Las únicas que se utilizan en este trabajo son la de bombeo de datos y la de consulta guiada por datos. La primera para la migración de clases y especializaciones, ya que únicamente se realizan inserciones de tuplas. La segunda para las agregaciones y asociaciones, ya que realizan modificaciones de tuplas existentes. Una tarea de bombeo de datos tiene asociada dos conexiones, una para leer datos y otra para depositarlos. La lectura de información se puede realizar mediante una sentencia de consulta SQL. Dicha información es asignada y transformada en el cuerpo en *script* de una función de transformación que toda tarea de bombeo de datos posee. La forma de asignar valores en el cuerpo *script* es:

$$D_i = f\_trans(O_1, ..., O_n)$$

donde D_i es el atributo i-esimo de la tabla destino $\forall i$: $1 <= i <= m$, siendo m el número de campos de la tabla destino. *f_trans* es la función de transformación que se aplica sobre los operandos $O_1, ..., O_n$, donde O_k es un campo origen o una expresión introducida por el usuario.

El traductor lee el plan de migración y los esquemas conceptuales y extrae la información necesaria para generar paquetes automáticamente. La información extraída es almacenada en estructuras intermedias en memoria principal. Estas estructuras guardan, de una forma adecuada la información, de los esquemas conceptuales y del plan de migración, para la generación de paquetes. Las estructuras con la información del plan de migración mantienen referencias a aquellas que poseen la información de los esquemas para facilitar el acceso y almacenamiento de la información requerida. Las estructuras poseen procedimientos adecuados para deducir información con la finalidad de generar los paquetes. Estos procedimientos son llamados por el traductor para generar el conjunto de paquetes DTS de salida. En ello se resuelve la diferencia semántica entre el contexto OO del plan de migración, y el contexto relacional de los paquetes de código.

En el proceso de traducción se establecen correspondencias semánticas entre las secciones del plan de migración orientado a objetos generado y evaluado por el analista en la segunda fase, y los paquetes DTS. Dichas correspondencias son necesarias para definir patrones de generación de código. La forma de cada patrón depende de la representación que en base de datos se hace de los elementos del esquema conceptual.

La siguiente tabla muestra las correspondencias entre las secciones de un plan de migración y un paquete DTS:

PLAN DE MIGRACIÓN	**CÓDIGO DTS**
Módulo de migración	Paquete
Submódulo de migración	Tarea
Filtro poblacional	Condición WHERE de la consulta de la tarea
Funciones de transformación	Función en lenguaje script de la sección de transformación de una tarea.
Condiciones sobre atributos	Sentencias condicionales indicadas con un lenguaje script en la sección de transformación de la tarea

A partir del conjunto de las correspondencias y dependiendo del tipo de módulo de migración (depende del elemento del esquema conceptual destino al que está asociado) se expresa un patrón que indica la forma de generar código de forma automática a partir de cada módulo de entrada.

Existen dos modos de funcionamiento del traductor:

- *Modo intérprete: el traductor genera un conjunto de paquetes, tantos como clases, relaciones de agregación y asociación haya en el esquema conceptual evolucionado. La ejecución de paquetes del mismo tipo es independiente, pero se han de ejecutar los paquetes que migran la información de clases antes que los que provocan la migración de agregaciones y asociaciones. La migración de las especializaciones se hace junto a la de clases, de forma intrínseca. El usuario puede observar el correcto resultado de la migración parcial consultando los estados intermedios alcanzados para evaluar si es o no correcto el proceso.*

- *Modo compilado: el traductor genera un único paquete capaz de migrar toda la base de datos origen de una vez. El orden de migración lo impone el propio traductor mediante relaciones de precedencia entre tareas del paquete.*

3. TRABAJOS RELACIONADOS

En la actualidad, son los SGBD los que dan soporte a la evolución de los datos, ya sean relacionales como ORACLE [9] o SQL-Server [8] u orientados a objetos como O2 [3], Poet [12] o Versant [21] ya que éstos son las que almacenan la información que se genera con las aplicaciones. Muchos SGBD permiten la migración de datos mediante sus herramientas ETL (Extract, Transform & Load), bien ejecutando de forma individualizada sentencias SQL o bien codificando *scripts* que posteriormente serán ejecutados sobre las bases de datos. Estas herramientas no ofrecen un soporte automático a la generación de *scripts* para la migración de los datos, tal y como propone nuestra solución. Las herramientas ETL sólo proporcionan un entorno de interacción amigable para la migración de datos entre las bases de datos origen y destino. Por este motivo, no consiguen paliar el gran número de personas implicadas en el proceso de migración de datos y el alto coste temporal que desencadena. En todas estas soluciones el usuario ha de concebir la migración a un nivel de abstracción bajo, donde es difícil expresar transformaciones complejas. A su vez, se carece de integración entre las distintas fases del proceso de evolución, por lo que se requiere de personal especializado en las distintas fases de desarrollo de software. Las soluciones existentes en el mercado obligan a implementar miles de líneas de código en un lenguaje específico, con el fin de dar solución a un problema de migración concreto, requiriendo gran cantidad de tiempo y recursos en las etapas de codificación, consiguiendo soluciones de baja calidad.

Un enfoque más próximo a nuestra solución, debido a que es un proceso automático y se basa en la evolución del esquema, es la herramienta TESS que presenta Barbara Staund Lerner en su artículo [16]. TESS analiza el código asociado a los esquemas relacionales tras su previa traducción a un lenguaje independiente. Sin embargo, nuestra aproximación se basa la comparación de los esquemas conceptuales OO independientes del SGBD en el que se almacena la información, ahorrando el proceso de traducción para conseguir mayor abstracción.

Respecto a la visualización de las diferencias entre esquemas conceptuales, existen herramientas como *Visual Differencing* proporcionada por Rational Rose [13] que permite visualizarlas. La visualización es en forma árbol mostrando la unión de los esquemas que se están comparando. Las diferencias encontradas se muestran desde el punto de vista del esquema origen y el criterio de comparación es único, no se puede aplicar a subconjuntos del esquema y no es modificable de forma manual. Esta herramienta realiza la comparación entre esquemas conceptuales, pero su resultado no es utilizado posteriormente, y por lo tanto, no persigue el mismo objetivo que el presente trabajo, utilizar

las diferencias entre esquemas para obtener de forma automática las expresiones que constituirán el plan de migración de datos.

4. CONCLUSIONES

Se ha presentado una solución al problema de migración de los datos en las aplicaciones para adecuarse a los nuevos requisitos que surgen con el paso del tiempo. Además, se ha mostrado la arquitectura de una herramienta que, basándose en una herramienta CASE existente, aprovecha la información generada por ésta para dirigir el proceso de migración de los datos de forma automatizada.

De forma resumida, las fases en las que se ha dividido la herramienta para migrar la información consiste en: primero, comparar la información estructural de los esquemas inicial y evolucionado que definen el sistema de información; segundo, generar por aplicación de unos patrones una primera versión del plan de migración que debe de ser validada por el analista de la aplicación; para finalmente, tras un proceso de traducción a paquetes DTS, aplicar el plan generado a las bases de datos entre las que se desea migrar la información.

El proceso de modificación de las aplicaciones se simplifica gracias a la utilización de la herramienta descrita en este artículo. El proceso consiste en analizar el cambio, introducir la información de los nuevos requisitos en los modelos adecuados, regenerar el código de la aplicación y la base de datos mediante la herramienta CASE y, finalmente, validar el plan de migración de datos que es propuesto por la herramienta de migración.

Pruebas experimentales realizadas han demostrado que la utilización de esta herramienta ha conseguido reducir el número de personas involucradas y el tiempo dedicado en la modificación de las aplicaciones alrededor de un 80%.

REFERENCIAS

[1] S. Abad, J.A. Carsí, I. Ramos, “Obtención de un Orden de Migración para los Elementos de un Esquema Conceptual Orientado a Objetos”, *Taller de Evolución, VI Jornadas de Ingeniería del Software y Bases de Datos*, Almagro (Ciudad Real), Nov. 2001.

[2] V. Anaya, *Generación de Módulos de Trasformación para la Migración de Información entre Bases de Datos a Partir de Migración*, Proyecto Fin de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Sep. 2001.

[3] Ardent Software, “O2”, http://www.ardent.com/.

[4] E.H. Bersoff, V.D. Henderson, S.G. Siegel, *Software Configuration Management*, Prentice-Hall, 1980.

[5] IEEE, *IEEE Std 12207. Standard for Software Maintenance*, IEEE Computer Society Press. 1992.

[6] P. Letelier, P. Sanchez, I. Ramos, O. Pastor, *OASIS 3.0: Un Enfoque Formal para el Modelado Conceptual Orientado a Objeto*, Servicio de Publicaciones, Universidad Politécnica de Valencia, SPUPV -98.4011, ISBN 84-7721-663-0, 1998.

[7] J.R. McKee, *“Maintenance as a Function of Design”, Proc. AFIPS National Computer Conf.*, Las Vegas, pp. 187-93.

[8] Microsoft® Corporation, “SQL Server”, http://www.microsoft.com/sql.

[9] Oracle® Corporation, “Oracle”, http://www.oracle.com.

[10] O. Pastor, E. Insfrán, V. Pelechano, J. Romero, J. Merseguer, “OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods”, *Conference on Advanced Information Systems Engineering (CAiSE '97)*. LNCS 1250, Springer-Verlag 1997, ISBN: 3-540-63107-0, Barcelona, pp. 145-159, Jun. 1997.
[11] J. Pérez, *Generación de un Plan de Migración de Datos para Esquemas Conceptuales OASIS Generados con OO-Method/CASE*, Proyecto Fin de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Sep. 2001
[12] POET Software, “POET”, http://www.poet.com.
[13] Rational Software, “Rational Rose”, http://www.rational.com/products/rose/.
[14] A. Sernadas, J.F. Costa, C. Sernadas, “Object Specifications Through Diagrams: OBLOG Approach”, INESC Lisboa, 1994.
[15] D. Siebold, Visual Basic Developer’s Guide to SQL Server, Sybex, 2000.
[16] B. Staund Lernerm, “A Model for Compound Type Changes Encountered in Schema Evolution”, *ACM Transactions on Database Systems (TODS)*, Mar. 2000, Vol. 25 No. 1.
[17] J. Silva, *Comparación de Esquemas Conceptuales OASIS para la Obtención de las Diferencias Producidas por un Proceso de Evolución*, Proyecto Fin de Carrera, Facultad de Informática, Universidad Politécnica de Valencia, Sep. 2001.
[18] J. Silva, I. Ramos, J.A. Carsí, “Comparación Automática de Esquemas Conceptuales Orientados a Objeto”, *Ingeniería Informática, Revista electrónica del DIICC, Edición número 7*; ISSN : 0717 – 4195, Ene. 2002.
[19] System Architect, http://www.popkin.com/products/sa2001/systemarchitect.htm.
[20] TogetherSoft Corporation, “Together”, http://www.togethersoft.com/.
[21] Versant Object Technology, “Versant”, http://www.versant.com/.
[22] Wrox®, “Recursos del Programador de la Editorial Wrox”, http://p2p.wrox.com/archive/sql_server/2001-03/.
[23] E. Yourdon, "RE-3: Re-engineering, Restructuring and Reverse Engineering”, *American Programmer*, 2 (4), pp. 3-10, 1989.