Comparación automática de esquemas conceptuales orientados a objeto.

J.Silva, J.A.Carsí, I.Ramos
{ jsilva | pcarsi | iramos }@dsic.upv.es

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
E-46071 Valencia – España

Resumen

En este trabajo se realiza un estudio algorítmico de cómo comparar esquemas conceptuales orientados a objeto en un contexto de migración de datos. Se presenta un algoritmo de comparación de esquemas basado en la comparación de árboles que aprovecha información semántica para optimizar la eficiencia de la comparación. Se calcula a partir de dos esquemas cualesquiera, y de manera automática, su plantilla de equivalencias y diferencias en términos de inserciones, modificaciones y borrados. El presente trabajo forma parte del desarrollo de una herramienta para la migración automática de bases de datos, que está siendo desarrollada en el Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.

Palabras clave: Comparación de esquemas, Algoritmos de comparación, Anclajes, Criterios de comparación, Recorrido de árboles, Segmentación de esquemas.

1. Introducción

Muchos sistemas y procesos pueden ser representados como estructuras de árbol, y en ocasiones se da la necesidad de realizar una comparación entre dichas estructuras con el fin de determinar la posible interoperación, las diferencias existentes, o simplemente el grado de similitud entre dichos sistemas. Un caso particular se da en la ingeniería del software cuando lo que queremos comparar son esquemas conceptuales de sistemas de información orientados a objeto.

La comparación de esquemas conceptuales tiene un especial interés en la migración automática de bases de datos. Idealmente, la evolución del software se realiza a nivel de esquema conceptual, y a partir del mismo se generan automáticamente las aplicaciones y bases de datos del sistema de información. En ocasiones ocurre que las bases de datos del sistema original contienen información, y dicha información debe ser traspasada al nuevo sistema, este hecho es conocido como migración de datos. El principal inconveniente en la migración de datos es que las bases de datos entre las que se quiere realizar la migración suelen ser muy distintas, y en ocasiones incompatibles.

En una migración de datos, es fundamental conocer exactamente el grado de evolución sufrido por el esquema y la base de datos asociada que va a ser migrada. Se hace indispensable conocer qué modificaciones han sufrido los elementos del esquema origen, así como qué inserciones y borrados se han producido durante el proceso de evolución. También es una tarea importante determinar qué información se debe usar para identificar el origen y el destino de la migración. Por ejemplo, si en la BD del sistema origen se dispone de la tabla A, y en el destino también se dispone de la tabla A, ¿qué nos asegura que la información de A en el origen debe ser traspasada a la tabla A del destino? Puede haber ocurrido que la tabla A del origen se llame B en el destino, y esa información es determinante para la migración.

En este contexto, adquiere especial relevancia un algoritmo capaz de automatizar el proceso de comparación de esquemas conceptuales y que además no se limite a la mera comparación de

árboles, sino que aproveche información semántica de los esquemas durante la comparación para resolver posibles ambigüedades. En el contexto de migración de datos, este trabajo se corresponde con la primera fase a realizar por la herramienta de migración.

En el apartado 2 se presenta el problema de la comparación de esquemas conceptuales y su estrecha similitud con la comparación de árboles simples. Posteriormente, en el apartado 3, veremos cómo han tratado diversos autores el problema de la comparación, y mostraremos el hecho de que los algoritmos presentados hasta el momento no son adecuados para tratar el problema en contextos de migración de datos. En el apartado 4, se presenta un algoritmo que resuelve el problema de la comparación de esquemas conceptuales, y que es capaz de generar automáticamente las correspondencias que servirán de base para la generación automática del plan de migración [Per01]. Finalmente, en el apartado 5 se extraerán las conclusiones fruto del estudio realizado.

2. Presentación del problema

Se pretende obtener un algoritmo capaz de automatizar la comparación de esquemas conceptuales con el fin de migrar las instancias almacenadas entre las bases de datos de dichos esquemas. Dicho algoritmo, tendrá una doble utilidad; por un lado servirá para medir el grado de evolución existente entre dos esquemas; puesto que comparar implica medir, podremos medir aspectos como la complejidad ciclomática de los servicios, el número de clases, el número de atributos de las clases, nivel de herencia, etc. Por otro lado, cuando los esquemas conceptuales que se comparan sean modelos de un mismo sistema, el algoritmo servirá para identificar la evolución individual que ha tenido cada uno de los elementos de dichos esquemas, y de esta manera indicará los orígenes y los destinos de los datos a una herramienta de migración.

Algunas herramientas comerciales son capaces de identificar la evolución entre los esquemas sin necesidad de comparar, sino que guardan una traza de las operaciones de evolución realizadas. Esta solución tiene el problema de ser poco flexible puesto que obliga a que los dos esquemas que se comparan sean dependientes entre sí, es decir, uno sea la evolución del otro. El hecho de plantear una comparación de esquemas, permite que los esquemas que se comparan sean independientes entre sí, e incluso que tengan distinto formato. Además, la solución de la traza de servicios no es adecuada en un contexto de evolución de datos, puesto que, en la práctica, la mayoría de las migraciones que se desarrollan son realizadas entre sistemas antiguos , que no tienen un EC asociado, y que por lo tanto no existe una traza de evolución entre el modelo antiguo y el nuevo. En estos casos, se ha propuesto modelar el sistema legado a partir de su base de datos mediante un proceso de ingeniería inversa, y posteriormente realizar el proceso de comparación. Se extrae la conclusión de que existen casos, en los que es necesario aventurarse en un proceso de comparación, y por lo tanto es necesario un algoritmo específico de comparación de EC.

El algoritmo ha de ser capaz de generar automáticamente las diferencias existentes entre dos esquemas cualesquiera. Dichas diferencias vendrán dadas en términos de inserciones, modificaciones y borrados. Desde el punto de vista de la teoría de grafos, el problema que se nos plantea es un problema de comparación de grafos acíclicos no-dirigidos, es decir, de árboles. En nuestro caso, el objetivo que se persigue es migrar las instancias almacenadas de los elementos de los esquemas, por lo tanto sólo es necesario considerar aquellos elementos que tienen repercusión directa sobre la persistencia de los datos. Es necesario estudiar cuáles son esos elementos: Clases, Atributos y Relaciones de Especialización y Agregación [Car99]. A efectos de comparación, un esquema conceptual será considerado como un árbol de tres niveles de profundidad. Ver figura 1.

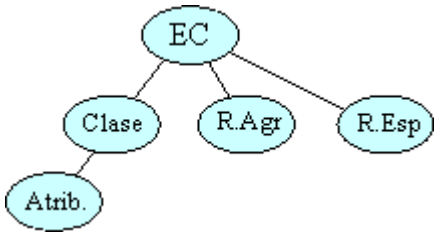


FIGURA 1: Representación de esquemas conceptuales en árboles.

En la figura 1 podemos ver que cada elemento del esquema conceptual queda representado como un nodo del árbol; siendo el nodo raíz el propio esquema conceptual.

Debemos fijar ahora una meta en la comparación, ¿qué esperamos obtener de la comparación de esquemas conceptuales? En un principio, podríamos pensar que lo más interesante es encontrar un conjunto de transformaciones (inserciones, modificaciones y borrados) del esquema conceptual origen, que nos lleven al esquema conceptual destino; y que además, esas transformaciones sean mínimas, es decir, queremos pasar de un árbol a otro realizando el menor número de transformaciones posibles. Nos encontraremos entonces ante un problema de optimización.

Realmente, no queremos un camino mínimo que nos lleve de un árbol a otro; lo interesante es tratar de identificar correctamente qué evolución ha tenido cada uno de los elementos del esquema conceptual inicial, sin importarnos el coste que pueda tener la transformación entre árboles. En este caso, el proceso algorítmico a utilizar es radicalmente diferente, y lo importante es identificar cada uno de los elementos de cada árbol y establecer una función biyectiva entre aquellos elementos identificados como correspondientes (dos elementos pertenecientes a dos esquemas distintos son correspondientes, un elemento se deriva del otro, si existe una correspondencia entre ambos elementos). A dicha función biyectiva la llamamos 'correspondencia'; a continuación se presenta la definición formal de correspondencia:

Correspondencia entre árboles: Intuitivamente, denota cómo una secuencia de operaciones transforman un árbol T en otro T', sin importar el orden en que se han aplicado las operaciones. Formalmente, una correspondencia entre árboles es expresada mediante la tripleta (M, T, T') donde T es el árbol inicial, T' el final, y M un conjunto de pares de enteros (i, j) que satisfacen:

1) $1 \le i \le |T|$, $1 \le j \le |T'|$;

2) Para cada par (i_1,j_1) y (i_2,j_2) en M se cumple:

a) $i_1 = i_2$ si y sólo si $j_1 = j_2$

b) $i_1 < i_2$ si y sólo si $j_1 < j_2$

c) $T[i_1]$ es un antecesor (descendiente) de $T[i_2]$ si y sólo si

$T'[j_1]$ es un antecesor (descendiente) de $T'[j_2]$.

Correspondencia entre elementos: Llamaremos correspondencia entre elementos (o simplemente correspondencia) a cada pareja (i,j) ☐ M de una correspondencia entre árboles.

Los objetivos principales que fijamos para este trabajo, son los siguientes:

1. Obtener un algoritmo de comparación de esquemas conceptuales que trate de identificar correctamente qué evolución ha tenido cada uno de los elementos del esquema

conceptual inicial.

2. Que el algoritmo sea capaz de establecer una correspondencia para cada elemento de los esquemas conceptuales.

3. Que el algoritmo no se limite a la comparación de árboles, sino que sea específico para EC; que utilice información semántica para comparar.

4. Que los EC a comparar no tengan por qué ser dependientes entre sí.

A continuación, vamos a realizar un estudio algorítmico que solucione el problema atendiendo a cada uno de los objetivos establecidos. Abordaremos en primera instancia, el problema del camino mínimo entre árboles, y seguidamente, el problema de la identificación de elementos entre esquemas conceptuales.

3. Estado del arte: El problema 'Árbol a Árbol'

El problema de la obtención de diferencias entre esquemas, es tratado actualmente por diferentes herramientas comerciales utilizando una traza de las operaciones de evolución. Este es el caso de VDIFF de Rational Rose. El principal problema que plantean estas herramientas es que los EC deben ser dependientes. Además, el hecho de que deba existir una traza de evolución entre los esquemas obliga a que los esquemas sean del mismo formato o pertenezcan a la misma aplicación. Estos problemas desaparecen con un proceso de comparación.

No se han encontrado en la bibliografía algoritmos específicos de comparación de esquemas, sin embargo sí existen diversos algoritmos para comparar árboles etiquetados, en los que se plantea el problema de encontrar la distancia mínima entre los mismos. Aunque en la mayoría de los casos no tendrá sentido plantear la comparación de esquemas como un problema de optimización, puede haber situaciones en las que sea interesante conocer el camino mínimo entre dos árboles. En cualquier caso, necesitaremos del conocimiento de estos algoritmos para el desarrollo de los presentados en el siguiente apartado; y además nos establecerán una cota inferior en el paso de un árbol a otro (estos algoritmos encuentran el camino mínimo entre los dos árboles).

Muchas estructuras y procesos pueden modelizarse como árboles etiquetados, y en ocasiones puede ser muy útil conocer cuál es la forma de pasar (o transformar) de un árbol a otro con el menor número de cambios posibles. Un caso particular de este problema ocurre cuando los árboles son de profundidad dos. Este es el caso de tratar de comparar dos cadenas de caracteres donde cada carácter está representado por un nodo que guarda una determinada posición entre los nodos hoja. Este problema fue abordado y resuelto por diversos autores. Sankoff [San72] y Wagner y Fisher [Wag74] presentaron un algoritmo capaz de computar una secuencia de operaciones de edición de mínimo coste. Dicho algoritmo presentaba un coste computacional O(n*m), siendo m y n el número de hojas de cada uno de los árboles. Posteriormente, Wong y Chandra [Won76]; y Aho, Hirschberg y Ullman [Aho76] probaron que para muchos modelos de computación el algoritmo de Sankoff es óptimo. Otro algoritmo capaz de solucionar el problema, también con coste óptimo, fue presentado en 1977 por Selkow [Sel77] en Knoxville.

Una generalización del problema, es considerar que los árboles pueden tener cualquier profundidad. En este caso, el problema aumenta su complejidad en un factor cuadrático con relación a la misma. Vamos a presentar a continuación un algoritmo que resuelve el problema y que está atribuido a Kuo-Chung Tai [Kuo79].

Partimos de la siguiente información:

- **r**: Función arbitraria de coste que asigna un valor no negativo a cada operación de edición ($a \rightarrow b$).

- **D(T, T')**: Distancia mínima entre dos árboles; formalmente, min{r(S) / S es una secuencia de operaciones que transforma T en T'.
- Coste(M): Representa el coste asociado a una correspondencia M. Dicho coste es función del número de inserciones, modificaciones y borrados.
- **MIN_M(i + 1, j + 1)**: min{coste(M) / M es una correspondencia de T(1:i+1) a T'(1:j+1), y (i+1, j+1) ∈ M}.
- **E[s: u: i, t: v: j]**: min{coste(M) / M es una correspondencia de T(s: i) a T'(t: j), y (s, t) ∈ M, y ningún descendiente de T[s] (T'[t]) en el camino de T[s] (T'[t]) a T[u] (T'[v]) es tocado por una línea de M}.

El algoritmo se compone de tres pasos:

1. Calcular E[s: u: i, t: v: j] para todo s, u, i, t, v, j, donde

   $1 \le i \le |T|$, $1 \le j \le |T'|$,

   T[u] (T'[v]) está en el camino de T[1] (T'[1]) a T[i] (T'[j]),

   T[s] (T'[t]) está en el camino de T[1] (T'[1]) a T[u] (T'[v]);

2. Calcular MIN_M(i, j) para todo i, j, donde $1 \le i \le |T|$, $1 \le j \le |T'|$;

3. Calcular D(i, j) para todo i, j, donde $1 \le i \le |T|$, $1 \le j \le |T'|$.

Dados dos árboles T y T' el algoritmo propuesto por Kuo-Chung Tai, computa la distancia entre ambos árboles con un coste asociado O(V * V' * L2 * L'2), donde V y V' son los números de nodos de T y T' respectivamente, y L y L' son los niveles de profundidad máximos de cada árbol.

4. Análisis algorítmico basado en información semántica

El algoritmo de Selkow encuentra el camino mínimo entre dos árboles de profundidad dos, y el de Kuo-Chung Tai, calcula el mínimo para árboles de cualquier profundidad. Tenemos, por tanto, una cota inferior del coste de un algoritmo de comparación de esquemas conceptuales. Podríamos pensar que utilizar el algoritmo de Kuo-Chung Tai para comparar EC es la solución al problema; sin embargo, ese algoritmo no fue pensado para comparar EC, sino árboles simples, y desaprovecha mucha información de los esquemas que pueden mejorar el resultado de la búsqueda.

En el marco de la comparación de esquemas conceptuales para la generación automática de planes de migración, se dispone de mucha información semántica que puede ser aprovechada por los algoritmos de comparación. Del mismo modo, la información que los algoritmos generan durante la comparación de dos elementos simples puede ser reutilizada en futuras comparaciones del resto de elementos del mismo esquema. En este apartado, vamos a analizar cómo se puede aprovechar toda esa información para desarrollar un algoritmo de comparación de esquemas conceptuales.

El concepto de distancia entre dos árboles, puede contraponerse a la medida de similitud entre árboles. En un contexto de evolución, dicha distancia puede servir para medir el grado de evolución que ha existido entre dos esquemas conceptuales, los cuales representaremos como árboles. No obstante, el cálculo del camino mínimo entre dos árboles, no es una buena solución para identificar la evolución sufrida por esquemas conceptuales; en la figura 2 se muestra un ejemplo muy simple que deja al descubierto el inconveniente que se plantea.

Una comparación de esquemas que busque el camino mínimo entre los dos árboles, daría como solución el cambio de nombre de la clase 'Coche' por 'Moto'; cuando realmente, la clase 'Coche'

ha sido eliminada, y la clase 'Moto' creada, puesto que se trata de clases distintas. En este caso, no debería existir una correspondencia entre coche y moto, puesto que no deben migrarse las instancias de uno a otro. El hecho de que la búsqueda de un conjunto mínimo de operaciones no sea adecuada para comparar esquemas conceptuales, se debe a que la evolución de esquemas conceptuales sigue patrones de evolución aleatorios, que no tienen porqué coincidir con caminos mínimos. Mientras que los algoritmos de optimización encuentran la forma más rápida de pasar de un árbol a otro, los algoritmos del siguiente apartado tratarán de encontrar métodos alternativos para comparar esquemas conceptuales que intenten identificar qué evolución real han tenido las instancias de las clases. Por otro lado, los algoritmos utilizados hasta el momento no tenían en cuenta información semántica del modelo, ya que es posible aprovechar las propiedades de los esquemas para mejorar el proceso de comparación.

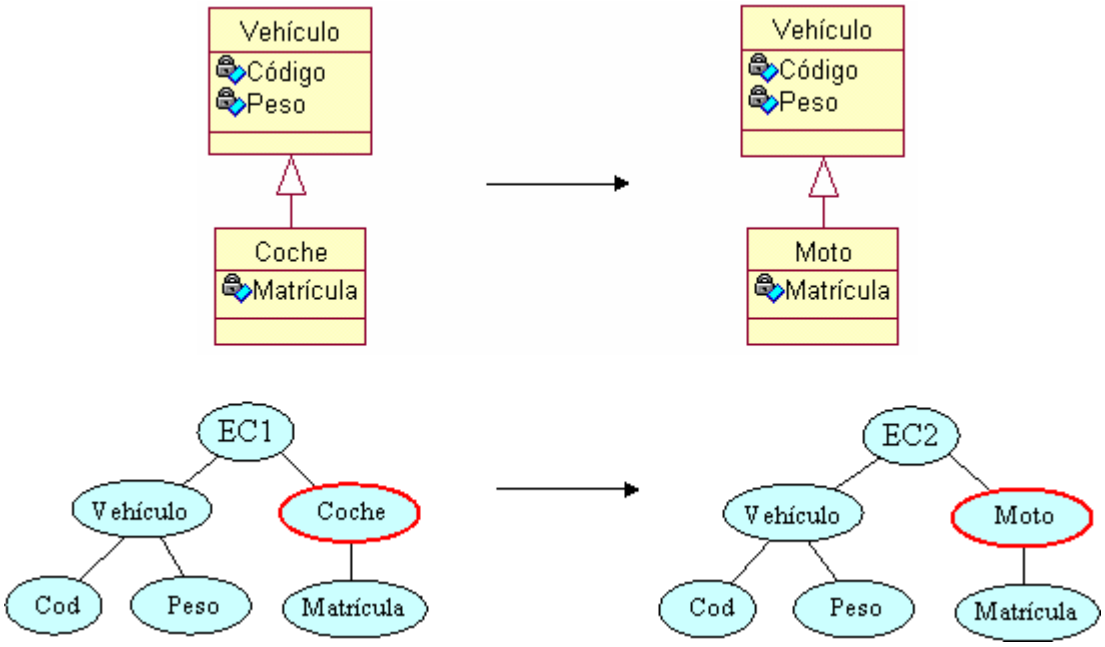


FIGURA 2: Comparación de esquemas conceptuales 'Árbol a Árbol'.

La aproximación 'Árbol a Árbol', puede ser adecuada cuando los esquemas conceptuales no tienen población asociada; pero en un contexto de evolución y migración de datos, los esquemas conceptuales están asociados a una población que debe evolucionar a la par que el esquema. En este tipo de contextos, la aproximación 'Árbol a Árbol' no suele ser adecuada porque no tiene en cuenta la evolución de población.

Para solucionar el problema, debemos buscar métodos alternativos de comparación, que tengan en cuenta la migración de población, y que traten de fijar como meta identificar qué hizo el analista con cada elemento de los esquemas cuando llevó a cabo la evolución. Necesitamos encontrar un algoritmo que compare los esquemas aprovechando, no sólo su estructura de árbol, sino también sus propiedades de esquemas conceptuales. Además, es muy importante que el algoritmo de comparación utilice criterios que traten de identificar qué evolución pretende el analista.

No hay que olvidar que una de las premisas iniciales era que ambos árboles debían estar etiquetados, la cuestión es ¿qué información contienen las etiquetas de los nodos? En un esquema conceptual la información disponible para comparar es muy rica (identificadores, nombres, relaciones con otros elementos, número de atributos en el caso de las clases, etc.) y dicha información puede determinar en gran medida el éxito de la comparación. En [Sil01] se analizan con detalle los diversos criterios de comparación de esquemas conceptuales.

El hecho de comparar dos esquemas conceptuales como si fueran sólo árboles desaprovecha mucha información semántica. Un algoritmo de comparación de esquemas conceptuales debe

aprovechar su estructura de árbol, pero también el hecho de que no es meramente un árbol, sino que se trata de un esquema conceptual. Esta idea puede aportar información muy rica a la hora de comparar. Los algoritmos del apartado anterior desaprovechaban esta información semántica y por ello trataban a todos los nodos del árbol de la misma forma. A priori, podríamos pensar que para comparar los elementos de los esquemas conceptuales deberemos comparar todos con todos, como hacían los anteriores algoritmos... ¿todos con todos? En un árbol-esquema conceptual, podemos distinguir cuatro tipos de ramas:

- Clases
- Atributos (subrama de una clase)
- Relaciones de Agregación
- Relaciones de Especialización

Es evidente, que no tiene sentido comparar clases con atributos, sino que habrá que comparar elementos de la misma naturaleza, esto es clases con clases, atributos con atributos, relaciones de agregación con relaciones de agregación y finalmente relaciones de especialización con relaciones de especialización.

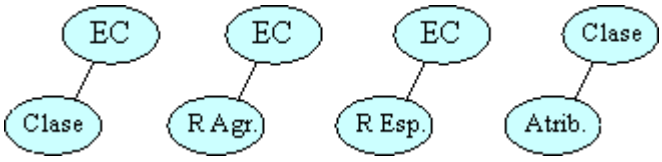
Partiendo de esta idea, podemos mejorar sustancialmente los algoritmos generales de comparación de árboles; para ello vamos a utilizar una técnica de fragmentación, con el fin de partir los árboles en subárboles comparables.

4.1 Mejora de la propuesta 'Árbol a Árbol' mediante fragmentación en subárboles

Dados dos árboles etiquetados cualesquiera, siendo V y V' sus números de nodos, y siendo L y L' sus respectivas profundidades, el algoritmo de Kuo-Chung Tai calcula el camino mínimo en términos de Inserciones, Modificaciones y Borrados con un coste:

Si representamos un esquema conceptual en forma de árbol, tal y como aparece en la figura 1, el coste aplicando el algoritmo de Kuo sería: Coste1 = $V \cdot V' \cdot 4 \cdot 4 = 16 \cdot V \cdot V' \in O(V \cdot V')$

Este coste puede mejorarse si aprovechamos la idea de que el árbol está representando un esquema conceptual, y cada subárbol tiene unas características distintas. Por lo tanto, si fragmentamos el árbol podríamos reducir L y L':



El coste sería: $Coste2 = (V1 \cdot V'1) + (V2 \cdot V'2) + (V3 \cdot V'3) + (V4 \cdot V'4) \in O(V \cdot V')$

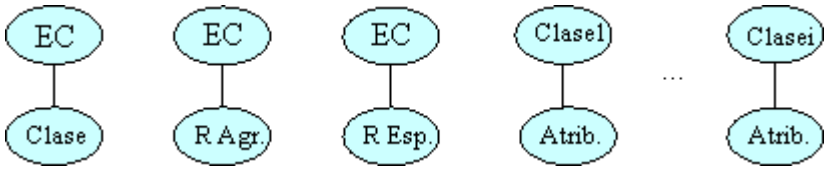
Como $V = V1 + V2 + V3 + V4$ y $V' = V'1 + V'2 + V'3 + V'4$

entonces

$Coste1 = V1 \cdot V'1 + \ldots + V1 \cdot V'4 + V2 \cdot V'1 + \ldots + V2 \cdot V'4 + \ldots + V4 \cdot V'1 + \ldots + V4 \cdot V'4 > Coste2$

Se demuestra que coste1 > coste2.

Podemos mejorar el coste todavía más si continuamos fragmentando los subárboles:



El coste sería:

Coste3 = (V1·V'1) + (V2·V'2) + (VC1·VC'1) + … + (VCn·VC'n) $\in$ O(V·V') < Coste2

Se demuestra que coste1 > coste2 > coste3.

Otro tipo de información semántica que conviene tener en cuenta es el orden de comparación. No tiene sentido comparar atributos, sin antes haber identificado las clases (*nodos padres en el árbol*) que contienen esos atributos. La comparación de esquemas conceptuales ha de seguir un orden lógico, dicho orden no es un orden prefijado, sino que depende del criterio de comparación que vaya a aplicarse. Por ejemplo, supongamos que queremos comparar dos esquemas con un criterio de comparación basado en OID, el orden lógico de comparación sería el siguiente:

- Análisis de clases (parte 1).
- Análisis de atributos.
- Análisis de clases (parte 2).
- Análisis de relaciones.

El primer paso será identificar qué clases del esquema conceptual inicial han sido borradas, y qué clases del final han sido insertadas; el resto de clases consideraremos que permanecen invariantes (*Pueden haber sido modificadas, pero son clases correspondientes*). Este es a lo que llamamos el análisis de clases (parte 1). A continuación, procederemos a analizar los atributos de cada clase invariante (sólo de las invariantes, porque los atributos de las inserciones son todos nuevos, y los de los borrados han sido todos eliminados) para identificar los insertados, modificaciones, borrados e invariantes. Esta parte constituye el análisis de atributos. A partir del análisis de atributos podremos determinar qué clases han sufrido modificaciones, y cuáles son realmente invariantes. Finalmente podremos analizar las relaciones entre clases, puesto que ya tendremos totalmente identificadas las mismas.

Cuando aplicamos la información semántica anterior, nos damos cuenta de que ya no tenemos un árbol que comparar, sino muchos de profundidad uno. Podríamos pensar que el algoritmo de Selkow se ajusta perfectamente al problema, pero no es así. Dicho algoritmo tiene en cuenta a la hora de comparar el orden de los nodos del árbol, y este hecho a nosotros nos es totalmente indiferente; esto nos permite relajar todavía más las restricciones originales.

A continuación, vamos a presentar dos versiones del algoritmo que resuelve el problema:

Partimos de la siguiente información:

A, B: Listas con los elementos del árbol T y T' respectivamente.

Criterio(Ai, Bj)**:** Devuelve CIERTO si Ai y Bj son elementos iguales según algún criterio de comparación preestablecido.

Emparejar(Ai, Bj)**:** Introduce Ai y Bj en una lista de parejas de elementos, y los saca de las respectivas listas de nodos A y B.

Borrado(Pa): Inserta en la lista de elementos borrados el elemento Pa.

Inserción(Pa): Inserta en la lista de elementos insertados el elemento Pa.

Ordenar(A): Ordena la lista A.

Algoritmo (versión 1):

```
Procedure (A, B);
(Asumimos que A y B tienen como
subárboles A1,...,Am; B1,...,Bn respectivamente)
For i = 1, 2, …, m do
    For j = 1, 2, …, n do
    Begin
        If Criterio(Ai, Bj) = CIERTO then
            Emparejar(Ai, Bj);
            Exit;      /* Sale del bucle interno */
         End if;
    End;
End Procedure.
```

Algoritmo (versión 2):

```
Procedure (A, B);
(Asumimos que A y B tienen como
subárboles A1,...,Am; B1,...,Bn respectivamente)
Ordenar(A);
Ordenar(B);
Pa = Ai;  Pb = Bj;   /* Punteros a los elementos de los vectores */
While((Pa < Length(Ai)) and ( Pb < Length(Bj))
    If Criterio(Pa, Pb) = CIERTO then
        Emparejar(Pa, Pb);
        Pa = Pa+1;  Pb = Pb+1;
    Else If Pa < Pb then
        Borrado(Pa);
```

$P_a = P_a+1;$

Else If $P_a > P_b$ **then**

Inserción(Pb);

$P_b = P_b+1;$

End if;

End While;

/* Completar resto de la lista en inserciones o borrados */

End Procedure.

Al finalizar el algoritmo dispondremos de tres listas. La primera lista contendrá todos los elementos que han sido eliminados del esquema conceptual inicial; la segunda lista contendrá todos los elementos que han sido insertados en el esquema conceptual final; y la tercera lista contendrá todos los elementos que se mantienen tras la evolución entre los dos esquemas. Es fácil darse cuenta de que el algoritmo es totalmente dependiente del criterio de comparación elegido, lo cual es lógico, porque el criterio adoptado determinará cuándo dos elementos son iguales; Este hecho es muy importante, dado que no todos los criterios son igual de buenos, ni existe un criterio que sea siempre mejor que los demás.

La primera versión del algoritmo tiene un coste asociado O(m*n) (*Estos costes no incluyen el coste asociado al criterio de comparación*); dicho coste es mejorado por la segunda versión. Ha sido presentado porque en él se ve claramente la dependencia del coste respecto del criterio de comparación utilizado. En cualquier caso, siempre debemos tener en cuenta que el número de elementos a comparar no superará el centenar en la mayoría de los casos, con lo que la diferencia de coste se ve sustancialmente disminuida. La segunda versión del algoritmo tiene un coste asociado O(n*log(n)) (*Realmente, el coste sería n*log(n) + m*log(m) + n + m : las dos primeras componentes se derivan de la ordenación de las listas (utilizando por ejemplo 'mergesort' o 'quiksort'); la tercera y cuarta componentes del coste proceden del recorrido de las listas.*), dado que ordena las listas antes de su tratamiento.

Es importante hacer notar que el criterio de comparación puede tener una complejidad elevada si utiliza la información que va obteniendo el algoritmo durante la comparación. Aunque este tipo de criterios presenta mayores costes y su complejidad ciclomática es de un orden superior (debido a su interdependencia con el propio algoritmo de comparación), este será en la mayoría de los casos el criterio de comparación apropiado, puesto que por termino medio, presenta una mejoría sustancial en sus tasas de evaluación. En [Sil01] puede encontrarse un análisis completo sobre este tipo de criterios.

El algoritmo propuesto es el núcleo de una herramienta de comparación automática de esquemas conceptuales que ha sido desarrollada para un entorno de migración automática de bases de datos. En ese contexto, la herramienta es capaz de identificar automáticamente los orígenes y los destinos de la migración.

5. Conclusiones

La comparación de esquemas conceptuales es un proceso mucho más rico que la comparación simple de árboles. Los algoritmos optimizados de comparación de árboles pueden utilizarse para comparar esquemas; pero también pueden mejorarse utilizando información semántica:

- Existen cuatro tipos de elementos comparables distintos en un esquema conceptual. Sólo

deben compararse elementos pertenecientes a la misma clase de equivalencia.
- Cada elemento de un esquema conceptual es único y está perfectamente identificado: La primera comparación exitosa del algoritmo podrá cortar la búsqueda de ese elemento.
- A diferencia de la comparación simple de árboles, los algoritmos de comparación de esquemas conceptuales pueden aprovechar dinámicamente la información semántica que van generando durante el propio proceso de comparación.
- El algoritmo de comparación debe estar regido según un orden lógico de comparación.
- El criterio de comparación determinará en gran medida el éxito final de la comparación.

El objetivo final de la comparación no es tanto la búsqueda de caminos mínimos, como la identificación de la evolución de cada elemento. Finalmente, la comparación de esquemas conceptuales no es un problema de optimización.

6. Bibliografía

[San72] D. Sankoff, Matching sequences under deletion/insertion constraints, Proc. Nat. Acad. Sci. 1972, 4-6.

[Wag74] R.A.Wagner y M.J. Fisher, The string to string correction problem, J. Assoc. Comput. 1974, 13-16.

[Won76] C.K. Wong y A.K. Chandra, Bounds for the string editing problem, J. Assoc. Comput. 1976, 168-173.

[Aho76] A.V. Aho, D.S. Hirschberg y J.D. Ullman, Bounds on the complexity of the longest common subsecuence problem, J. Assoc. Comput. 1976, 1-12.

[Sel77] S. Selkow, The tree-to-tree editing problem. Dep. Com. Sci., University of Tennessee, Knoxville, 1977.

[Kuo79] Kuo-Chung Tai, The tree-to-tree correction problem, Dep. Com. Sci., North Carolina State University, Raleigh, 1979.

[OOM98] Ramos I., Pastor O., OO-Method: Una metodología OO para la producción automática de software., DSIC, UPV, 1998.

[Gra98] Grau A., Computer-Aided Validation of formal conceptual models, Tesis Doctoral, Institute For Software, Information Systems Group, Technical University of Braunschweig, 1998.

[OAS98] Letelier P., Ramos I., Sánchez P. Pastor O., Oasis 3.0: Un enfoque formal para el modelado conceptual orientado a objeto, SPUPV-98.4011, 1998.

[Ana99] Anaya R., Desarrollo y gestión de componentes reutilizables en el marco de Oasis, Tesis Doctoral, Facultad de Informática, Universidad Politécnica de Valencia, Octubre 1999.

[Car99] Carsí J.A., OASIS como marco conceptual para la evolución del software, Tesis Doctoral, DSIC, UPV, 1999.

[Mor00] Moreno M., García F., Polo M., Medición de la calidad del software en el ámbito de la especificación de requisitos , Universidad de Salamanca, Departamento de informática y automática 2000.

[Sta00] Staudt B., A model for compound type changes encountered in schema evolution, University of Massachusetts, Amherst, ACM Transactions on Database Systems, Vol. 25, No. 1, March 2000, Pages 83–127.

[Dol00] Dolado J.J., Fernández L., Medición para la gestión en la ingeniería del software. RA-MA, 2000.

[Var00] Varas M., Pradenas J., Hacia la definición de métricas de calidad para esquemas conceptuales de bases de datos., revista electrónica del DIICC, edición nº 6, 2000.

[Sil01] Silva J., Ramos I., Carsí J.A., Análisis teórico/experimental de criterios de comparación de esquemas conceptuales a efectos de la generación automática de planes de migración, enviado a las JISBD,2001.

[Per01] Pérez J., Anaya V., Silva J., Carsí J.A, Ramos I., Generación Automática de un plan de migración entre poblaciones de esquemas conceptuales orientados a objetos, Distributed Objects Languages Methods & Environments, DOLMEN, Sevilla 2001.

[Cal01] Calero C., Definición de un conjunto de métricas para la mantenibilidad de bases de datos relacionales, activas y objeto-relacionales, Tesis Doctoral, Universidad de Castilla-La Mancha, 2001.