Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche

researchportal.unamur.be

University of Namur

UNIVERSITÉ DE NAMUR

RESEARCH OUTPUTS / RÉSULTATS DE RECHERCHE

Stratégies pour l'évolution des applications de bases de données relationnelles : l'approche DB-MAIN

Hick, Jean-Marc; Hainaut, Jean-Luc; Englebert, Vincent; Roland, Didier; Henrard, Jean

Published in:
XVIIe congrès INFORSID

Publication date:
1999

Link to publication

Citation for pulished version (HARVARD):
Hick, J-M, Hainaut, J-L, Englebert, V, Roland, D & Henrard, J 1999, Stratégies pour l'évolution des applications de bases de données relationnelles : l'approche DB-MAIN. dans *XVIIe congrès INFORSID.* INFORSID, La garde, pp. 33-54.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 18. Mar. 2025

Stratégies pour l'évolution des applications de bases de données relationnelles : l'approche DB-MAIN

Jean-Marc Hick, Jean-Luc Hainaut, Vincent Englebert, Didier Roland, Jean Henrard

Institut d'Informatique - Facultés Universitaires de Namur
Rue Grandgagnage 21, B-5000 Namur, Belgique

E-mail : jmh@info.fundp.ac.be - Tél. : +32 81 72 49 85 - Fax: +32 81 72 49 67

Résumé

Si les nouvelles technologies en matière de SGBD envisagent le problème de l'évolution du schéma d'une base de données, les systèmes d'information actuels posent des problèmes particulièrement ardus lors des phases d'évolution. Cet article étudie ces problèmes dans un contexte pratique tel qu'il est vécu actuellement par les développeurs. Il propose une analyse du phénomène de l'évolution et son impact sur les structures de données, les données et les programmes d'application au travers de stratégies typiques. L'article décrit ensuite l'environnement de génie logiciel DB-MAIN avec lequel a été construit un prototype d'outil d'aide à l'évolution. Cet outil permet notamment de générer automatiquement les programmes de conversion de la base de données à partir de la trace des opérations de modification des schémas conceptuel ou logique de la base de données. Il permet aussi d'aider le programmeur à modifier les programmes d'application en conséquence.

Mots-clés

Evolution, conversion de bases de données, transformation de schéma, traçabilité, rétro-ingénierie, AGL.

Abstract

If the recent DBMS technologies consider the problem of databases schema evolution, standard Information Systems in use raise hard problems when evolution is concerned. This paper studies these problems in the current developer context. It analyzes the evolution of systems and its impact on the data structures, data and programs through typical strategies. The paper introduces the DB-MAIN CASE environment with which an evolution tool prototype has been developed. This tool can automatically generate the conversion programs for the database from the operational trace of the conceptual and logical schema modifications. It can also help the programmer to modify the application programs.

Keywords

Evolution, databases conversion, schema transformation, history, reverse ingeneering, CASE tools.

1 Introduction

On désignera par le terme *application de bases de données* un système logiciel dont le composant *données persistantes* (ou base de données) est central. Au cours de leur cycle de vie, ces applications sont amenées à évoluer. Cette évolution provient de changements dans les besoins de l'entreprise ou de l'apparition de nouveaux besoins. La modification des besoins peut être de trois types : fonctionnelle (satisfaire les exigences des utilisateurs du système en termes de fonction de celui-ci), organisationnelle (répondre au changement du cadre de travail de l'entreprise) et technique (adaptation aux nouvelles contraintes techniques ou matérielles). Suite à cette évolution des besoins, le concepteur apporte des modifications techniques à l'application.

Confronté au problème de l'évolution, le concepteur est démuni tant du point de vue méthodologique que du point de vue technique. En effet, on ne dispose pas à l'heure actuelle de règles systématiques de traduction des modifications des besoins des utilisateurs en modifications des composants techniques de l'application. On ne dispose même pas de recommandations méthodologiques garantissant des applications stables, robustes et faciles à maintenir. Au niveau des outils, le développeur se voit offrir des ateliers de génie logiciel qui ignorent pour la plupart les processus de maintenance et d'évolution. En ce qui concerne les bases de données, ces outils permettent de construire un schéma conceptuel, de le transformer de manière automatique en schéma logique et de générer les structures de la base de données. Le code généré doit souvent être remanié de manière significative pour devenir véritablement opérationnel et toute modification des spécifications entraînera la modification du schéma conceptuel, la transformation en un nouveau schéma logique et la production d'un nouveau code. Ce nouveau composant est sans lien formel avec la version précédente. La conversion des données et la modification des programmes est entièrement à la charge du développeur.

Le problème de l'évolution de bases de données a d'abord été analysé pour des structures de données standards. La modification d'un schéma relationnel a été étudiée entre autres par [SHNE82], [ANDA91] ou [RODD93]. Le paradigme objet a permis de développer des solutions élégantes grâce au contrôle des versions de schémas et des versions d'instances ([NGUY89], [BELL93], [ALJA95]). [VANB94] et [RODD93] ont également étudié l'impact des modifications conceptuelles sur les schémas relationnels. [VERE97] analyse les facteurs qui influencent les impacts des modifications sur le système existant en termes de maintenabilité et de capacité à évoluer.

Certains projets de recherche se sont préoccupés du problème de la gestion des modifications dans des systèmes d'information. Citons, par exemple, le projet NATURE dans le cadre duquel un environnement d'ingénierie des besoins jouant un rôle dans la gestion des modifications a été développé [JARK94], ainsi que le projet SEI-CMU qui s'intéresse entre autres à l'évaluation de la capacité d'évoluer de systèmes d'information existants sur base d'outils d'extraction d'informations du code source et d'une stratégie globale d'évolution [BROW96].

L'objectif de cet article est d'analyser le phénomène de l'évolution du composant *base de données* des applications au travers de stratégies liées au niveau d'abstraction des besoins dont la modification induit cette évolution. Cet article présente un modèle abstrait et un environnement d'assistance relatifs à l'évolution des applications de bases de données, considérées comme l'intégration de trois composants : les structures de données, les données et les programmes.

L'article est organisé en quatre parties. La première (section 2) positionne le problème et détermine les frontières du champ d'application de l'approche adoptée. La deuxième partie (section 3) présente les trois paradigmes sur lesquels l'approche est fondée. La troisième partie (section 4) décrit les stratégies

répondant à des problèmes de modification des spécifications. Finalement, la quatrième partie (section 5) est dédiée à l'approche DB-Main de la maintenance d'application de bases de données, montrant comment les bases méthodologiques peuvent supporter les stratégies.

2 Description du problème

On considère l'analyse du phénomène de l'évolution dans le cadre de la modélisation classique qui envisage la conception d'une base de données comme une activité complexe décomposable en processus plus élémentaires qui vont de la collecte des informations concernant les besoins des utilisateurs jusqu'à la génération de code. Selon l'approche classique, que nous adopterons ici, car elle est familière aux développeurs, on définit trois niveaux d'abstraction correspondant chacun à des spécifications intégrant une famille de critères de conception bien définie. Il s'agit des niveaux conceptuel, logique et physique. Dans la figure 1, le schéma conceptuel répond aux besoins R1 (fonctionnels et organisationnels), le schéma logique aux besoins R2 (organisationnels et techniques) et le schéma physique aux besoins R3 (techniques). La partie opérationnelle contient la base de données (données et structures de données) et les programmes.

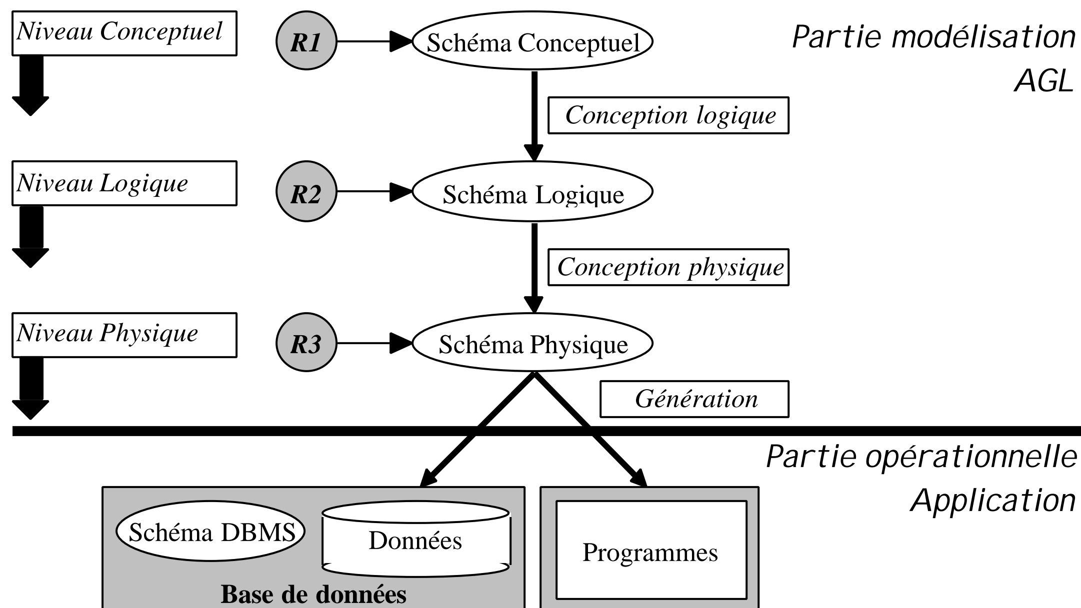


Figure 1 *- Modélisation classique définie selon trois niveaux d'abstraction.*

Nous adopterons l'hypothèse selon laquelle l'analyste dispose des spécifications de l'application à ces différents niveaux d'abstraction, et que la trace des processus d'élaboration de ces spécifications est également disponible. Cette hypothèse étant irréaliste dans la plupart des situations (il n'est pas rare que le code source des programmes constitue la seule documentation fiable de l'application), nous serons amenés à compléter les stratégies d'évolution pour tenir compte de leur absence.

Au cours de son cycle de vie, une application de base de données subit des évolutions. Celles-ci traduisent l'apparition de nouveaux besoins ou la modification de besoins existants. Pour amener cette application à satisfaire ces nouveaux besoins, le concepteur va modifier certains de ces composants.

Dans le cadre de cet article, nous limiterons le propos au composant *données persistantes* de l'application, c'est-à-dire à l'ensemble des fichiers, ou base de données, qui représentent essentiellement les objets remarquables du domaine d'application.

Le problème peut alors se reformuler de la manière suivante :

- suite à l'évolution des besoins des utilisateurs, l'analyste est amené à modifier les spécifications de la base de données à un certain niveau d'abstraction;
- comment gérer le processus d'évolution de l'application complète suite à cette modification des spécifications ?

Remarque : il est important de noter que la discussion se situe dans le cadre d'un scénario particulier, quoique particulièrement fréquent, selon lequel tous les composants de l'application sont remplacés par une nouvelle version, et que l'application nouvelle se substitue complètement à l'ancienne. Sauf procédure de sauvegarde des anciens composants, ignorée ici, il n'est plus possible d'accéder à la version précédente de l'application, et en particulier aux données qui n'auraient pas été reprises dans la nouvelle version. Autrement dit, on s'écarte des architectures avancées, dans lesquelles la modification du schéma de la base de données peut se traduire par l'ajout d'une nouvelle version de celui-ci avec conservation du schéma ancien et de l'accès à ses données, mémorisées ou recalculées (schema/data versioning) ([JENS94], [RODD93]). Concrètement, l'ajout d'une nouvelle propriété à une classe d'objets conceptuelle se traduira, dans une base relationnelle, par la modification de celle-ci via une requête du type "`alter table <table name> add <column spec>`".

3 Les bases méthodologiques

Selon l'hypothèse de travail énoncée plus haut, nous disposons donc des spécifications aux différents niveaux d'abstraction. La modification des besoins se traduisant, suivant des règles ignorées ici, en modification des spécifications à un certain niveau, le problème consiste alors à rendre cohérentes toutes les spécifications, et donc à propager les modifications vers les autres niveaux d'abstraction, tant en amont qu'en aval.

Dans ce contexte, un environnement de génie logiciel, ou AGL, à large spectre, destiné à assister le concepteur dans ses activités de maintenance et d'évolution d'une application de base de données doit satisfaire les exigences suivantes :

1. **Généricité** : l'environnement doit offrir un modèle générique de représentation des spécifications quel que soit le niveau d'abstraction, les technologies ou les paradigmes de modélisation utilisés.
2. **Formalité** : l'environnement doit pouvoir décrire de manière précise et rigoureuse toutes les activités d'ingénierie des bases de données. Toute opération exécutée sur une spécification, telle que la dérivation ou la modification, doit pouvoir être décrite de manière formelle.
3. **Traçabilité** : les relations entre les spécifications de différents niveaux, ainsi qu'entre les différentes versions successives de celles-ci, doivent être formalisées. Elles doivent pouvoir être analysées et manipulées de manière à fournir les informations nécessaires à la propagation des modifications.

Les trois points suivants présenteront les concepts de base sur lesquels l'approche DB-MAIN a été élaborée, notamment :

- un modèle générique permettant de représenter des spécifications multi-niveaux et multi-paradigmes;
- une approche transformationnelle de l'ingénierie de bases de données, y compris de la rétro-ingénierie;
- le concept d'historique des opérations effectuées sur des spécifications.

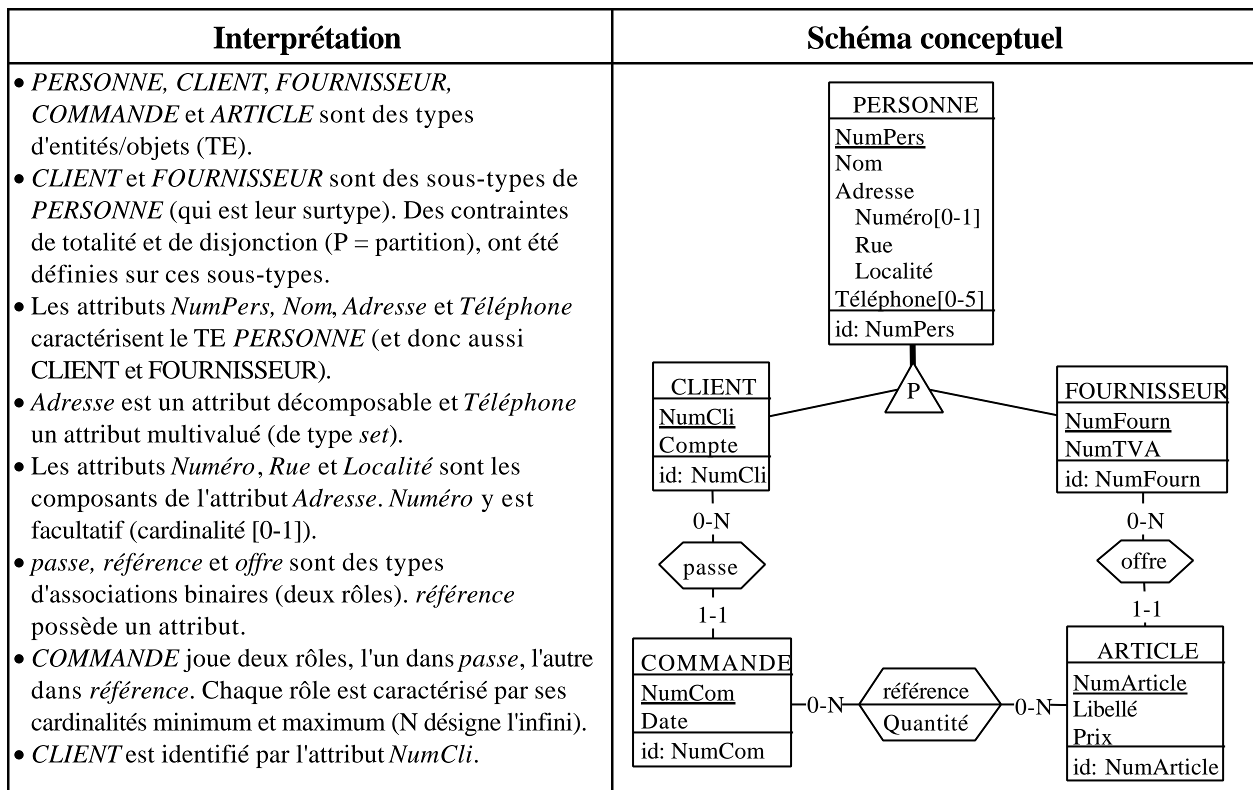
3.1 Modèle générique de représentation des spécifications

Le modèle utilisé dans l'approche proposée présente des caractéristiques de généricité selon deux dimensions. Il permet :

- de représenter des spécifications aux différents niveaux d'abstraction, et notamment des spécifications conceptuelles, logiques et physiques;
- de couvrir, à chaque niveau, les principaux paradigmes de modélisation et technologies tels que les modèles Entité/Association, NIAM, à Objets, relationnels, CODASYL, IMS, fichiers, etc.

Il est basé sur un modèle Entité/Objet-Association générique qui permet de définir, par spécialisation, les différents modèles propres à une démarche méthodologique particulière. Nous illustrerons son application à la description des trois schémas qui apparaissent lors de l'élaboration d'une petite base de données relationnelle.

Chacun de ces modèles est défini comme un sous-modèle du modèle générique. Un sous-modèle M est obtenu par spécialisation[1] du modèle générique, c'est-à-dire par une restriction du modèle générique selon laquelle on sélectionne les objets pertinents et on précise les lois définissant les assemblages licites dans M, ainsi que par un renommage des objets selon la nomenclature propre à M, éventuellement accompagné de conventions graphiques appropriées pour la représentation des objets. Les schémas des figures 2 à 4 sont exprimés dans trois sous-modèles classiques : une variante du modèle entité-relation (figure 2), le modèle logique relationnel (figure 3) et le modèle Oracle 7 (figure 4).



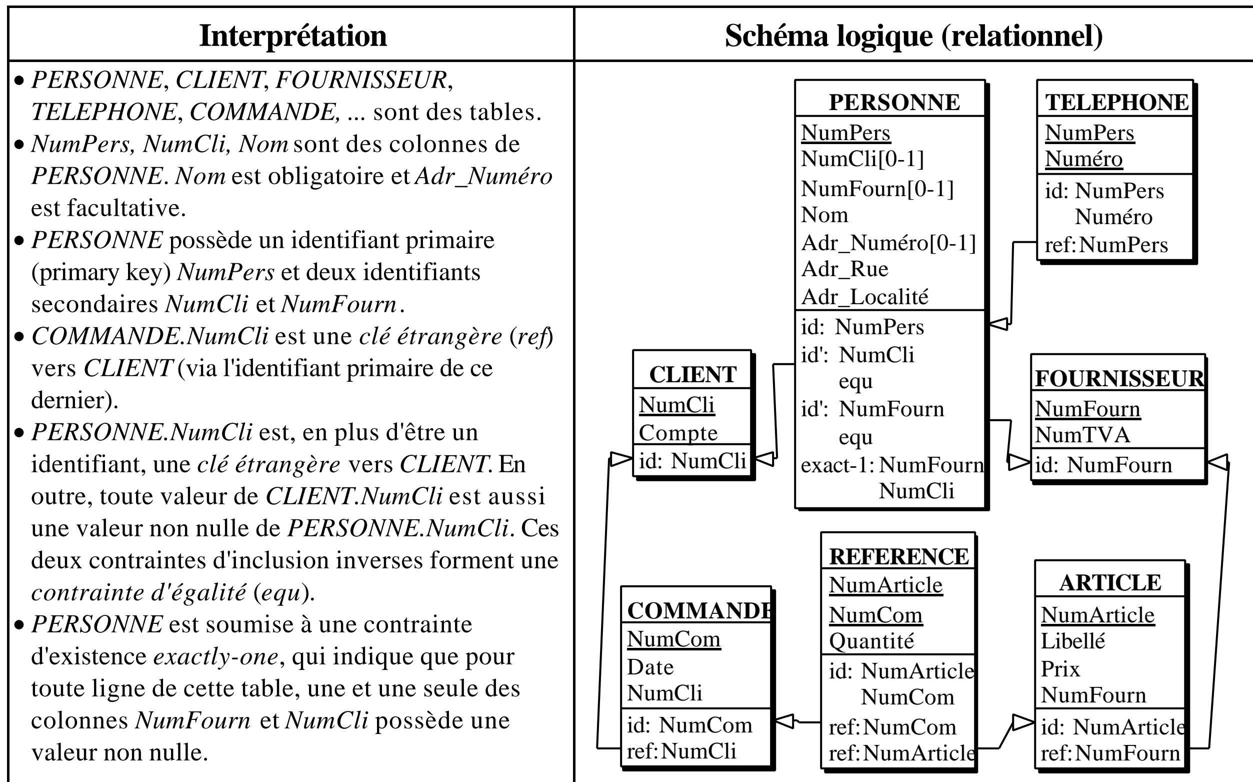
Interprétation

- *PERSONNE, CLIENT, FOURNISSEUR, COMMANDE* et *ARTICLE* sont des types d'entités/objets (TE).
- *CLIENT* et *FOURNISSEUR* sont des sous-types de *PERSONNE* (qui est leur surtype). Des contraintes de totalité et de disjonction (P = partition), ont été définies sur ces sous-types.
- Les attributs *NumPers, Nom, Adresse* et *Téléphone* caractérisent le TE *PERSONNE* (et donc aussi CLIENT et FOURNISSEUR).
- *Adresse* est un attribut décomposable et *Téléphone* un attribut multivalué (de type *set*).
- Les attributs *Numéro*, *Rue* et *Localité* sont les composants de l'attribut *Adresse*. *Numéro* y est facultatif (cardinalité [0-1]).
- *passe, référence* et *offre* sont des types d'associations binaires (deux rôles). *référence* possède un attribut.
- *COMMANDE* joue deux rôles, l'un dans *passe*, l'autre dans *référence*. Chaque rôle est caractérisé par ses cardinalités minimum et maximum (N désigne l'infini).
- *CLIENT* est identifié par l'attribut *NumCli*.

Schéma conceptuel

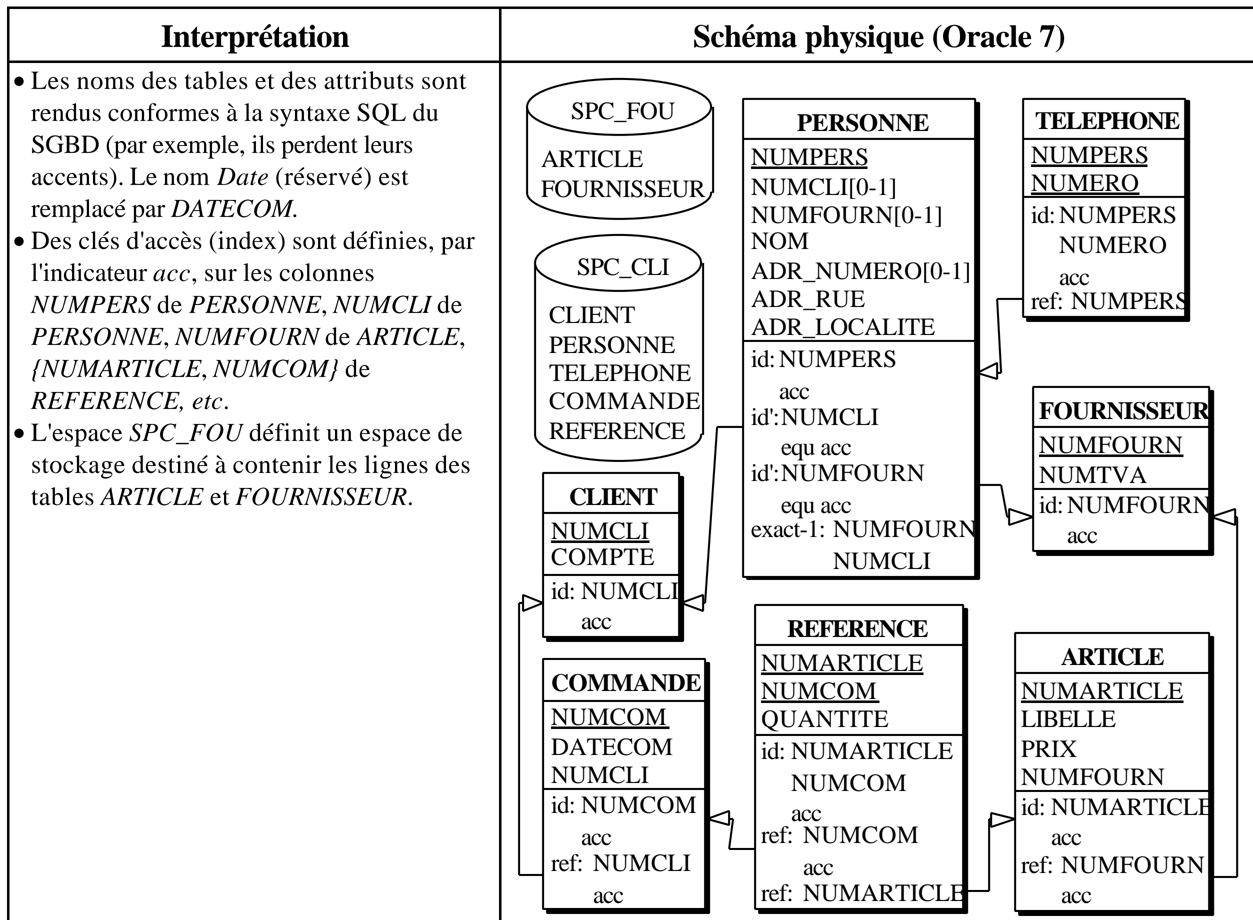
Figure 2 *- Vue graphique et interprétation d'un schéma conceptuel typique.*

[1] Le terme de *spécialisation* peut apparaître comme impropre au sens traditionnel qui lui est généralement assigné. Etant donné un modèle G et un modèle M, déclaré spécialisation de G, tout schéma conforme à M est également conforme à G.



- *PERSONNE*, *CLIENT*, *FOURNISSEUR*, *TELEPHONE*, *COMMANDE*, ... sont des tables.
- *NumPers, NumCli, Nom* sont des colonnes de *PERSONNE*. *Nom* est obligatoire et *Adr_Numéro* est facultative.
- *PERSONNE* possède un identifiant primaire (primary key) *NumPers* et deux identifiants secondaires *NumCli* et *NumFourn*.
- *COMMANDE.NumCli* est une *clé étrangère* (*ref*) vers *CLIENT* (via l'identifiant primaire de ce dernier).
- *PERSONNE.NumCli* est, en plus d'être un identifiant, une *clé étrangère* vers *CLIENT*. En outre, toute valeur de *CLIENT.NumCli* est aussi une valeur non nulle de *PERSONNE.NumCli*. Ces deux contraintes d'inclusion inverses forment une *contrainte d'égalité* (*equ*).
- *PERSONNE* est soumise à une contrainte d'existence *exactly-one*, qui indique que pour toute ligne de cette table, une et une seule des colonnes *NumFourn* et *NumCli* possède une valeur non nulle.

Figure 3 *- Vue graphique et interprétation d'un schéma logique relationnel.*



- Les noms des tables et des attributs sont rendus conformes à la syntaxe SQL du SGBD (par exemple, ils perdent leurs accents). Le nom *Date* (réservé) est remplacé par *DATECOM.*
- Des clés d'accès (index) sont définies, par l'indicateur *acc*, sur les colonnes *NUMPERS* de *PERSONNE*, *NUMCLI* de *PERSONNE*, *NUMFOURN* de *ARTICLE*, *{NUMARTICLE*, *NUMCOM}* de *REFERENCE, etc*.
- L'espace *SPC_FOU* définit un espace de stockage destiné à contenir les lignes des tables *ARTICLE* et *FOURNISSEUR*.

Figure 4 *- Vue graphique et interprétation d'un schéma physique relationnel (Oracle 7).*

3.2 Approche transformationnelle

Dans le domaine des bases de données, tout processus d'ingénierie peut être défini comme une suite de modifications de structures de données [BATI92]. Par exemple, la conception logique sera modélisée comme une chaîne de transformations du schéma conceptuel pour que le résultat final soit conforme au modèle relationnel et satisfasse à des critères de performance. De même, la construction d'un schéma conceptuel, la normalisation d'un schéma conceptuel, l'optimisation d'un schéma logique, la génération de code SQL ou la rétro-ingénierie de fichiers COBOL peuvent être perçues comme des séquences de transformations. L'ajout d'un type d'entités, la modification du nom d'un attribut ou le changement d'un type d'associations en clés étrangères sont des transformations élémentaires qu'on peut combiner de manière à former des processus plus complexes. Cette section présente brièvement la notion de transformation. Nous nous limiterons ici à une présentation superficielle des transformations, suffisante toutefois pour la compréhension de l'approche proposée. Le lecteur en trouvera une définition plus précise dans [HAIN96].

Une transformation est un opérateur qui substitue à un ensemble (éventuellement vide) d'objets C d'un schéma un autre ensemble d'objets (éventuellement vide) C'. Cet opérateur est défini par les préconditions minimales que C doit satisfaire avant l'opération et les postconditions maximales que C' vérifie après transformation. Si **T** est une transformation, C et C' des structures de données, dont l'une peut être vide, on peut écrire : C' = T(C). Pour être complète, la définition d'une transformation doit aussi inclure la fonction **t** de transformation des instances de C en instances de C' (figure 5).

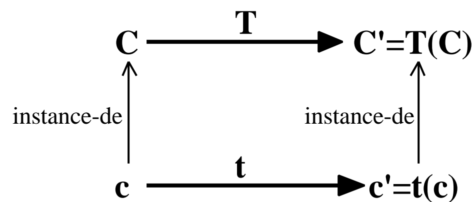


Figure 5 *- Schéma d'une transformation.*

Certaines transformations augmentent la sémantique d'un schéma (*ajouter un type d'entités*) et sont dites appartenir à la catégorie T+, d'autres relevant de la catégorie T-, la diminuent (*supprimer un attribut*). Il existe aussi des transformations qui préservent la sémantique d'un schéma, dites de catégorie T=, et qui sont telles que C et T(C) décrivent le même univers du discours (*remplacer un type d'associations par un type d'entités*). Ces dernières transformations sont appelées *réversibles*. Une transformation <T,t> est réversible s'il existe une transformation <T',t'> telle que : T'(T(C)) = C et t'(t(c)) = c pour toute instance c de C. Si <T',t'> est également réversible, on dit que ces transformations sont *symétriquement réversibles*.

Bien que souvent citée, la notion de *sémantique d'un schéma* n'a pas reçu de définition généralement acceptée. Nous dirons que la sémantique du schéma S1 inclut celle du schéma S2 si le domaine d'application représenté par S2 est une partie de celui que représente S1. Par conséquent, ajouter un type d'objets à un schéma en augmente la sémantique tandis que l'ajout d'une contrainte la diminue. Ce concept n'étant pas critique pour l'exposé, nous nous en tiendrons à cette définition intuitive.

Les transformations T= sont principalement utilisées dans la production de schémas logiques et physiques, tandis que les transformations T+ et T- forment la base du processus d'évolution des spécifications. Il existe un grand nombre de transformations de schéma. On peut citer par exemple la désagrégation d'un attribut composé, l'agrégation d'attributs, la transformation d'attributs multivalués en listes d'attributs monovalués, la fusion et l'éclatement de types d'entités, etc.

3.3 Notion d'historique

Comme nous l'avons vu au point 3.2, tout processus d'ingénierie est modélisable comme une séquence de transformations de spécifications. Dans le contexte de l'évolution, où les modifications apportées à un niveau d'abstraction doivent être propagées dans les autres niveaux, garder une trace des transformations est une nécessité si le concepteur veut éviter de reformuler les séquences de transformations lors de chaque modification.

La trace des opérations de transformation qui ont produit une spécification Sj à partir d'une spécification Si est appelée *historique* (Hij) du processus de transformation. Hij étant lui-même une macro-transformation, on utilise la notation fonctionnelle Sj = Hij(Si). Le lecteur intéressé par ces concepts est renvoyé à [HAIN96a].

En exécutant les transformations enregistrées dans Hij sur Si, on dit que Hij est rejoué sur le schéma Si. La correspondance entre les objets de Si et de Hij se fait sur base de leur nom et de leur type. Si un objet traité dans Hij n'est pas présent dans Si, la transformation est ignorée. Il est également possible de dériver de Hij la fonction inverse Hji telle que Si = Hji(Sj). Hji peut être obtenue en substituant à chaque opération d'origine son inverse, puis en inversant l'ordre des opérations. Un historique étant de nature procédurale, il ne peut être manipulé aisément que s'il respecte certaines règles :

1. **Exhaustivité** : toute opération est enregistrée dans l'historique de manière précise et complète pour permettre son inversion, ce qui est essentiel pour les transformations qui ne conservent pas la sémantique. Par exemple, l'enregistrement d'une suppression d'un type d'entités nécessite l'enregistrement de ses attributs, groupes, rôles et relations de sous-typage.
2. **Normalisation** : l'historique est monotone (pas de retours en arrière) et linéaire (pas de branchements multiples).
3. **Non-concurrence** : un historique est attaché à un schéma qui ne peut être modifié que par une seule personne à la fois.

4 Stratégies d'évolution

Pour développer une méthodologie débouchant sur des réalisations concrètes, la démarche se base sur la technologie relationnelle pour dégager une typologie des modifications et permettre ainsi de concevoir des outils de conversion pour la partie opérationnelle. Elle est donc applicable à des systèmes développés à l'aide de langages standards de troisième génération tels que COBOL/SQL ou C/SQL. Cette limite n'enlève rien à la démarche qui pourrait être mise en oeuvre avec d'autres technologies d'implémentation des applications moyennant des typologies adaptées pour les modifications et des outils de conversion spécifiques pour la couche physique.

Le problème de la propagation est d'autant plus complexe qu'il peut y avoir plusieurs contextes d'évolution. En effet, idéalement, une application de bases de données s'inscrit dans un contexte où les trois niveaux sont présents et documentés. Cette hypothèse est généralement irréaliste. Certains niveaux peuvent être absents ou incomplets et, dans les cas les plus graves, seuls les programmes et les données sont disponibles. Pour pallier ces lacunes, il faut d'abord reconstruire la documentation d'une application grâce à des techniques de rétro-ingénierie.

Dans cette section, nous allons d'abord présenter brièvement le problème de la reconstruction des spécifications. Ensuite trois stratégies de référence (une par niveau d'abstraction) répondant chacune à des problèmes de modification des spécifications à un niveau d'abstraction seront analysées. Nous

limiterons l'étude de ces stratégies au contrôle de la propagation des modifications vers les autres niveaux.

4.1 Reconstruction des spécifications des différents niveaux

La rétro-ingénierie d'une application est un processus complexe qui sort du cadre de cet article. Toutefois, nous allons présenter succinctement une démarche de rétro-ingénierie qui s'inscrit dans l'approche choisie (modélisation par niveaux). Une analyse plus approfondie des techniques et méthodes de rétro-ingénierie est présentée dans [HAIN93b]. La démarche est décomposée en deux phases principales : l'extraction des structures de données et la conceptualisation (figure 6).

La reconstruction des schémas physique SP0 et logique SL0 fait partie de la première phase du processus de rétro-ingénierie, appelée *extraction des structures de données*, dont le but est la recherche des structures et contraintes explicites et implicites grâce à l'analyse de diverses sources telles que les fragments éventuels de documentation, le code DDL, les programmes, les données, ainsi que les rapports, écrans et formulaires. Lors de cette phase, le concepteur enregistre dans l'historique CP0' les transformations effectuées sur SP0 pour obtenir SL0.

La reconstruction du schéma conceptuel SC0 à partir de SL0 fait partie de la deuxième phase de la démarche appelée *conceptualisation* ou *interprétation des structures de données* dont le but est de détecter et transformer les redondances et structures non conceptuelles. L'historique CL0' contient l'enregistrement des transformations effectuées sur SL0 pour obtenir SC0 pendant cette phase. Pour compléter la démarche, les historiques CP0' et CL0' sont inversés pour donner les historiques de conception CP0 et CL0. Ainsi, une documentation complète des spécifications a été recréée, les schémas de chaque niveau d'abstraction et les historiques de conception sont disponibles pour appliquer les stratégies de propagation développées dans les points suivants.

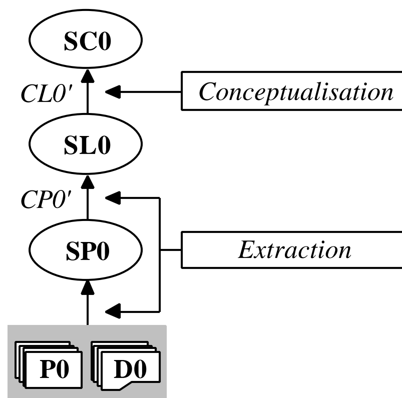


Figure 6 - *Démarche de rétro-ingénierie.*

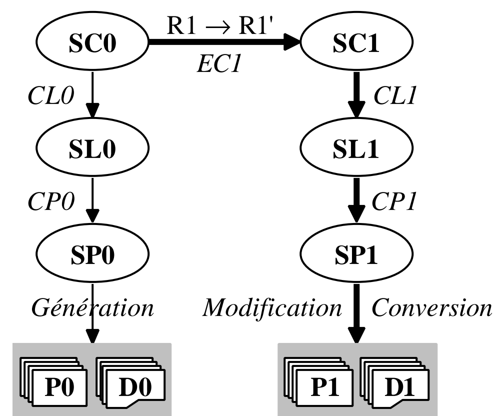


Figure 7 - *Propagation vers l'aval des modifications déduites de R1 ® R1'.*

4.2 Première stratégie : Modification des spécifications conceptuelles

Dans ce premier scénario, les modifications proviennent de changements dans les besoins au niveau conceptuel. Le problème est celui de la propagation de ces modifications aux couches inférieures, logique et physique. On suppose que les spécifications de tous les niveaux de modélisation existent : le schéma conceptuel SC0, le schéma logique SL0, le schéma physique SP0, la base de données D0 (schéma + données) et les programmes P0 de l'application (figure 7). Les transformations appliquées sur SC0 pour obtenir SL0 (conforme au modèle relationnel) et sur SL0 pour obtenir SP0 sont

enregistrées dans les historiques CL0 (conception logique) et CP0 (conception physique). Cette situation est traitée en quatre étapes.

4.2.1 Etape 1 : Modification du schéma conceptuel de la base de données

On suppose que les besoins auxquels SC0 satisfait évoluent de R1 vers R1'. Le changement est traduit par l'analyste en modifications du schéma SC0 qui devient SC1. Les transformations appliquées pour obtenir SC1 à partir de SC0 sont enregistrées dans l'historique EC1. On a donc la formulation transformationnelle : SC1 = EC1(SC0).

Le tableau ci-dessous présente une classification des principales modifications par type d'objets et degré de conservation de la sémantique. Des études plus approfondies de typologies de modifications aux niveaux conceptuel, logique et physique sont proposées dans [RODD93] et [HICK97].

Type d'objets	T+	T-	T=
Type d'entités	ajout	suppression	modification du nom, transformation en attribut
Type d'associations	ajout	suppression	modification du nom
Rôle	ajout, augmentation card. max., diminution card. min.	suppression, diminution card. max., augmentation card. min.	modification du nom
Relation IS-A (Sous-type)	ajout	suppression	
	changement de type		
Attribut	ajout, diminution card. min., augmentation card. max., extension domaine	suppression, aug. card. min., diminution card. max., restriction domaine	modification du nom, transformation en TE
	changement de type		
Identifiant	suppression, ajout composant	ajout, retrait composant	modification du nom, changement de type (P/S)

4.2.2 Etape 2 : Propagation des modifications en aval vers le niveau logique

Cette étape consiste d'abord à transformer le schéma SC1 en un schéma logique relationnel SL1 qui soit le plus proche possible de l'ancienne version, mais qui intègre les modifications conceptuelles. Pour ce faire, on rejoue l'historique CL0 sur SC1. Cet historique contient les transformations nécessaires pour transformer un schéma conceptuel (par exemple, figure 2) en un schéma conforme à un modèle relationnel (par exemple, figure 3). Le tableau ci-dessous présente une classification des principales transformations (de type T=) par type d'objet.

Type d'objets	T=
Type d'entités	Modification du nom, éclatement, fusion
Relation IS-A	Transformation en type d'associations + transformation en attribut de référence
Type d'associations	Transformation en type d'entités + transformation en attribut de référence, transformation en attribut de référence
Attribut	Modification du nom, transformation en type d'entités + transformation en attribut de référence, désagrégation d'un attribut décomposable, instantiation d'un attribut multivalué en attributs mono-valués, concaténation d'un attribut multivalué

Dans le schéma SC1, des structures ont pu être créées, supprimées ou modifiées par rapport au schéma SC0. On en déduit donc quatre situations de base :

- Un objet est inchangé, les transformations de CL0 le concernant sont appliquées telles quelles.
- Un objet inexistant dans SC0 est introduit dans SC1 : les transformations de CL0 resteront sans effet, l'objet devant être, si nécessaire, traité de manière spécifique. Ce traitement est sous la responsabilité de l'analyste.

- Un objet de SC0 a disparu dans SC1 : les transformations de CL0 peuvent être appliquées, mais resteront sans effet.
- Un objet de SC0 a été modifié dans SC1 : soit la modification est mineure et les transformations de CL0 sont toujours applicables sur l'objet modifié, soit l'objet doit être traité de manière spécifique car les transformations de CL0 ne sont plus adaptées (traitement sous la responsabilité de l'analyste).

L'application à SC1 de l'historique ancien CL0 auquel s'ajoute le traitement des nouveaux objets (par création ou modification) et duquel sont soustraits les traitements sans effet représente le nouveau processus de conception logique. Il y correspond un nouvel historique CL1.

4.2.3 Etape 3 : Propagation des modifications en aval vers le niveau physique

Le schéma SL1 doit être transformé en un schéma physique relationnel SP1 proche de l'ancienne version (SP0) et intégrant les modifications. On procède comme dans l'étape 2 en rejouant l'historique CP0 sur SL1. Cet historique contient les transformations concernant les structures physiques relationnelles (index et espaces de stockage) introduites au niveau physique. Le tableau ci-dessous présente une classification des principales transformations (de type T+ et T-).

Type d'objets	T+	T-
Index	ajout, ajout colonne	suppression, retrait colonne
Espace de stockage	ajout, ajout table	suppression, retrait table

Les modifications des structures dans le schéma SL1 peuvent engendrer quatre situations de base identiques à l'étape 2 lorsqu'on rejoue l'historique CP0. Il en résulte un historique CP1 contenant les transformations des objets non modifiés de SL1 ainsi que celles des nouveaux objets et duquel les transformations sans effet sont soustraites.

4.2.4 Etape 4 : Propagation des modifications vers les données et les traitements

Il reste à convertir la base de données (D0) - structures et données - et les programmes (P0) qui doivent se conformer aux nouvelles spécifications. La conversion des données (D0) peut être automatisée par la génération de *convertisseurs*. L'analyse de l'historique EC1 permet de repérer les modifications opérées au niveau conceptuel et l'analyse différentielle de CL0 et CL1, CP0 et CP1 nous fournit les informations nécessaires pour en dériver les structures physiques à supprimer, modifier ou ajouter.

Rejouer successivement l'historique CL1 puis CP1 sur SC1 donne le schéma physique SP1. On a : SP1 = CP1(CL1(SC1)) respectivement SP0 = CP0(CL0(SC0)). Pour simplifier l'exposé, nous appellerons C1 la concaténation des historiques CL1 et CP1. On obtient : SP1 = C1(SC1) et SP0 = C0(SC0).

En fonction du type des modifications présentes dans EC1, trois comportements distincts se présentent :

1. Dans le cas de la création d'un nouvel objet, l'analyse de C1 donne les nouvelles structures physiques à créer en fonction des transformations éventuelles appliquées successivement sur l'objet créé.
2. Dans le cas d'une suppression, l'analyse de C0 fournit les anciennes structures physiques à détruire en fonction des transformations éventuelles qui étaient appliquées sur l'objet détruit.
3. Le cas de la modification est plus complexe. Premièrement, les nouvelles structures doivent être créées sur base de l'analyse de C1. Ensuite, les instances doivent être transférées des anciennes structures vers les nouvelles. Et finalement, les anciennes structures sont détruites d'après le résultat de l'analyse de C0.

Sur base de l'analyse de EC1, C0 et C1, on déduit les transformations des structures de données et les transformations des instances de SP0 (c'est-à-dire D0) en instances de SP1 (D1). Ces transformations sont traduites sous la forme de convertisseurs, constitués de scripts SQL dans les cas simples, et de programmes dans les situations plus complexes. On obtient ainsi la nouvelle version de la base de données par application d'une chaîne de transformations à l'ancienne. A titre d'illustration, considérons la transformation de la figure 8, du type T+ (extension de la cardinalité maximale d'un rôle).

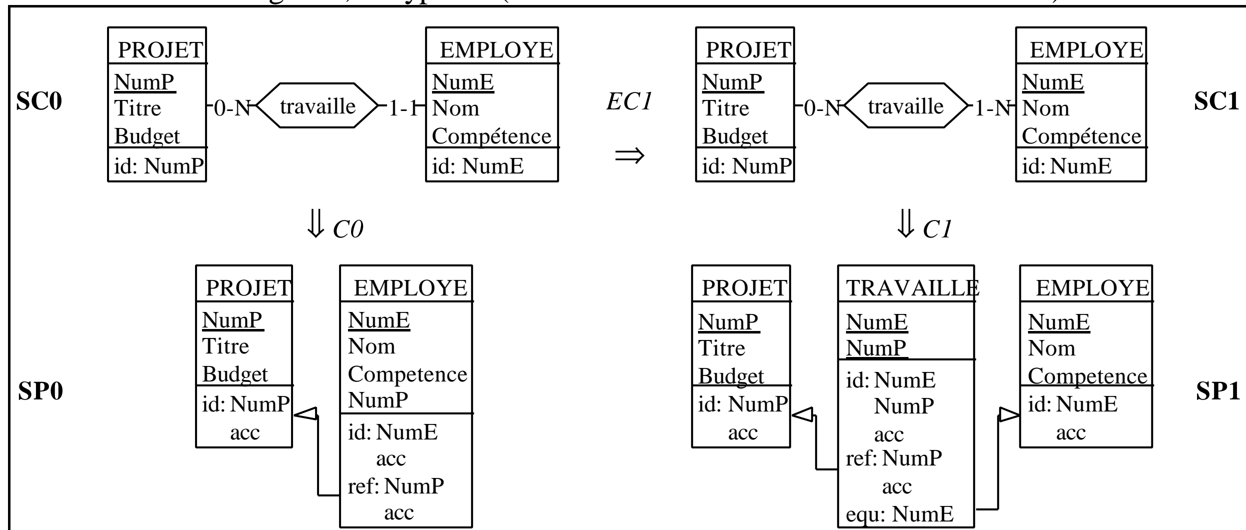


***Figure 8** - Extension de la cardinalité maximale du rôle travaille.EMPLOYE dans le schéma SC1 et propagation de cette modification en aval jusqu'au niveau physique (SP1).*

La modification de la cardinalité a entraîné la modification du type d'associations fonctionnel (un à plusieurs) *travaille* en type d'associations complexe (plusieurs à plusieurs). *travaille* ne peut plus être transformé en clé étrangère mais doit être transformé en table. L'analyse de C0 indique que le type d'associations *travaille* était représenté dans SP0 par un attribut de référence *NumP* dans *EMPLOYE* et une clé étrangère vers l'identifiant de *PROJET*. L'analyse de C1 nous apprend que *travaille* est maintenant représenté par une table *TRAVAILLE* contenant deux clés étrangères (une vers *PROJET* et l'autre vers *EMPLOYE*). Sur base de cette analyse, le script de conversion (instructions SQL) ci-dessous peut être généré :

```
-- Création des nouvelles structures :
CREATE TABLE TRAVAILLE (NumP NUMERIC(5) NOT NULL,NumE NUMERIC(5) NOT NULL,
                        CONSTRAINT IDTRAVAILLE PRIMARY KEY(NumP, NumE));
ALTER TABLE TRAVAILLE ADD CONSTRAINT FKPROJET
                      FOREIGN KEY (NumP) REFERENCES PROJET (NumP);
ALTER TABLE TRAVAILLE ADD CONSTRAINT FKEMPLOYE
                      FOREIGN KEY (NumE) REFERENCES EMPLOYE (NumE);
CREATE UNIQUE INDEX IDXTRAVAILLE ON TRAVAILLE (NumP, NumE);
CREATE INDEX IDXPROJET ON TRAVAILLE (NumP);
-- Transfert des instances :
INSERT INTO TRAVAILLE (NumP,NumE) (SELECT Nump,NumE FROM EMPLOYE);
-- Destruction des anciennes structures :
ALTER TABLE EMPLOYE DROP CONSTRAINT FKPROJET;
ALTER TABLE EMPLOYE DROP NumP;
```

Certaines modifications causent des pertes d'information et des violations de contraintes dans la base. Dans ce cas, le script de conversion doit d'abord fournir une requête pour vérifier la violation de la contrainte et, si c'est le cas, assurer la cohérence de la base de données après sa modification. Par exemple, si le concepteur crée un identifiant sur une colonne existante, les données stockées dans la table peuvent violer la contrainte d'unicité. Le script doit stocker dans une table temporaire les lignes ne

respectant pas la contrainte (une autre solution serait de laisser la première ligne dans la table et transférer les autres dans la table temporaire). Comme le montre l'exemple, il est toujours possible de créer des scripts de conversion mais une intervention de l'utisateur est parfois nécessaire pour traiter les instances problématiques ou simplement pour décider de l'exécution du script sur base du résultat de la requête de vérification de contrainte.

Modifier les programmes est une tâche beaucoup plus complexe, et pour l'instant non automatisable, sauf dans des situations simples où les modifications sont mineures. Pour caractériser l'impact des modifications des structures de données sur les programmes, nous avons défini trois catégories de modifications[2].

1. Certaines modifications de structures n'entraînent aucune modification au niveau des programmes. C'est le cas par exemple des modifications sur les index et les espaces de stockage. A priori, ces modifications ne nécessitent aucun aménagement des programmes[3].
2. Certaines modifications de structures nécessitent des modifications mineures au niveau des programmes. Dans le cas d'une table renommée, l'utilisation d'une fonction de recherche-remplacement est suffisante pour mettre à jour les fichiers de programmes.
3. Certaines modifications entraînent des modifications de la structure des programmes. Elles nécessitent souvent une connaissance approfondie de l'application.

```
Declare cursor E for
select NumE from EMPLOYE where <C>;
open E;
fetch E into :V1;
while SQLCODE = 0 do
   select Titre into :V2;
   from PROJET where NumP = :V1;
   TrP;
   TrE;
   fetch E into :V1;
endwhile;
close E;
```

⇒

```
declare cursor E for
select NumE from EMPLOYE where <C>;
open E;
fetch E into :V1;
while SQLCODE = 0 do
   declare cursor P for
   select Titre from TRAVAILLE T,PROJET P
   where T.NumE = :V1 and T.NumP = P.NumP;
   open P;
   fetch P into :V2;
   while SQLCODE = 0 do
      TrP;
      fetch P into :V2;
   endwhile
   close P;
   TrE;
   fetch E into :V1;
endwhile;
close E;
```

Figure 9 *- Impact sur un programme de la modification de la figure 8.*

Considérons la section de pseudo-code de la figure 9 (basée sur la figure 8) qui illustre la complexité de la propagation. Elle exprime un traitement TrP appliqué au projet de chaque employé vérifiant la condition <C> et un traitement TrE appliqué à chacun de ces employés. Les sections TrP et TrE se présentent sous la forme de séquences d'instructions. La conversion consiste à transformer l'accès à l'unique projet de l'employé courant et la séquence TrP en une itération qui parcourt les projets de l'employé courant. La difficulté réside dans l'identification de la séquence TrP, dont la frontière avec TrE n'est pas toujours aisée à déterminer, d'autant plus que des instructions indépendantes du projet peuvent

[2] Voir aussi [SHNE82], qui distingue les programmes dépendants et indépendants des modifications.

[3] Toutefois, dans le cas d'une modification d'index, sachant qu'elle peut avoir une influence sur les performances au niveau des accès, il est parfois utile de corriger les programmes pour profiter de ces performances.

apparaître dans TrP. Idéalement, seules les instructions dépendant du curseur P devrait subsister dans le corps de la boucle mineure.

En toute généralité, la modification des programmes est de la responsabilité du programmeur. Cependant, il est possible de préparer ce travail par une analyse du code qui permet de repérer et d'annoter les sections critiques des programmes. Des techniques d'analyse de programmes telles que les graphes de dépendance[4] et la fragmentation de programmes[5] (*program slicing*) permettent de localiser avec une bonne précision le code à modifier [WEIS84] [HENR98].

4.3 Deuxième stratégie : Modification des spécifications logiques

Selon un deuxième scénario, l'analyste apporte des modifications résultant du changement des besoins au niveau logique (figure 10). Le problème est celui de la reconstitution d'un historique de conception logique correcte et de la propagation en aval des modifications vers la couche physique, pour rendre les modifications opérationnelles. Cette situation est traitée en quatre étapes.

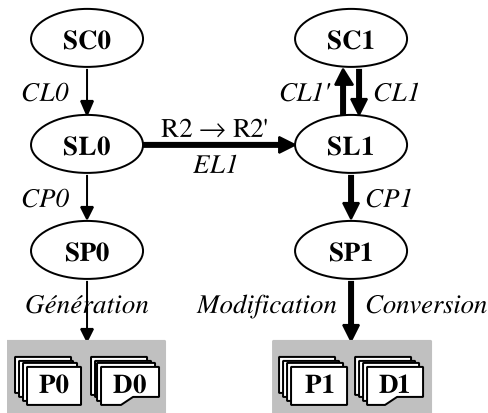


Figure 10 *- Propagation vers l'aval des modifications déduites de R2 ® R2' et reconstruction de CL1.*

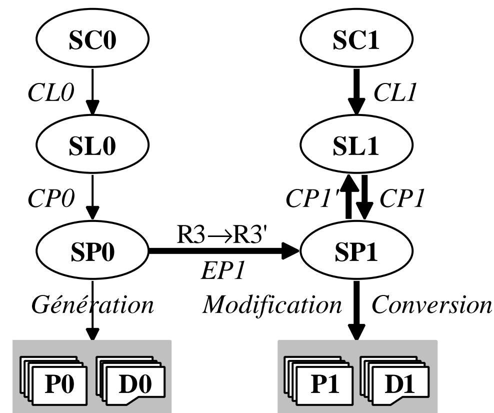


Figure 11 *- Propagation vers l'aval des modifications déduites de R3 ® R3' et reconstruction de CP1*

4.3.1 Etape 1 : Modification du schéma logique de la base de données

Cette étape consiste à modifier le schéma SL0 pour obtenir le schéma SL1. Les transformations sont enregistrées dans l'historique EL1. Il faut propager les modifications conduisant à SL1 en aval, vers SP1, D1 et P1, mais aussi reconstruire l'historique de conception logique CL1 pour obtenir SL1 à partir de SC1 et disposer ainsi d'une documentation complète du processus de conception. Une table des principales modifications au niveau logique[6] est présentée ci-dessous pour un sous-modèle relationnel.

Objets	Sémantique (T=)
Table	Modification du nom, éclatement, fusion
Colonne	Modification du nom
Identifiant	Modification du nom

[4] Il s'agit d'un graphe où chaque variable du programme est représentée par un noeud et dont les arcs (orientés ou non) représentent une relation (assignation, comparaison, etc.) entre deux variables.

[5] Cette technique extrêmement puissante sélectionne dans un programme toutes les instructions qui contribuent à définir l'état d'une variable à un point déterminé d'un programme.

[6] Il existe d'autres transformations permettant de prendre en compte des critères organisationnels ou techniques. Seules les plus fréquentes sont mentionnées.

Clé étrangère	Modification du nom

Une autre façon de modifier le schéma logique consiste à représenter par une autre structure logique un objet conceptuel non modifié. Par exemple, un attribut multivalué représenté par une série de colonnes dans SL0 est transformé en table dans SL1. Dans ce cas, le concepteur utilise la première stratégie. L'historique d'évolution conceptuel est vide, SC1 est identique à SC0 (étape 1 inutile). Lorsque le concepteur rejoue CL0 sur SC1 (étape 2), il n'exécute pas les transformations de CL0 sur l'objet qui doit être transformé différemment. Il enregistre à la fin de CL1 la nouvelle transformation appliquée sur l'objet. L'étape 3 reste inchangée. L'étape 4 se base sur les historiques C0 et C1 pour modifier les données et les programmes. C0 contient les anciennes structures physiques à détruire et C1 les nouvelles structures à créer.

4.3.2 Etape 2 : Reconstruction de l'historique de conception logique CL1

La propagation est basée sur les mêmes principes que ceux qui ont été développés dans la première stratégie. On dispose de l'historique CL0' (obtenu par inversion de CL0). SC1 est obtenu à partir de SL1 en deux phases : application de CL0', puis conceptualisation des nouveaux objets de SL1, cette dernière phase étant pilotée par l'analyste. L'historique de ces deux phases constitue CL1', dont l'inversion fournit CL1. On a donc de la sorte reconstruit une documentation à jour, qui permettra l'application des stratégies 1 et 2.

4.3.3 Etape 3 : Propagation des modifications en aval vers le niveau physique

Cette étape est similaire à la troisième étape de la première stratégie.

4.3.4 Etape 4 : Propagation des modifications en aval vers les données et les programmes

Cette étape consiste à propager les modifications vers la partie opérationnelle. Cette propagation a déjà été analysée dans le premier scénario. On peut toutefois préciser que, pour générer les convertisseurs, les historiques EL1, CP0 et CP1 sont analysés à la place des historiques EC1, C0 et C1 dans la première stratégie.

4.4 Troisième stratégie : Modification des spécifications physiques

La troisième stratégie consiste à modifier le schéma physique suite à des modifications des besoins techniques (figure 11). Les modifications doivent être propagées en aval vers les données et les programmes. L'historique de conception physique doit également être mis à jour. Cette situation est traitée en trois étapes.

4.4.1 Etape 1 : Modification du schéma physique de la base de données

Suite à des changements de besoins techniques, le schéma SP0 est transformé en un schéma SP1. Les transformations sont enregistrées dans l'historique EP1. Il faut propager les modifications conduisant à SL1 en aval, vers D1 et P1, mais aussi reconstruire l'historique de conception physique CL1 pour obtenir SP1 à partir de SL1. Une table des principales modifications au niveau physique est présentée ci-dessous pour un sous-modèle relationnel.

Type d'objets	T+	T-	T=
index	création, ajout colonne	suppression, retrait colonne	modification du nom
espace de stockage	création, ajout table	suppression, retrait table	modification du nom

4.4.2 *Etape 2 : Reconstruction de l'historique de conception logique CP1*

On inverse l'historique CP0 pour obtenir CP0'. SL1 est obtenu à partir de SP1 en deux phases : application de CP0', puis transformation des nouveaux objets de SP1, cette dernière phase étant pilotée par l'analyste. L'historique de ces deux phases constitue CP1', dont l'inversion fournit CP1. Notons que la conception physique travaille uniquement sur des index et des espaces de stockage. Les transformations des nouveaux objets de SP1 sont exclusivement des suppressions de ces structures. Ces structures techniques étant ignorées dans la conception logique, il n'est pas nécessaire de reconstruire l'historique CL1 car il equivaut à CL0.

4.4.3 *Etape 3 : Propagation des modifications vers les données et traitements*

La propagation des modifications vers les données et programmes a été analysée dans le premier scénario. Toutefois, pour générer les convertisseurs, l'analyse de l'historique EP1 est suffisante puisque EP1 contient les modifications sur le modèle physique qui est conforme aux structures de données de l'application.

5 L'approche DB-MAIN

L'étude de l'évolution et la maintenance d'applications de bases de données est réalisée dans le cadre du projet DB-MAIN. L'une des activités principales du projet est la conception d'un AGL en ingénierie et rétro-ingénierie de bases de données qui tente d'apporter des réponses aux lacunes des outils existants [HAIN96b]. Cet AGL fournit des outils qui supportent les stratégies développées dans la section 4. Dans les points suivants, nous présenterons d'abord les composants et les fonctions générales susceptibles d'intervenir dans l'évolution des applications. Ensuite, nous décrirons comment l'usage de cet AGL permet d'aider à propager les modifications des spécifications suivant la première stratégie (le principe est identique pour les deux autres).

5.1 Fonctions et composants de l'atelier

L'atelier met à disposition des fonctions et des composants d'usage très général qui permettent en particulier le développement de processeurs spécialisés assurant la gestion de l'évolution :

- un modèle générique de représentation de schémas basé sur un modèle Entité/Objet-Association étendu qui permet de représenter les structures de données à tous les niveaux d'abstraction;
- une interface graphique, qui permet de visualiser le contenu du référentiel selon plusieurs formats et de demander l'exécution d'opérations;
- une boite à outils transformationnelle concrétisée par une trentaine de transformations élémentaires applicables aussi bien en ingénierie qu'en rétro-ingénierie;
- des assistants programmables qui permettent la résolution de problèmes répétitifs et spécifiques (de transformation, de rétro-ingénierie, d'analyse de conformité de schémas, etc.) avec la possibilité de développer des résolveurs de problèmes via des scripts personnalisés; en particulier, les assistants de transformation permettent de créer des scripts complexes correspondant aux heuristiques de transformation implémentant les principaux processus d'ingénierie : normalisation, conceptualisation, conversion conceptuel/relationnel, optimisation, etc.
- des fonctions de gestion des historiques comprenant l'enregistrement, la sauvegarde, l'inversion ou la ré-exécution des opérations enregistrées;

- des outils d'analyse de programmes permettant de construire et consulter des graphes de dépendance des variables ainsi que des fragments de programmes;
- un langage de quatrième génération (Voyager 2) qui permet au concepteur de personnaliser l'atelier et de développer ses propres fonctions. C'est dans ce langage que sont implémentés les outils de conversion des données et d'annotation de programmes.

5.2 Implémentation de la première stratégie dans l'atelier

Au départ, l'existant (figure 12) est composé du schéma conceptuel SC0, du schéma logique SL0, du schéma physique SP0, de l'historique CL0 (traduction de SC0 en SL0), de l'historique CP0 (traduction de SL0 en SP0), de la base de données D0 (scripts de création des structures de données relationnelles + données) et des programmes P0 (fichiers contenant les textes sources).

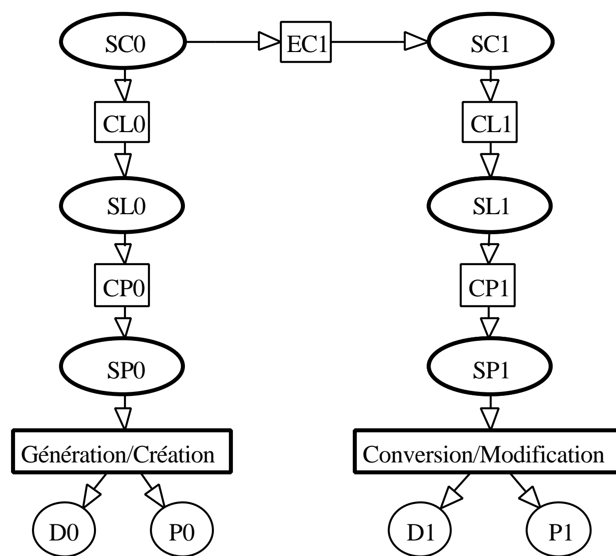


Figure 12 *- Représentation des schémas et historiques pour la première stratégie.*

Les processeurs nécessaires à la gestion de l'évolution existent sous forme native (C++) dans le noyau de l'atelier (point 5.1) à l'exception du générateur de convertisseurs, du générateur de patterns de recherche, du générateur de rapports de modifications des programmes qui sont développés en Voyager 2.

Examinons en détails les étapes de la démarche pour la première stratégie (figure 12).

5.2.1 Production de la nouvelle version du schéma conceptuel SC1

Le concepteur demande une copie de SC0, nommée SC1, et modifie SC1 en déclenchant l'enregistrement des transformations dans l'historique EC1.

5.2.2 Production de la nouvelle version du schéma logique SL1

Exécution de l'historique de conception logique CL0 sur SC1 : le concepteur demande une copie de SC1 (nommée SL1) sur laquelle il rejoue l'historique CL0 en déclenchant l'enregistrement des opérations dans le nouvel historique CL1. Il obtient ainsi une première version incomplète du nouveau schéma logique SL1.

Transformation des nouveaux composants de SC1 : les nouveaux composants de SC1 (provenant de la création de nouveaux objets ou de la modification d'objets de SC0) sont alors transformés

spécifiquement selon les heuristiques de transformation en vigueur dans l'entreprise. Ces opérations sont enregistrées dans l'historique CL1 à la suite des opérations enregistrées lors de l'exécution de CL0. L'historique CL1 contient toutes les transformations exécutées pour obtenir SL1 à partir de SC1.

5.2.3 Production de la nouvelle version du schéma physique SP1

Exécution de l'historique de conception physique CP0 sur SL1 : le concepteur réalise une copie de SL1 (nommée SP1) sur laquelle il rejoue CP0 en déclenchant l'enregistrement des opérations dans CP1. Il obtient ainsi un nouveau schéma logique incomplet SP1.

Transformation des nouveaux composants de SL1 : les nouveaux composants de SL1 (provenant de la création de nouveaux objets ou de la modification d'objets du schéma conceptuel) sont alors transformés spécifiquement. Ces opérations (modifications des index et des espaces de stockage pour prendre en compte les modifications de SL1) sont enregistrées dans CP1 à la suite des opérations déjà enregistrées. CL1 contient toutes les transformations nécessaires pour obtenir SP1 à partir de SL1.

5.2.4 Conversion de la base de données

L'analyste exécute le générateur de convertisseur sur base des historiques EC1, CL0 et CP0, CL1 et CP1. Ce générateur effectue les opérations suivantes :

1. Il construit une table des modifications qui contient les signatures des modifications effectuées sur le schéma conceptuel SC1. Une signature est composée d'une référence aux objets du schéma conceptuel qui ont été modifiés, créés ou supprimés ainsi que le type de la modification qui a été effectuée. Pour une création, on stocke la référence à l'objet de SC1 qui a été créé, pour une suppression, la référence à l'objet de SC0 qui a été supprimé et, pour une modification, la référence à l'objet qui est modifié de SC1 ainsi que la référence à l'objet original de SC0.
2. La table est ensuite simplifiée, car elle contient des modifications inutiles. En effet, pour modifier le schéma, le concepteur est parfois amené à procéder par essais et erreurs. Les modifications étant toutes enregistrées dans l'historique, il est nécessaire de simplifier la table pour qu'elle ne contienne plus que les modifications nécessaires et suffisantes. Par exemple, la modification du nom d'un attribut dans un type d'entités qui est détruit par la suite est inutile, seule la destruction du type d'entités est pertinente pour la table des modifications.
3. Les modifications sur SC1 sont traduites en modifications sur le schéma physique SP1. Après la concaténation (mise bout à bout) des historiques CL0 et CP0 (respectivement CL1 et CP1) en un historique unique C0 (respectivement C1), les références aux objets conceptuels sont remplacées par des références aux objets physiques. Pour ce faire, les historiques C0 et C1 sont parcourus pour trouver les structures physiques correspondantes aux structures conceptuelles modifiées. Dans le cas de la création, l'historique C1 est analysé et les références aux objets de SC1 sont remplacées par les références aux objets de SP1. Pour une destruction, l'historique C0 donne les anciennes références. Pour une modification, la référence à l'ancien objet est trouvée dans C0 et celle à l'objet modifié dans C1.
4. Une nouvelle simplification de la table est nécessaire. Elle dépend de la technologie choisie. Dans le cas des bases de données relationnelles, il est nécessaire de réaménager la table pour obtenir un script de conversion performant. Supposons que la table contienne la création d'une clé étrangère vers une table qui est renommée par la suite. La modification du nom d'une table, dans certains SGBD relationnels, est réalisée par une destruction suivie d'une création avec le nouveau nom. Dans ce contexte, il est inutile de créer une clé étrangère qui sera détruite par la destruction de la table référencée. Il faut donc déplacer la création de la clé étrangère après la modification de la table.

5. Sur base de la table des modifications, le générateur produit un convertisseur, généralement sous la forme d'un script de conversion des structures de données et des données elles-mêmes.

L'analyste exécute ce convertisseur.

5.2.5 Génération du rapport des modifications des programmes

Construction des patterns de recherche : sur base de la table des modifications créée au point 5.2.4, un analyseur génère les patterns syntaxiques qui vont constituer une base de recherche pour les analyseurs de programmes (*constructeur de graphes de dépendances* et/ou *fragmenteur de programmes*). Ces patterns dépendent du langage de programmation utilisé.

Exécution des analyseurs de programmes et génération du rapport des modifications : l'analyste exécute un analyseur de programmes (avec les patterns de recherche comme arguments) qui repère les sections de code dépendant des composants de la base de données qui ont été modifiés. Un générateur traite les résultats de l'analyseur et produit un rapport des modifications qu'il conviendrait d'appliquer aux programmes sous contrôle du programmeur.

6 Conclusions

Le problème de l'évolution d'un système d'information inclut celui des données qui en constituent le noyau. L'évolution des besoins se traduit techniquement par la modification des spécifications d'un des niveaux d'abstraction de ces données. La difficulté réside dans la propagation de cette modification vers les autres niveaux, et en particulier celui des composants opérationnels : schéma physique, données et programmes.

Les concepts de l'approche DB-MAIN, dédiée à l'ingénierie des données, forment une base formelle favorable à la compréhension et à la résolution du problème de l'évolution : modélisation transformationnelle des processus, représentation uniforme rigoureuse des spécifications aux différents niveaux d'abstraction et selon différents paradigmes, traçabilité des processus. Si les documents de la conception d'un système sont encore disponibles, ou à défaut, peuvent être reconstitués par rétro-ingénierie, alors le contrôle de l'évolution devient un processus formellement défini, et donc largement automatisable, pour ce qui concerne la base de données. En revanche, la modification des programmes reste un problème ouvert dans le cas général. Il est cependant possible d'aider le développeur à modifier le code de ces programmes par le repérage des sections où des instances des types d'objets modifiés sont traitées.

Nous disposons actuellement d'un prototype de contrôle de l'évolution de bases de données relationnelles selon les stratégies mentionnées dans cet article. Ce prototype a été développé en Voyager 2 sur la plate-forme générique DB-MAIN. Il est actuellement possible de générer de manière automatique les convertisseurs de bases de données correspondant à une séquence de modifications quelconques des types T-, T+ ou T=. Les règles relatives aux modifications des programmes sont définies, mais les outils sont en cours d'implémentation.

7 Références

[ALJA95] L. Al-Jadir, T. Estier, G. Falquet, M. Léonard, Evolution features of the F2 OODBMS, in Proc. of the 4th International Conf. on Database Systems for Advanced Applications, Singapore, World Scientific Publishing, Avril 1995.

[ANDA91] J. Andany, M. Léonard, C. Palisser, Management of Schema Evolution in Databases, Proc. of 17th International Conference on Very Large Databases (VLDB), Barcelona, Spain, September 1991.

[BATI92] C. Batini, S. Ceri, S.B. Navathe, Conceptual Database Design - An Entity-Relationship Approach, Benjamin/Cummings, 1992.

[BELL93] Z. Bellahsene, An Active Meta-Model for Knowledge Evolution in an Object-oriented Database, in Proc. of CAiSE'93, Springer-Verlag,1993.

[BROW96] A. Brown, E. Morris, S. Tilley, Assessing the evolvability of a legacy system, Software Engineering Institute, Carnegie Mellon University, technical report, 1996.

[HAIN93a] J.-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, Transformational techniques for database reverse engineering, in Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, E/R institute and Springer-Verlag, LNCS 823, 1993.

[HAIN93b] J.-L. Hainaut, M. Chandelon, C. Tonneau, M. Joris, Contribution to a theory of database reverse engineering, in Proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, IEEE Computer Press, May 1993.

[HAIN94] J-L. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, D. Roland, Evolution of database Applications: the DB-MAIN Approach, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994.

[HAIN96] J.-L. Hainaut, Specification Preservation in Schema transformations - Application to Semantics and Statistics, Data & Knowledge Engineering, Vol. 19, pp. 99-134, Elsevier, 1996.

[HAIN96a] J.-L. Hainaut, J. Henrard, J.-M. Hick, D. Roland, V. Englebert, Database Design Recovery, in Proc. of the 8th Conf. on Advanced Information Systems Engineering, Springer-Verlag, 1996.

[HAIN96b] J.-L. Hainaut, V. Englebert, J. Henrard, J.-M. Hick, D. Roland, Database Reverse Engineering: from requirements to CARE tools, in Journal of Automated Software Engineering 3(2), Kluwer Academic Press, 1996.

[HENR98] J. Henrard, D. Roland, V. Englebert, J.-M. Hick, J.-L. Hainaut, Outils d'analyse de programmes pour la rétro-ingénierie de bases de données, dans Actes du 16ème congrès INFORSID, Montpellier, Mai 1998.

[HICK97] J.-M. Hick, Typologie des modifications d'une application de base de données relationnelles, rapport technique, Institut d'informatique, FUNDP, Mai 1997.

[JARK94] M. Jarke, H.W. Nissen, K. Pohl, Tool integration in evolving information systems environments, 3rd GI Workshop Information Systems and Artificial Intelligence : Administration and Processing of Complex Structures, Hamburg, February 1994.

[JENS94] C. Jensen, and al., A consensus glossary of temporal database concepts, in Proc. International Workshop on an Infrastructure for Temporal Databases, Arlington (Texas), 1994.

[NGUY89] G.T. Nguyen, D. Rieu, Schema evolution in object-oriented database systems, in Data & Knowledge Engineering (4), Elsevier Science Publishers, 1989.

[RODD93] J.F. Roddick, N.G. Craske, T.J. Richards, A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models, in Proc. of 12th International Conf. on the Entity-Relationship Approach, Arlington, LNCS, 1993.

[SHNE82] B. Shneiderman, G. Thomas, An architecture for automatic relational database system conversion, ACM Transactions on Database Systems, 7 (2): 235-257, 1982.

[VANB94] P. van Bommel , Database Design Modifications based on Conceptual Modelling, in Information Modelling and Knowledge Bases V: Principles and Formal Techniques. Pages 275-286, Amsterdam, The Netherlands, 1994. IOS Press. Edited by H. Jaakkola, H. Kangassalo, T. Kitahashi, and A. Markus.

[VERE97] J. Verelst, Factors in conceptual requirements modeling influencing maintainability of information system: an empirical approach, in Proceedings of the Doctoral Consortium of the 3rd IEEE international Symponium on Requirements Engineering (RE), Annapolis (USA), 6-10 january 1997.

[WEIS84] M. Weiser, Program Slicing, IEEE TSE, Vol. 10, pp 352-357, 1984.