DB-MAIN : un atelier d'ingénierie de bases de données[1,2]

V. Englebert, J. Henrard, J.-M. Hick, D. Roland, J.-L. Hainaut[3]

Résumé

Les AGL (ou outils CASE) tant en conception qu'en rétro-ingénierie proposent trop souvent des solutions partielles aux problèmes d'ingénierie des bases de données. L'adoption d'hypothèses de travail trop simplistes, l'ignorance de critères non fonctionnels tels que l'optimisation, le manque de souplesse dans l'approche imposée par l'outil (transformation d'un schéma conceptuel en code sans raffinement possible) et l'ignorance de certains processus essentiels tels que la maintenance sont quelques exemples de faiblesses reconnues. De ces lacunes est née l'idée de concevoir un AGL, DB-MAIN, dont les caractéristiques seraient telles qu'il pourrait appréhender la majorité des problèmes inhérents à l'ingénierie des bases de données. Nous présenterons sept aspects (dont certains sont déjà opérationnels) qui nous ont semblés essentiels dans un tel outil : l'architecture de l'atelier, le modèle de spécification générique, l'approche transformationnelle, la personnalisation méthodologique de l'atelier, la rétro-ingénierie de BD, la maintenance et l'évolution des applications de BD et enfin la personnalisation fonctionnelle de l'atelier.

Mots clés

AGL, ingénierie des bases de données, méthodologie, rétro-ingénierie, évolution.

1. Introduction

Depuis quelque dix ans, des ateliers (AGL) d'aide à la conception de bases de données ont fait leur apparition sur le marché. Il apparaît aujourd'hui que ces logiciels auront eu sur la pratique des développeurs un impact relativement marginal eu égard à leur nombre (près d'une centaine ont été recensés) et à l'investissement que le développement de chacun a nécessité.

Cet échec a plusieurs causes objectives, dont l'une, essentielle, est sans conteste la faiblesse, pour ne pas dire l'absence, du support que ces outils offrent dans certaines phases critiques de l'ingénierie des systèmes d'information et des bases de données en particulier. Nous citerons deux domaines particulièrement frappants.

1. Le processus que privilégient ces outils est celui de la conception de bases de données, qui part de l'élaboration du schéma conceptuel, et qui se termine par la production du code exécutable de définition de structures, typiquement sous la forme d'un script SQL. Ces outils montrent cependant rapidement leurs limites lorsqu'on envisage la production de systèmes complexes et performants, au-delà des prototypes et des applications pédagogiques. Des processus critiques tels que la normalisation conceptuelle, l'intégra-

---

[1] DB-MAIN est un projet réalisé avec les partenaires suivants : 3 Suisses (Be), ACEC-OSI (Be), Ariane-II (Be), Banque UCL (Lux), BBL (Be), Centre de recherche public H. Tudor (Lux), CGER (Be), Cockerill-Sambre (Be), CONCIS (Fr), D'Ieteren (Be), DIGITAL (Be), EDF (Fr), Groupe S (Be), IBM (Be), OBLOG Software (Port), ORIGIN (Be), Winterthur (Be), en collaboration avec l'EPFL (CH).

[2] Le projet conjoint DB-PROCESS est supporté par la Communauté Française de Belgique.

[3] Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, rue Grandgagnage 21, B-5000 Namur, BELGIQUE; Tél.: +32 81 72 49 85; Fax : +32 81 72 49 67; Internet : db-main@info.fundp.ac.be

tion de schémas, la conception logique (avec prise en compte de critères tels que la minimisation du temps de réponse ou de l'espace occupé, ou la distribution optimale dans un réseau de serveurs), la conception physique et la dérivation des vues sont le plus souvent ignorés.

2. Il apparaît que les besoins en développement de nouveaux systèmes sont actuellement dépassés par les besoins en maintenance, extension, migration, ré-ingénierie et évolution de systèmes existants. En dépit des affirmations publicitaires et des réputations soigneusement entretenues, il n'existe pas aujourd'hui d'outils commercialisés capables de supporter de manière convaincante la rétro-ingénierie de bases de données et de fichiers complexes et mal conditionnés comme ils le sont en majorité en pratique.

Loin d'être inutiles pourtant, ces outils ne sont cependant souvent appréciés que pour leurs qualités passives : ergonomie (via leur interface graphique notamment) et production de documentation. Cette situation risque en outre de perdurer en raison, d'une part, de la complexité intrinsèque de ces processus et de la maîtrise insuffisante qu'on en a, et d'autre part de la faible rentabilité du développement et de la commercialisation de ces outils.

Ces observations sont à l'origine du projet de recherche DB-MAIN[4], consacré aux problèmes de la maintenance et de l'évolution des applications de bases de données. L'une des activités principales du projet est la conception et le développement d'un AGL en ingénierie de bases de données qui tente d'apporter des réponses aux lacunes des outils actuels. Cet article présente brièvement quelques aspects originaux d'une première version de cet atelier :

- l'architecture de l'atelier (section 2)
- le modèle de spécification générique (section 3)
- l'approche transformationnelle (section 4)
- la personnalisation méthodologique (section 5)
- la rétro-ingénierie des bases de données (section 6)
- la maintenance et l'évolution des bases de données (section 7)
- la personnalisation et l'extensibilité fonctionnelle de l'atelier (section 8).

Le lecteur intéressé trouvera dans [HAINAUT,94] un exposé détaillé du contenu du projet DB-MAIN et dans [HAINAUT,95] un développement plus complet des bases théoriques et techniques de l'atelier.

2. Architecture de l'atelier

L'architecture de DB-MAIN décrite à la figure 1 met en évidence les principaux composants de l'atelier :

- l'interface graphique, qui permet de visualiser le contenu du référentiel et de demander l'exécution des opérations; elle est pilotée par le moteur méthodologique;
- le moteur méthodologique, qui guide l'analyste dans le suivi d'une méthode d'ingénierie propre à son entreprise (section 5);
- les assistants, qui sont des outils d'aide à la résolution de problèmes répétitifs et spécifiques (de transformation, de conformité, de rétro-ingénierie, ...); pour résoudre des problèmes complexes et répétitifs, l'analyste peut utiliser des scripts prédéfinis ou personnalisés (section 8);

---

[4] pour *Database Maintenance*.

- la machine VOYAGER, interpréteur du L4G *Voyager 2*, qui exécute des fonctions personnalisées développées par l'analyste (section 8);
- les outils de base, qui permettent l'accès au référentiel, la gestion de son contenu, l'importation/exportation de spécifications, la transformation de schéma (section 4), l'analyse de textes (extracteurs et moteur de *patterns,* section 6),...

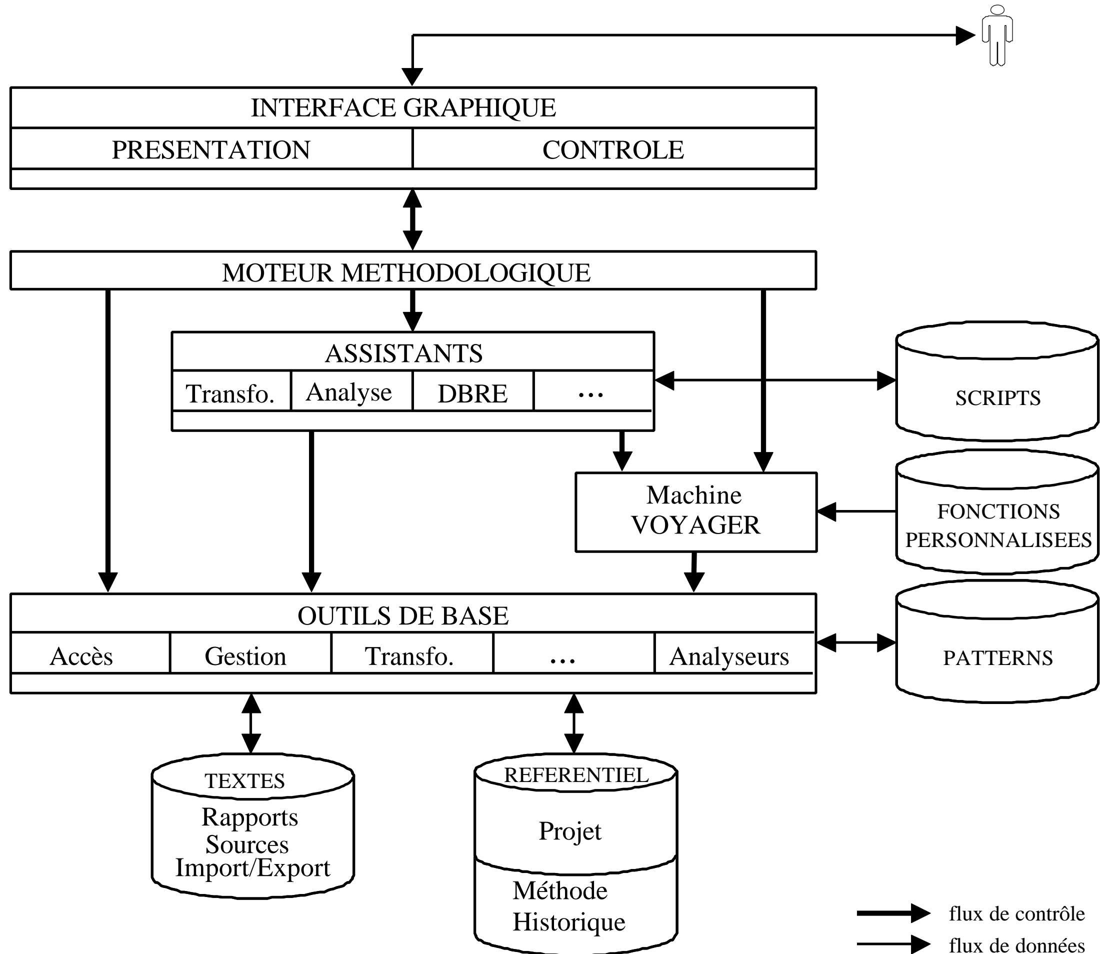

Figure 1 - Architecture de l'atelier DB-MAIN

3. Modèle de spécification générique

DB-MAIN utilise un modèle unique de représentation des schémas de bases de données qui est basé sur le modèle Entité-Association (EA) étendu [HAINAUT,89], [HAINAUT,92]. Ce modèle présente des caractéristiques de généricité selon deux dimensions. D'une part, il couvre les niveaux d'abstraction habituels dans le domaine des bases de données. Il permet, par exemple, de représenter aussi bien des schémas conceptuels que des schémas logiques ou physiques. Il est également possible, pour certains processus, tels que la rétro-ingénierie, de combiner dans un même schéma des objets de niveaux d'abstraction différents. D'autre part, il couvre les principaux paradigmes et technologies. Des schémas exprimés dans les modèles EA, NIAM, OO, relationnels, CODASYL, IMS, fichiers standards peuvent être représentés avec

précision dans ce modèle. Ce modèle unique permet la personnalisation méthodologique (section 5), et constitue un support idéal pour l'approche transformationnelle (section 4). On trouvera dans [ROSENTHAL,94] une proposition d'architecture basée sur la même idée d'un modèle unique.

L'ensemble des schémas produits lors d'une analyse sont regroupés sous forme d'un projet. Ce dernier comprend également des références à toutes les autres informations utilisées et enregistrées dans des fichiers annexes (rapports d'entretiens, fichiers sources, scripts SQL générés,...).

4. Approche transformationnelle

Dans le domaine du génie logiciel, la conception d'un composant est souvent modélisée comme une séquence de transformations [FICKAS,85]. Dans le domaine des bases de données, la conception de schéma peut être définie comme une suite de modifications de structures de données. On parle alors d'approche transformationnelle car toutes les modifications appliquées sur un schéma sont considérées comme des transformations et le processus de conception de bases de données est modélisé comme une séquence de transformations de schéma. Par exemple, ajouter un type d'entités, modifier le nom d'un attribut ou changer un type d'associations en type d'entités sont des transformations de schéma alors que normaliser un schéma conceptuel, optimiser un schéma logique, générer du code SQL et reconstruire un schéma conceptuel à partir du code source sont des séquences de transformations. DB-MAIN est basé sur une telle approche transformationnelle. On consultera également [ROSENTHAL,94], qui propose une technique similaire. Le lecteur intéressé par l'approche transformationnelle consultera par exemple [HAINAUT,93].

Une transformation est définie par les préconditions minimales qu'une structure de données doit satisfaire avant l'opération et les postconditions maximales qu'elle doit vérifier après transformation. Si T est une transformation, C et C' des structures de données, on peut écrire : C' = T(C).

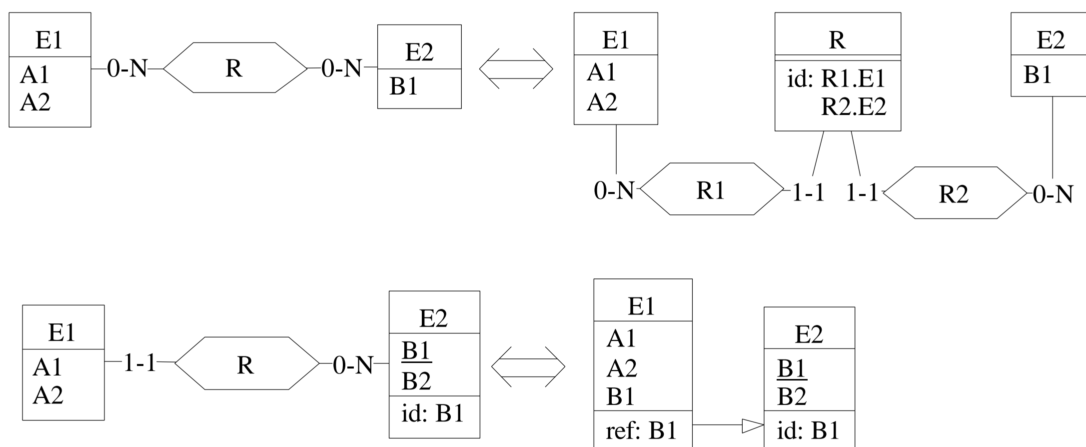

Figure 2 - En haut : transformation d'un type d'associations R en type d'entités et, son inverse. En bas : transformation d'un type d'associations R en clé étrangère, et son inverse.

Certaines transformations augmentent la sémantique d'un schéma (ajouter un type d'entités), d'autres la diminuent (supprimer un attribut). Il existe aussi des transformations qui préservent la sémantique d'un schéma, c'est-à-dire qui sont telles que C et T(C) décrivent le même univers de discours (remplacer un type d'associations par un type d'entités). Ces dernières transformations sont appelées *réversibles*. Une transformation T est réversible s'il existe une transformation T' telle que : T'(T(C)) = C. Si T' est également réversible, on dit que T et T' sont *symétriquement réversibles*. Pour plus d'informations sur la notion de transformation, on consultera [HAINAUT,91], [HAINAUT,94].

La figure 2 présente deux exemples de transformations symétriquement réversibles.

5. Personnalisation méthodologique de l'atelier (sous-projet DB-Process)

L'outil peut être utilisé dans des contextes très variés. C'est pourquoi les fonctions de base présentées dans la section précédente sont neutres vis-à-vis de toute démarche. Mais il peut également être paramétré pour refléter une démarche particulière, liée à la culture méthodologique locale d'une entreprise.

La spécialisation méthodologique de l'atelier est basée sur un modèle des processus de conception qui permet de spécifier, via un langage adéquat, une méthode particulière. Cette méthode est interprétée par le moteur méthodologique qui contrôle à la fois les actions permises à l'utilisateur et la présentation du contenu du référentiel.

Cette personnalisation permet essentiellement quatre fonctions :

- adaptation à la culture locale;
- guidance méthodologique;
- enregistrement de l'historique des activités (tracing);
- support des activités de maintenance et d'évolution des bases de données (section 7).

Des travaux connexes ont déjà été réalisés en ingénierie des besoins [ROLLAND,93], en software engineering [YONESAKI,93], [POTTS,88] et dans les systèmes d'information [JARKE,93].

Dans un premier temps, nous allons présenter les briques de base et la manière dont elles s'assemblent dans un modèle en deux couches. Un petit exemple viendra ensuite préciser les concepts. Enfin, nous expliquerons comment tout ceci peut être mis en pratique.

5.1. Spécialisation du modèle générique de représentation

Le modèle de représentation des schémas étant générique, il offre un très grand nombre de degrés de liberté, conduisant par exemple à mélanger des concepts de paradigmes ou de niveaux d'abstraction différents dans un même schéma. Nous allons définir des sous-modèles caractérisant ces niveaux d'abstraction. D'une part, en restreignant le modèle générique, c'est-à-dire en spécifiant une série de contraintes sur ce modèle, et d'autre part en adaptant la terminologie des différents objets. Par exemple, pour définir un sous-modèle de représentation de **schémas relationnels**, les contraintes suivantes doivent être imposées :

- Pas de relation is-a;
- Pas de types d'associations;
- Pas d'attributs décomposables;
- Pas d'attributs multivalués;
- Pas de types d'entités sans attributs.

Tout schéma qui respecte ces contraintes est conforme au sous-modèle relationnel. Quant à la terminologie, elle devra être adaptée comme suit :

- Collection → *dbspace*
- Type d'entités → *table*
- Attribut → *column*
- Identifiant primaire → *primary key*

Dans le cadre de la modélisation méthodologique, un sous-modèle définit un type de schéma. Plus généralement, nous parlerons de produits et de types de produits, un *produit* est soit un *schéma*, soit un *fichier* pouvant contenir des informations relatives à des structures de données. Un script SQL, un fichier source COBOL, un rapport d'entretien, des copies d'écran,... ne sont que quelques exemples de fichiers. Tous ces fichiers sont autant de sources d'informations qui peuvent être utiles dans les processus de conception, mais aussi dans les processus de rétro-ingénierie.

5.2. Historique de la conception

Un AGL doit permettre l'enregistrement sous forme d'un historique de toutes les manipulations effectuées. Celui-ci peut avoir de nombreuses utilisations telles que :

- documenter la conception d'une application en enregistrant toutes les actions qui ont conduit à sa réalisation;
- annuler la ou les dernières actions effectuées;
- rejouer une conception à partir du schéma de départ légèrement modifié (cfr. section 7);
- recouvrer une conception, c'est-à-dire reconstruire un historique possible d'une application à partir de l'historique de la rétro-ingénierie de cette application (cfr. section 6).

Ces différentes utilisations ont des exigences très différentes, voire contradictoires, quant à la forme et au contenu de l'historique. Pour la documentation, il doit être lisible et compact tandis que pour sa réexécution, l'annulation et le recouvrement d'une conception il doit être complet et formel. Dans ce dernier scénario, l'historique doit contenir les informations sur l'action effectuée et l'objet sur lequel l'action s'effectue, tandis que pour l'annulation et le recouvrement de conception il faut disposer de suffisamment d'informations pour retrouver la situation de départ. Il faut donc des vues différentes d'un même historique.

5.3. Modélisation des processus : un modèle à deux niveaux

Un *processus* est une transformation de produits. Le processus de conception logique, par exemple, est la transformation d'un schéma conceptuel en un schéma logique. Un processus est représenté par son historique. Un *type de processus* est la description d'une transformation. Il est défini d'une part par la liste des types de produits qui seront analysés et ceux qui seront générés, d'autre part par la marche à suivre pour réaliser la transformation (sa *stratégie*). Une *hypothèse* est la restriction d'un type de processus à un contexte donné. Ainsi, pour réaliser un processus d'un type donné, l'analyste pourra faire plusieurs hypothèses et réaliser plusieurs processus du même type, chaque fois dans un contexte différent. Il obtiendra plusieurs *versions* d'un schéma. Il devra ensuite prendre une *décision*, c'est-à-dire choisir une solution parmi toutes les versions obtenues. Cette décision est un processus particulier, d'un type prédéfini, qui ne donne pas lieu à une transformation de produit, mais qui sélectionne simplement un produit parmi plusieurs. Une *justification* est une annotation libre de l'historique par laquelle l'analyste explique un choix effectué.

Toutes les notions qui viennent d'être définies s'agencent dans un modèle à deux niveaux. Ce modèle est représenté à la figure 3. Au niveau des méthodes, un type de processus est la description d'une transformation de produits de types spécifiés. Les types de processus s'enchaînent en séquence (un type de processus est suivi d'un autre), en parallèle (un type de processus est suivi de plusieurs ou en suit plusieurs) ou itérativement (un type de processus se suit lui-même) et se décomposent en sous-processus qui sont eux-même des processus. Cette décomposition s'arrête lorsqu'on est face à des processus élémentaires. Au niveau des historiques, les processus se partagent en deux catégories : les processus de transformation qui sont des instances des types de processus et les décisions. Les produits sont des instances des types de produit. Les processus sont réalisés à partir de produits de type spécifié et en génèrent d'autres. A ce niveau aussi, les processus sont divisés en sous-processus, en gardant la structure définie au niveau des méthodes. Par exemple, le journal de la conception d'une base de données mentionne une série d'informations au sujet des différentes grandes phases (analyse conceptuel suivant deux hypothèses différentes, conception logique, conception physique) et ensuite affine chacune de ces phases. Au niveau des instances, les processus se succèdent séquentiellement (un processus en suit un autre, jamais lui-même), mais plusieurs hypothèses peuvent être suivies (un processus suivi de plusieurs) et des décisions peuvent être prises (un processus qui en suit plusieurs).

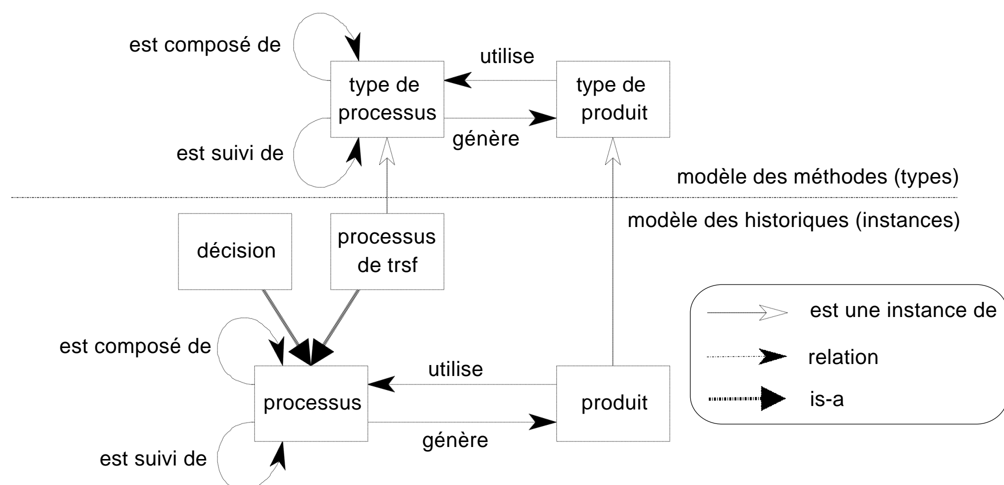

Figure 3 - Le modèle à deux niveaux

5.4. Exemple

Examinons l'exemple de la figure 4. A gauche, une méthode, à droite, un historique obtenu en suivant la méthode. Celle-ci se compose de deux types de processus : PA travaille sur des produits de type A et génère des produits de type B, PB travaille sur des produits de type B et génère des produits de type C. PA doit être exécuté avant PB. PA et PB pourraient représenter respectivement l'analyse conceptuelle et la conception logique. Ils doivent, à leur tour, être affinés par une description similaire.

L'historique nous montre que l'analyste a commencé son travail en accomplissant pa, une instance de PA, sur base de a1 et a2, deux produits de type A. Il a généré b, un produit de type B. Pour exécuter PB, un dilemme s'est présenté à lui. Il a envisagé deux hypothèses distinctes et a

réalisé deux processus de type PB : pb1 qui a donné naissance au produit c1 et pb2 qui a engendré c2. Sur base de ces deux résultats, il a pu prendre une décision. Il a choisi de conserver c2 et d'abandonner c1. Ces hypothèses et cette décision font partie de l'historique.

5.5. Mise en oeuvre dans DB-MAIN

Le moteur méthodologique s'insére dans l'atelier (Figure 1) afin d'assister l'analyste et de contrôler son travail. La procédure à suivre est la suivante : la méthode choisie est décrite dans le langage de spécification de méthode [ROLAND,95], puis sa définition est chargée dans le référentiel du projet sous la forme, entre autres choses, de types de produits et de types de processus. Dès cet instant, le projet est géré selon les concepts et la démarche de la méthode choisie. En particulier, le moteur méthodologique agit sur l'interface utilisateur (sous-modèle actif, terminologie, etc) et sur l'accessibilité des différentes fonctionnalités de l'atelier.

L'historique est enregistré de manière automatique. Un processeur spécialisé permet de traiter et manipuler les historiques :

- nettoyer par la suppression de séquences sans effet (p.ex. : une transformation directement suivie de son inverse);
- compléter de certaines annotations;
- dériver des historiques par inversion et par projection;
- produire des rapports synthétiques, etc.

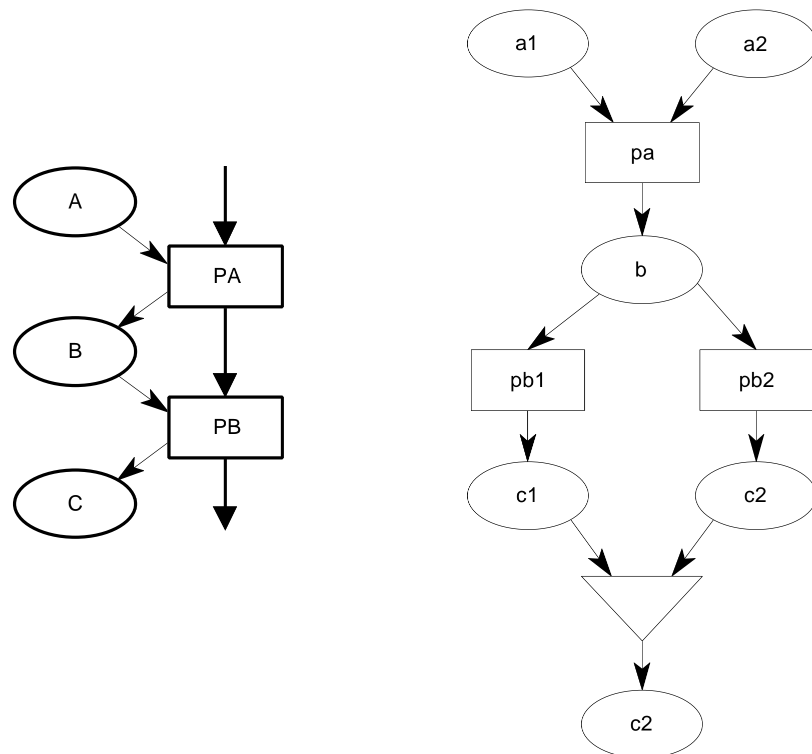

Figure 4 - Un fragment de méthode et un historique

6. Rétro-ingénierie des bases de données

La rétro-ingénierie d'un composant logiciel est un processus d'analyse de la version opérationnelle de ce composant qui vise à en reconstruire les spécifications techniques et fonctionnelles. La rétro-ingénierie a comme but la redocumentation, la conversion, la maintenance ou l'évolution d'anciennes applications.

La rétro-ingénierie est un processus d'autant plus complexe que l'application est mal structurée, ancienne, non ou mal documentée. Il est possible de réduire cette complexité en attaquant d'abord l'analyse des structures des données persistantes de l'application. Cette approche peut se justifier de la façon suivante :

- les données persistantes constituent le composant central de beaucoup d'applications;
- la connaissance de la structure des données persistantes facilite la compréhension de l'application;
- les données persistantes sont généralement la partie la plus stable d'une application;
- la méthodologie des bases de données est plus formalisée que celle du logiciel en général.

La majorité des propositions de méthodes imposent des hypothèses trop restrictives pour traiter complètement des applications complexes[5] :

- la base de données a été obtenue via des règles de transformation conceptuel/logique simplistes, et par conséquent la traduction du schéma physique vers le schéma conceptuel est presque immédiate;
- le schéma n'a pas subi de restructurations d'optimisation;
- toutes les contraintes ont été traduites dans le langage de description de données;
- les noms sont significatifs;
- les méthodes sont spécifiques à un type de SGD[6];
- pas (ou peu) de prise en compte du code procédural[7].

Cette section est structurée en deux parties. La première présente une méthode générique de rétro-ingénierie de bases de données, indépendante du type de SGD. La seconde énumère les fonctions de rétro-ingénierie offertes par DB-MAIN.

6.1. Une méthode générique de rétro-ingénierie de bases de données

Cette méthode comporte deux processus principaux (Figure 5), à savoir l'extraction des structures de données et la conceptualisation de ces structures [HAINAUT,95]. Cette division correspond à peu près à l'inverse des processus de conception physique et de conception logique habituellement admis pour l'ingénierie des bases de données.

6.1.1. L'extraction des structures de données

Cette phase a pour objet de retrouver le schéma logique complet du SGD, y compris toutes les contraintes et les structures de données non explicitement déclarées.

Certains SGD (les SGBD par exemple) offrent, sous une forme ou une autre, une description du schéma global des données. Bien que ce schéma soit déjà assez complet, il devra être enri-

---

[5] Contre-exemples : [PREMERLANI,93].

[6] SGD : Système de gestion de données.

[7] Les choses évoluent depuis peu cependant. Voir par exemple [PETIT,94], [ANDERS,94], [SIGNORE,94].

chi grâce à une analyse des autres composants de l'application (vues, schémas, code procédural, données, écrans de saisie).

Le problème est beaucoup plus complexe quand il s'agit de recouvrer le schéma conceptuel de fichiers classiques. Chaque programme source devra être analysé pour retrouver une partie de la structure des données. Cette analyse doit aller bien au-delà de la simple recherche de la structure des fichiers déclarés dans les programmes.

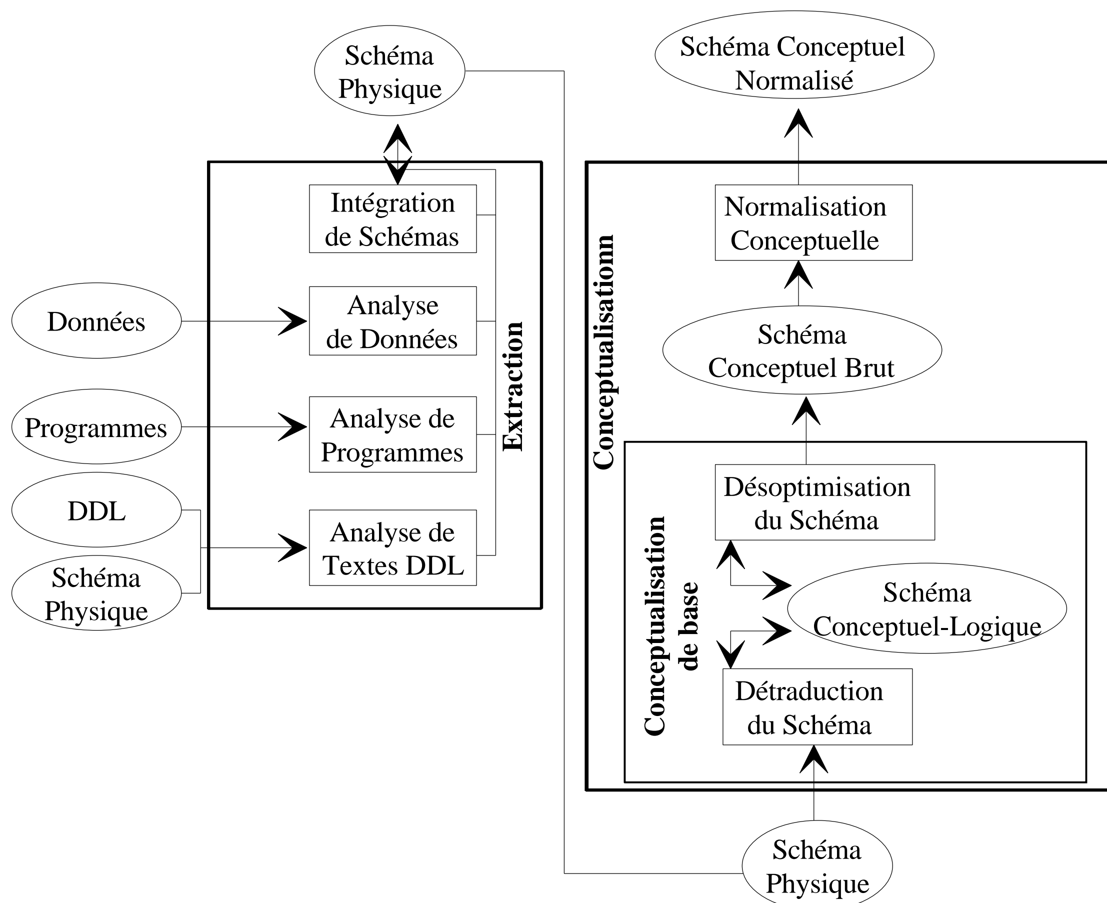

Figure 5 - Méthode générique[8] de rétro-ingénierie de bases de données

En particulier, trois types de problèmes peuvent être rencontrés, quel que soit le SGD : les structures cachées, les structures non déclarées, les pertes de spécifications.

Une *structure cachée* est une structure de données ou une contrainte qui aurait pu être représentée directement dans le SGD mais qui ne l'a pas été. Par exemple, un champ multivalué ou composé est représenté par un champ monovalué et atomique; une contrainte référentielle n'est pas implémentée comme une clé étrangère mais est vérifiée dans le code procédural.

Une *structure non déclarée* est une structure ou une contrainte ignorée par le SGD et qui doit être représentée ou vérifiée par d'autres méthodes, généralement procédurales. Par exemple : les contraintes référentielles dans les fichiers standards.

---

[8] Par simplicité, nous n'avons pas indiqué l'ordre d'exécution des processus comme indiqué à la figure 3.

La *perte de spécifications* vient de la non-implémentation dans le SGD et dans le programme, de certaines contraintes du schéma conceptuel. Elles peuvent, par exemple, être vérifiées par construction : les données sont importées et sont correctes; l'exécution du programme ne permet pas de violer ces contraintes.

Recouvrer les structures cachées, non déclarées et perdues est une tâche complexe, pour laquelle il n'existe pas encore de méthode déterministe. Seule une analyse méticuleuse de toutes les sources d'informations disponibles permettent de retrouver les spécifications. Le plus souvent ces informations doivent être consolidées par la connaissance du domaine.

Les processus principaux de l'extraction des structures de données sont les suivants :

- Analyse des déclarations des structures de données dans les scripts de définition du schéma et dans les sources du programme.
- Analyse du code source du programme pour recouvrer les structures cachées et non déclarées.
- Analyse des données sur lesquelles travaille le programme pour en trouver les structures et les propriétés et pour confirmer ou infirmer certaines hypothèses.
- Intégration des différents schémas obtenus lors des étapes précédentes.

6.1.2. La conceptualisation des structures de données

Ce deuxième processus consiste en l'interprétation du schéma obtenu lors de l'extraction des structures de données pour en dériver un schéma conceptuel. Il détecte et transforme (ou élimine) les redondances et les structures non conceptuelles introduites lors de la conception de la base de données. La conceptualisation des structures de données se fait en deux étapes : la conceptualisation de base et la normalisation.

Lors de la conceptualisation de base, toutes les structures de données qui ne sont pas conceptuelles sont éliminées ou transformées. Elle est elle-même décomposée en :

- Détraduction du schéma : le schéma de départ de ce processus est conforme au modèle du SGD utilisé, toutes ces structures de données spécifiques au SGD doivent être détectées et transformées en structures de données conceptuelles équivalentes.
- Désoptimisation du schéma : le schéma a été restructuré et enrichi pour des raisons d'optimisation. Il faut détecter et éliminer ces structures.

Finalement nous disposons d'un schéma conceptuel qu'on peut encore transformer de manière à le rendre conforme à un standard méthodologique par exemple. Il s'agit d'un processus classique de normalisation conceptuelle.

6.2. Les fonctionnalités de rétro-ingénierie offertes par DB-MAIN

Avant tout, les processus de rétro-ingénierie utilise intensivement les fonctionnalités de DB-MAIN développées au départ pour la conception de bases de données. Parmi celles-ci, nous pouvons citer le modèle de réprésentation des données générique, les différentes vues d'un même schéma, l'approche transformationnelle (via les transformations inverses), l'enregistrement de l'historique pour documenter et pour reconstituer un historique inverse.

Certains processeurs sont cependant spécifiques. Nous présenterons les outils de manipulation de textes, qui sont essentiellement (mais pas exclusivement) liés à la rétro-ingénierie.

Les textes sources fournissent une information importante pour la rétro-ingénierie, et font donc partie intégrante du projet (section 2). Plusieurs fonctions de présentation et d'analyse de ces textes ont été développées :

- Extraction automatique des structures de données à partir de textes SQL, COBOL, CODASYL, etc.
- Affichage des textes sources.
- Outil de recherche de *patterns* avec instantiation de variables (section 8.1). Les *patterns* sont exprimés dans un language de définition de *patterns* (PDL) dérivé de la notation BNF.
- Outil permettant de construire et consulter le graphe de dépendance des variables d'un texte source. On donne à l'outil la liste des *patterns* qui permettent de construire le graphe de dépendance.
- Enrichissement du schéma à partir de structures de données découvertes dans les textes sources.
- Outil de navigation des schémas vers les textes sources et inversément (parcours des correspondances).

7. Maintenance et évolution des applications de bases de données

7.1. Problème

Les applications de bases de données, au cours de leur cycle de vie, comportent généralement des phases d'évolution correspondant soit à des corrections, soit, plus généralement à des modifications d'évolution. Ces modifications proviennent de nouveaux besoins de type fonctionnel (satisfaire les exigences des utilisateurs), organisationnel (modification du cadre de travail dans lequel s'inscrit l'application) ou technique (modifications de contraintes techniques ou matérielles). Pour satisfaire ces nouveaux besoins, le concepteur va apporter des modifications techniques à son application.

On considére l'analyse de ces modifications dans le cadre de la modélisation classique qui définit essentiellement trois niveaux d'abstraction : conceptuel, logique et physique (Figure 6). Les modifications des besoins interviennent dans un de ces niveaux. Le problème de l'évolution sera de les propager aux autres niveaux (en amont et/ou en aval).

Le problème de la propagation devient encore plus complexe si on considère 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 bien documentés. Ce n'est pas toujours le cas. Dans certaines situations, les programmes et les données sont les seules parties disponibles de l'application. Dans ce cas, comment l'application va-t-elle évoluer ? Dans [HAINAUT,94], on détaille les principaux types de problèmes qui peuvent se présenter et les stratégies correspondantes.

Actuellement, on constate que, face à ces problèmes, le concepteur est démuni tant du point de vue méthodologique que du point de vue des outils d'assistance. En effet, il ne dispose pas de règles systématiques pour la propagation, ni même de recommandations pour construire des systèmes d'information robustes et faciles à maintenir. De même, au niveau des outils, l'évolution et la maintenance sont des processus souvent ignorés. Ces outils permettent de construire un modèle conceptuel, de le transformer en modèle logique (opération automatisée) et de générer un prototype. Mais les niveaux logique et physique sont très largement insuffi-

sants et le code généré doit souvent être remanié de manière significative par le concepteur. Toute modification des spécifications entraînera la modification du modèle conceptuel, la transformation en modèle logique et la production de code. Bref, on reconstruit une nouvelle application sans correspondance avec la version précédente.

Nous examinerons les principes du pilotage de l'évolution et de la maintenance des applications de bases de données dans l'atelier DB-MAIN.

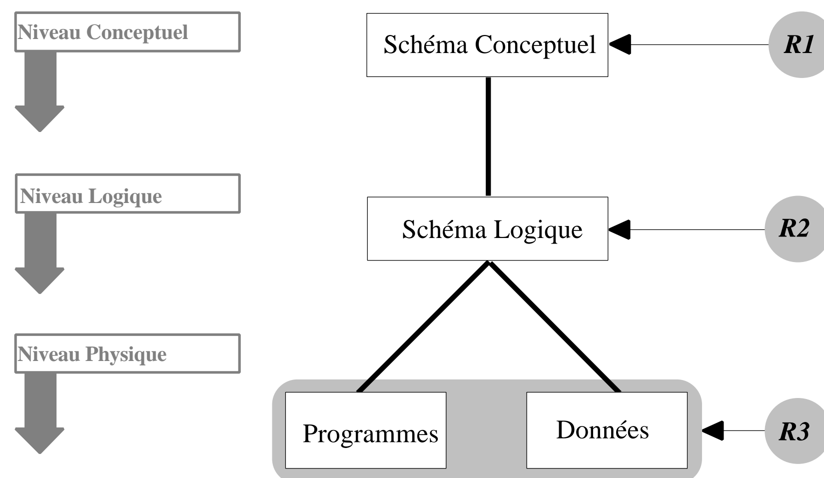

Figure 6 - Modélisation classique définie en trois niveaux d'abstraction. Le schéma conceptuel répond aux besoins R1, le schéma logique aux besoins R2 et l'application aux besoins R3.

7.2. Evolution de bases de données : l'approche DB-MAIN

Nous avons mis l'accent sur le manque de support méthodologique et logiciel au niveau de la maintenance. Les problèmes méthodologiques ne font pas partie des préoccupations de cet article (voir [HAINAUT,94]). Toutefois, un bon moyen pour le concepteur de faciliter la maintenance de l'existant reste la conception dès le départ d'une application bien documentée en prévision d'une maintenance souple. Pour ce faire, la modélisation des processus est une solution permettant la guidance des analystes dans une méthodologie définie par l'organisation (cfr. section 5).

Le problème de l'assistance technique du concepteur est un des soucis majeurs de la plateforme DB-MAIN. En reprenant les types de problèmes exposés au point 7.1., on analysera l'existant de l'outil ainsi que des pistes futures.

Dans un *premier scénario*, on suppose que tous les niveaux de la modélisation d'une application existent (SC0 est le schéma conceptuel, SL0 le schéma logique, P0 et DO les programmes et les données du niveau physique). Nous avons également enregistré toutes les opérations appliquées à SC0 pour obtenir SL0. Cet enregistrement constitue l'historique du processus de production de SL0 à partir de SC0 (on le désigne par *Histo*). Ensuite, les besoins R1 (auxquels répond le schéma SC0) évoluant, ils se transforment en R1'. Ce changement est traduit par l'analyste en modifications de SC0, qui devient ainsi SC1 (cfr. figure 7a). On peut utiliser

l'historique *Histo* pour transformer SC1 en SL1 : en "rejouant" *Histo* à partir de SC1, on obtient SL1 proche de SL0 mais dans lequel les modifications de SC1 ont été propagées[9]. Actuellement, il existe dans DB-MAIN une fonction d'enregistrement sous la forme d'historique des opérations du concepteur sur un schéma (Log) ainsi qu'une fonction permettant de "rejouer" un historique de manière automatique ou contrôlée (Replay).

P0 et D0 doivent aussi tenir compte des nouvelles spécifications. La conversion des données est une tâche qui peut être automatisée. On envisage la génération de convertisseurs sous la forme de scripts DDL ou JCL, ou encore de programmes dans les situations plus complexes. Les générateurs de convertisseurs sont des applications Voyager (section 8.3).

Modifier les programmes est un problème beaucoup plus complexe, et pour l'instant non automatisable. En effet, il suppose que la spécification formelle abstraite de tout programme puisse être retrouvée, ce qui n'est pas encore le cas, sauf pour des algorithmes élémentaires. Nous proposons donc le développement d'un outil d'annotation des programmes qui signalerait au programmeur les sections de code à modifier, et qui lui indiquerait la nature des modifications à effectuer.

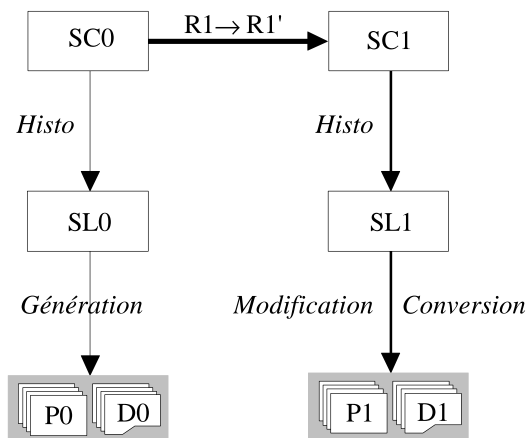

Figure 7a - Propagation vers l'aval des modifications déduites de (R1→R1') et apportées à SC0 .

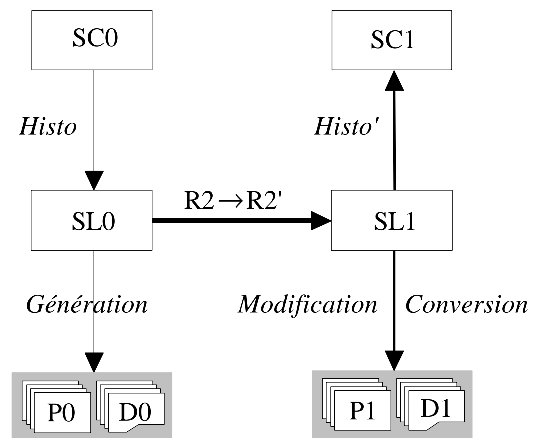

Figure 7b - Propagation vers l'amont et vers l'aval des modifications déduites de (R2→R2') et apportées à SL0.

Dans un *deuxième scénario*, on suppose qu'on apporte des modifications à SL0 (par exemple ajouter une colonne à une table) suite au changement des besoins R2 en R2' (cfr. figure 7b). Nous allons détailler la propagation en amont puisque la propagation vers le niveau physique a déjà été discutée. Dans un premier temps, on suppose que le contexte est le même que dans le premier scénario : on dispose d'un historique (*Histo*) contenant les transformations réalisées pour passer de SC0 à SL0. D'autre part, on sait que les transformations conceptuel/logique sont pour la plupart réversibles. On désigne par *Histo'* le scénario à jouer pour remonter de SL1 à SC1. *Histo'* est l'inverse de *Histo* (SC0 vers SL0) c'est-à-dire un historique où la séquence a été inversée et où chaque transformation est remplacée par son inverse. Cette inver-

[9] Cette vue est un peu simpliste, et doit être complétée par la prise en compte de l'ajout et du retrait de concepts. Il en sera de même du deuxième scénario.

sion d'historique permet de trouver SC1 proche de SC0 avec les modifications de SL1 propagées.

Dans un *troisième scénario*, SC0 et/ou SL0 manquent. Avant d'appliquer un des deux scénarios précédents, il faut retrouver les schémas logique et physique ainsi qu'un historique possible à partir de P0 et D0. Il s'agit donc d'un problème de rétro-ingénierie (cfr. section 6).

8. Personnalisation et extensibilité fonctionnelle de l'atelier

Sur bien des points, un AGL ne pourra satisfaire toutes les attentes tant de l'utilisateur que de l'ingénieur méthode. Par exemple, les règles de constitution des noms, les structures conceptuelles admises ou les règles de génération SQL diffèrent d'une entreprise à l'autre. Il est donc illusoire de tenter de développer un atelier complet et donc fermé. C'est pourquoi nous avons inclus dans différents processeurs des possibilités de personnalisation, soit par paramétrage, soit par le développement de fonctionnalités complètes. Nous présentons dans la suite trois outils de personnalisation de l'atelier: le langage de *pattern*, les *scripts* des assistants et le langage Voyager 2.

8.1. Recherche de *patterns*

Via son analyseur interactif, DB-MAIN offre une fonction de recherche de *patterns*. Ces *patterns* sont exprimés dans un langage de description de *patterns* (PDL) dérivé de la notation BNF et qui dispose de variables qui prennent une valeur pour chaque instantiation du *pattern* dans un texte. L'une des applications les plus simples est la recherche de correspondances entre variables structurées dans un programme, via les flux de données, et la recherche de requêtes SQL suggérant des contraintes d'intégrité non traduites (par exemple les clés étrangères). Le pattern `joint` montre la définition d'un pattern qui permet de retrouver les requêtes SQL qui font la jointure entre deux tables :

```
joint ::= "from" - @alias1 - @table1 ~ "," ~ @alias2 - @table2 -
          "where" - @alias1"."@col1 ~ "=" ~ @alias2"."@clo2 ;
```

avec `@alias1`, `@alias2`, `@table1`, `@table2`, `@col1` et `@col2` des variables qui seront instanciées lors de la recherche, les variables doivent aussi être définies comme des patterns. Les deux occurrences de `@alias1` (resp. `@alias2`) ont toutes les deux la même valeur. - représente un ou plusieurs espaces obligatoires, tandis que ~ représente des espaces facultatifs. On trouvera un exemple complet dans [HAINAUT,95].

Cet outil de recherche peut être utilisé pour effectuer une analyse fine de textes sources et des descriptions textuelles des objets du référentiel. La recherche permet une inspection visuelle des textes ou, par couplage avec des fonctions développées en Voyager 2 (voir ci-après), de déclencher des actions sur le référentiel. L'analyste peut donc développer une base de connaissances relative à la phase d'extraction de structures de données en rétro-ingénierie.

DB-MAIN dispose aussi d'un processeur de nom qui permet de transformer les noms d'objets sélectionnés du référentiel. Les règles de transformation peuvent être sauvées sous la forme de scripts.

8.2. Les assistants

Un assistant est un outil de haut niveau dédicacé à la résolution de problèmes spécifiques. DB-MAIN offre aujourd'hui deux assistants, mais d'autres sont en développement :

- L'assistant de transformation permet d'appliquer une ou plusieurs transformations aux objets sélectionnés. Il est structuré selon une approche problème/solution, dans laquelle un problème est défini par des préconditions et une solution par une action qui élimine le problème.
- L'assistant d'analyse permet d'analyser la conformité d'un schéma à un sous-modèle. Un sous-modéle est représenté par une expression booléenne composée de prédicats élémentaires qui expriment quels sont les objets valides.

Chacun de ces assistants permet l'écriture de scripts personnalisés. L'analyste peut ainsi développer des fonctions de résolution de problèmes spécifiques à une démarche donnée, soit sous la forme de scripts de transformation, soit sous le forme de sous-modèles.

8.3. Voyager 2

DB-MAIN est une plate-forme de base qui ne peut assumer à elle seule toutes les tâches rencontrées dans le cycle de vie d'une application. En effet, elle est d'une part souvent utilisée conjointement avec d'autres produits complémentaires (AGL, dictionnaires de données, SGBD, 4GL, etc) et présente, d'autre part, des lacunes vis-à-vis des attentes de l'ingénieur méthode. Dès lors, il s'avère que pour assurer la viabilité d'un outil, il est nécessaire de lui adjoindre un moyen de personnalisation tel un environnement de programmation permettant l'ajout dynamique de nouvelles fonctionnalités. Non seulement l'outil pourra communiquer avec d'autres, mais l'utilisateur pourra encore développer, et inclure dans l'atelier, des fonctions qui lui sont personnelles. C'est sur base de cette constatation que nous avons défini le langage et l'environnement de programmation Voyager 2.

Voyager 2 allie puissance et simplicité pour rendre la personnalisation de l'outil aisée comme le montre ses caractéristiques:

- *faiblement typé* : ce mécanisme accentue l'indépendance du langage vis-à-vis du référentiel utilisé dans DB-MAIN. Par ailleurs, la définition du référentiel étant orientée objet, le langage supporte le typage dynamique.
- *procédural* : le langage supporte les procédures/fonctions avec appels récursifs.
- *requêtes prédicatives et navigationnelles* : des requêtes concises et efficaces rendent aisées aussi bien la consultation que la modification du référentiel.
- *définition d'un type liste avec Garbage Collection* : les listes peuvent contenir des informations de tout type et leur gestion est assurée par un Garbage Collector. Elles sont utilisées aussi bien comme arguments que comme structures hôtes pour les résultats des requêtes.
- *analyseur lexical* : le langage permet d'écrire très rapidement des fonctions d'importation de fichiers textes par l'utilisation d'expressions régulières.
- *accès aux outils de l'atelier* : chaque fonctionnalité intrinsèque de l'atelier (outils de base) est accessible dans le langage.
- *enrichissement de l'atelier avec des fonctionnalités externes* : les fonctions et procédures d'un programme Voyager 2 peuvent être invoquées directement dans l'atelier.

D'autre part, le fait que les programmes *Voyager 2* soient précompilés en code binaire directement exécutable par la machine virtuelle VOYAGER (Figure 1) rend impossible la distinction de fonctionnalités écrites en code natif (C++) de celles écrites en *Voyager 2*. Ceci nous a encouragé à préférer ce langage pour développer et à distribuer certaines fonctionnalités de l'atelier que l'utilisateur pourrait désirer aménager selon ses besoins. Dans cette optique, il ne nous paraissait pas opportun de développer un générateur SQL pour chaque architecture existante sur le marché. C'est pourquoi le générateur SQL standard, initialement développé en C++, a été aussi rédigé en *Voyager 2*. Ce programme peut aisément être modifié et recompilé afin d'étendre DB-MAIN avec un nouveau générateur spécifique à un SGBD et aux standards de génération propres à une entreprise.

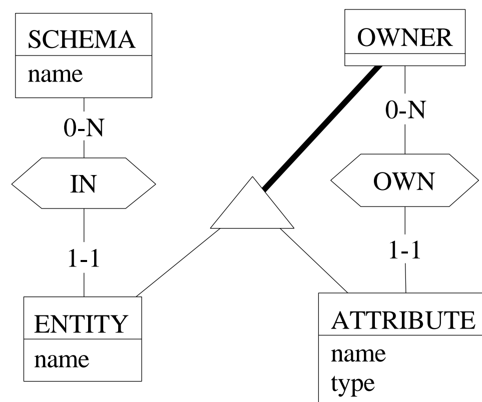

Figure 8 - Extrait fortement simplifié du référentiel utilisé dans DB-MAIN dont la sémantique est la suivante : un schéma possède des types d'entités composés d'attributs. Les attributs peuvent à leur tour être composés d'autres attributs. La relation *is-a* vers OWNER permet d'exprimer qu'un type d'entités et un attribut peuvent être décomposés en attributs.

L'exemple de la figure 9 illustre l'emploi d'un programme *Voyager 2* pour étendre DB-MAIN avec des fonctions statistiques. Ce programme utilise l'extrait du référentiel (Figure 8) pour fournir des informations sur le nombre de types d'entités et la profondeur maximale des attributs composés. La procédure *stat* est déclarée *export* (ligne 2) de sorte qu'elle soit visible dans l'atelier; elle apparaît sous forme d'un nouvel item dans un menu. La boucle d'itération (ligne 7) dans cette fonction parcourt tous les types d'entités du schéma ouvert (GetCurrentSchema) au moment de l'appel dans l'atelier. Pour chaque type d'entités, la fonction *prof* retourne la profondeur maximale des attributs du type d'entités. Les lignes 7, 12 et 19 montrent l'utilisation de constantes du type liste utilisées dans les boucles, comme arguments et dans les requêtes.

```
1:	/* Cette fonction peut être invoquée directement à partir de l'atelier */
2:	export procedure stat()
3:		entity: e;
4:		integer: ne, namax;
5:	{	ne:=0;
6:		namax:=0;
7:		for e in ENTITY[e]{IN:[GetCurrentSchema()]} do {
8:			/* pour chaque entité e du schéma courant */
9:			ne:=ne+1;
10:			namax:=max(namax,prof(e));
11:		};
12:		print(["entités:",ne,"\nprofondeur:",namax,'\n']);
13:	}

14:	/* retourne la profondeur maximale des attributs qui composent l'objet o */
15:	function integer prof(owner: o)
16:		integer: max_temp;
17:		attribute: a;
18:	{	max_temp:=0;
19:		for a in ATTRIBUTE[a]{OWN:[o]} do {
20:			/* pour chaque attribut a dont e est composé */
21:			max_temp:=max(max_temp,prof(a));
22:		};
23:		return max_temp+1;
24:	}
```

Figure 9 - Un programme *Voyager 2*. Ce programme comporte une procédure exportable (*stat*) qui permet d'étendre l'atelier DB-MAIN avec des fonctions statistiques.

9. Conclusion

Les aspects de l'atelier DB-MAIN présentés dans cet article permettent de répondre aux principales critiques faites aux outils actuellement sur le marché :

- Le modèle de spécification générique ne contraint pas l'utilisateur de l'outil à se plier à une méthode imposée, il peut garder une liberté totale. La personnalisation méthodologique autorise cependant d'aider l'utilisateur à suivre une méthodologie propre à la culture de son entreprise.
- Grâce à l'approche transformationnelle, l'utilisateur garde la maîtrise du processus de transformation du schéma.
- DB-MAIN fournit un certain nombre d'outils aidant l'utilisateur dans son travail de rétro-ingénierie. Ces outils autorisent l'automatisation de certaines phases du processus.
- Parce qu'il est vain de croire qu'un AGL peut répondre à toutes les attentes d'un utilisateur, des possibilités de personnalisation et d'extensibilité s'imposent tels un langage de définition de patterns, des assistants et un langage de programmation de l'atelier.

Le projet DB-MAIN a démarré en 1993, et sa première phase doit se terminer en 1997. L'atelier décrit dans cet article n'est donc qu'une première version qui ne remplit que partiellement les objectifs ultimes. Il est développé sur PC, dans l'environnement Windows. Les principales fonctions disponibles aujourd'hui (mars 1995) sont les suivantes :

- gestion de projets multi-schémas;
- saisie graphique de spécifications conceptuelles, logiques et physiques; gestion de ces spécifications; édition de rapports;
- transformations (>20) conceptuel/logique et logique/conceptuel (conceptualisation);
- visualisation des spécifications selon 4 vues textuelles et 2 vues graphiques
- extraction de schémas physiques SQL, CODASYL et COBOL à partir de leur code source;
- génération de code SQL selon différents SGBD;
- assistants (modules experts) en résolution de problèmes structurels et en analyse de spécifications; gestion de scripts;
- analyse interactive de textes sources;
- environnement Voyager : machine virtuelle et compilateur d'un noyau du langage; quelques applications industrielles dévéloppées en Voyager 2;
- journalisation des actions de l'analyste et fonction de *replay* (réexécution automatique ou assistée du journal).

A côté de son utilisation comme support pédagogique tant à l'université qu'en entreprise, cette première version a cependant déjà fait l'objet d'une expérimentation via plusieurs projets en vraie grandeur : redocumentation (ORACLE), analyse conceptuelle (comptabilité agricole), rétro-ingénierie (SYBASE, ORACLE, RPG, IDS2) et réingénierie (COBOL+SQL). La conduite de ces projets a permis non seulement une validation des principes théoriques à la base de l'atelier, mais aussi l'évaluation de ses caractéristiques d'ergonomie, de performance et de robustesse dans un contexte industriel.

Une version d'évaluation, fonctionnellement complète mais limitée à des projets de petite taille, peut être obtenue sur simple demande[10].

10. Bibliographie

ANDERS,94 M. Andersson, *Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994

HAINAUT,89 J-L. Hainaut, *A Generic Entity-Relationship Model*, in Proc. of the IFIP WG 8.1 Conf. on Information System Concepts : an in-depth analysis, North-Holland, 1989

HAINAUT,91a J-L. Hainaut, *Entity-generating Schema Transformation for Entity-Relationship Models*, in Proc. of the 10th Entity-Relationship Approach, San Mateo (CA), North-Holland, 1991

HAINAUT,92a J-L. Hainaut, M. Cadelli, B. Decuyper, O. Marchand, *Database CASE Tool Architecture : Principles for Flexible Design Strategies*, in Proc. of the 4th Int. Conf. on Advanced Information System Engineering (CAiSE-92), Manchester, May 1992, Springer-Verlag, LNCS, 1992

HAINAUT,92b J-L. Hainaut, *A Temporal Statistical Model for Entity-Relationship Schemas*, in G. Pernul et A.M. Tjoa, editors, in Proceedings of the 11th Conf. on ER Approach, Karlsruhe, Germany, Springer-Verlag, LNCS 645, octobre 1992

---

10 Adresse Email : db-main@info.fundp.ac.be

HAINAUT,93 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

HAINAUT,94 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, LNCS 881, 1994

HAINAUT,95 J-L Hainaut, V. Englebert, J. Henrard, J-M. Hick, D. Roland, *Requirements for Information System Reverse Engineering Support*, in Proc. of the IEEE Working Conference on Reverse Engineering, Toronto, IEEE Computer Society Press, July 1995

JARKE,93 M. Jarke, editor. *Database Application Engineering with DAIDA*, Springer - Verlag, 1993

PETIT,94 J-M. Petit, J. Kouloumdjian, J-F. Bouliaut, F. Toumani, *Using Queries to Improve Database Reverse Engineering*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, LNCS 881, 1994

POTTS,88 C. Potts, G. Bruns, *Recording the Reasons for Design Decisions*, in ICSE 88, 1988

PREMERLANI,93 W.J. Premerlani, M.R. Blaha, *An Approach for Reverse Engineering of Relational Databases*, in Proc. of the IEEE Working Conf. on Reverse Engineering, Baltimore, May 1993

ROLAND,95 D. Roland, *Process Modeling of Database Engineering : Principles.* Rapport technique, FUNDP Institut d'Informatique, March 1995

ROLLAND,93 C. Rolland, *Modeling the Requirements Engineering Process*, in 3rd European-Japanese Seminar on Information Modeling and Knowledge Bases, Budapest, May 1993

ROSENTHAL,94 A. Rosenthal, D. Reiner, *Tools and Transformations - Rigourous and Otherwise - for Practical Database Design*, ACM TODS, Vol. 19, No. 2, June 1994

SIGNORE,94 O. Signore, M. Loffredo, M. Gregori, M. Cima, *Reconstruction of ER Schema from Database Applications: a Cognitive Approach*, in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, LNCS 881, 1994

YONESAKI,93 N. Yonesaki, M. Saeki, J. Ljungberg, T. Kinnula. *Software Process Modeling with the TAP Approach - Tasks-Agents-Products*, in 3rd European-Japanese Seminar on Information Modeling and Knowledge Bases, Budapest, May 1993