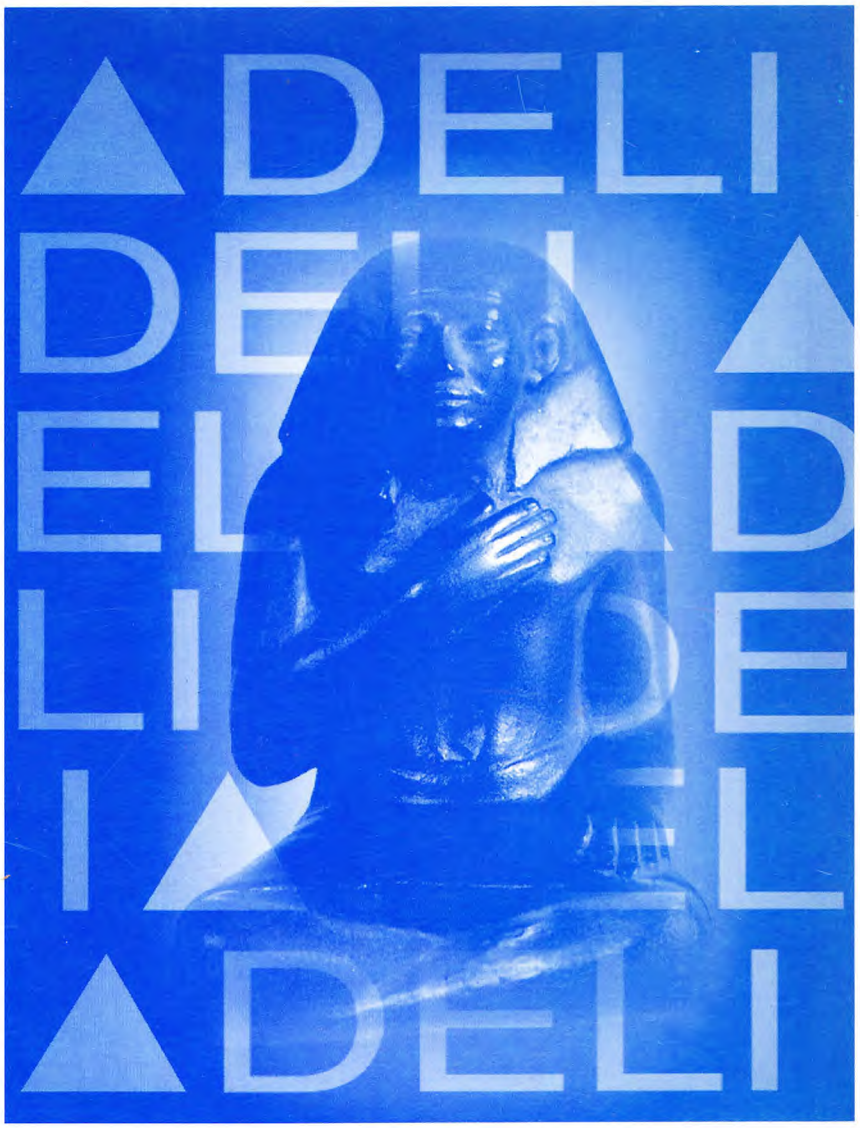
AFDEL



ASSOCIATION FRANCAISE DE GENIE LOGICIEL

La LETTRE n° 28

Juillet 1997

Square des Utilisateurs

Après l'an 2000... l'EURO

Les dernières années du vingtième siècle informatique seront marquées par deux événements.

Le premier, c'est le changement du nombre des centaines de nos dates : 20xx remplacera 19yy. C'est inéluctable, les règles de gestion sont impératives et incontournables.

Le second événement est d'origine politique. C'est le passage à la monnaie unique pour les pays européens qui rempliront certaines conditions économiques.

L'unité économique européenne est symbolisée par un jeu de billets de couleurs différentes. Chaque billet indique une valeur : un nombre suivi du mot EURO (grammaticalement neutre, qui ne s'accorde ni en genre ni en nombre) sur une silhouette de monument abstrait. Il n'est pas sûr que les Grecs soient admis dans la première fournée, mais la transcription en caractères grecs figure déjà sous le mot EURO.

Combien d'euro pour un franc ? Un écart de quelques millièmes de centimes sur le taux se traduit dans les organismes financiers par des millions de francs. Le président Giscard d'Estaing s'est déjà livré à un exercice de calcul mental en proposant un rapport : 7 francs pour un euro. Facile pour convertir des euro en francs ... mais pas évident pour convertir des francs en euro.

Alors, pour conduire le passage progressif des monnaies nationales à la monnaie unique européenne MUE, il va falloir gérer des règles de conversion entre ces monnaies. Il sera nécessaire de tenir, pendant la période transitoire, les écritures comptables dans les deux monnaies. Il faudra respecter la fongibilité (cumul les montants convertis en euro) et la traçabilité (conservation du montant et de la monnaie d'origine de chaque écriture).

Il conviendra aussi de prévoir un retour en arrière au cas où le système se gripperait devant les difficultés levées par les oppositions politiques.

La marche vers l'euro

Un cahier des charges a été émis par la Banque de France et l'AFECEI (Association Française des Établissements de Crédit et des Entreprises d'Investissement). Vendu pour la modique somme de 95 francs, il ne comporte pas moins de 1 000 pages. Le calendrier est diffusé sur le site web (www.finances.gouv.fr).

Avant le 1er janvier 1999, on sélectionnera les pays admis à entrer dans le premier peloton de la monnaie unique selon les fameux critères de convergence de Maastricht :

déficit public inférieur à 3 % du produit intérieur brut ;
dette publique inférieure à 60 % du produit intérieur brut ;
inflation pas supérieure de plus de 1,5 % à la moyenne des trois meilleurs ;
taux d'intérêt à long terme pas supérieur de plus de 2 % par rapport à la moyenne des trois meilleurs ;
respect pendant les deux dernières années des marges normales de fluctuation prévues par le SME.

On nommera le directoire de la Banque Centrale Européenne qui remplacera l'Institut Monétaire Européen.

Puis s'ouvrira la période de porosité par le basculement des systèmes informatiques le 4 janvier 1999 et la mise en service du système européen Target (transfert express automatisé à règlement brut en temps réel) pour les montants supérieurs à 5 MF.

Au cours de la période transitoire, de 1999 à fin 2001, les parités fixes seront définies de façon irrévocable. Les banques commerciales basculeront la totalité de leurs activités en monnaie unique et l'on fabriquera les billets et les pièces de monnaie. Les monnaies nationales deviendront des fractions d'euro.

Le SEBC (Système Européen des Banques Centrales) aura les rôles suivants :

définir et mettre en œuvre la qualité monétaire de l'Union Monétaire ;
conduire les opérations de change ;
définir et gérer les réserves officielles des changes des états membres ;
promouvoir le bon fonctionnement des systèmes de paiement.

Le 1er janvier 2002 l'euro sera émis sous forme de billets qui se substitueront progressivement aux monnaies nationales. Le 1er juillet 2002 les francs auront définitivement vécu et s'effaceront devant l'euro. De commune, l'euro sera devenue monnaie unique pour ceux qui l'auront adoptée.

Pour faciliter la mise en place de l'euro quelques mesures sont envisagées :

obligation du double affichage en francs et en euro ;
gratuité des opérations de conversion ;
continuité des contrats ;
maintien des déclarations fiscales en francs ;
taux de conversion (nombre de chiffres significatifs) ;
identification normalisée des monnaies.

Les avantages de l'euro

Pour les individus

La mondialisation de l'économie est un phénomène récent dont on peut attendre un accroissement du bien-être général selon nos critères conventionnels (revenu moyen par habitant).

Il sera pratique d'utiliser la même monnaie dans toutes ses transactions à l'étranger sans s'embarrasser de la variation des taux de change et de la gestion d'un stock de petites coupures dans les différentes monnaies étrangères. Cette continuité monétaire, associée à la suppression des contrôles aux frontières et à l'uniformisation des consommations, donnera l'impression d'être citoyen d'un même pays : l'Europe.

Pour les entreprises

Les avantages seront évidemment plus sensibles pour les entreprises qui pourront affronter un grand marché géographiquement et monétairement homogène. Ces facilités encourageront les entrepreneurs et dynamiseront l'innovation.

Ce sont surtout les entreprises multinationales qui bénéficieront de ce nouveau paysage monétaire auquel elles sont déjà accoutumées. Citons, parmi les premiers intéressés : opérateurs de télécoms, vendeurs par correspondance, commerce électronique.

Pour les États

L'euro est l'instrument d'une politique qui désigne le nouveau rôle dévolu aux États-nations aux côtés des acteurs économiques.

Les nouveaux acteurs économiques n'attendent plus des États qu'ils jouent un rôle de consommateur de produits et de services, par l'intermédiaire de leurs services publics, ni de producteur en s'impliquant directement dans la création de produits et de services stratégiques.

Dans le nouvel ordre économique mondial, chaque État doit maintenir, dans sa zone de souveraineté, les conditions du développement ; en particulier il doit veiller au maintien d'une paix sociale nécessaire à l'industrie et au commerce.

Les États doivent fournir les conditions d'accueil : infrastructures, compétences, climat fiscal, etc. pour attirer les investissements dans leur pays et ainsi privilégier l'emploi et la consommation sur leur territoire.

Un pacte européen de stabilité veillera au maintien de la neutralité des États et les empêchera d'être tentés de fausser la concurrence en favorisant leurs entreprises nationales.

Une opportunité pour les prestataires de service

Les médias ont déjà fixé le montant de cet énorme marché du passage à l'euro. Il serait de l'ordre de 20 milliards de francs français.

Ce montant inclut :

la création de nouvelles applications ou la refonte d'anciennes applications ;
la modification d'applications existantes ;
la mise au rebut d'anciennes applications obsolètes.

Sous l'égide du Ministère de l'Économie et des Finances, la mission Euro, en concertation avec le Syntec Informatique, vient d'éditer un guide « Le passage à l'euro, questions informatiques, la démarche informatique ».

Ce guide fournit une démarche méthodologique : approche globale, étude d'impact, plan d'action, mise en œuvre.

Mais un frein pour les systèmes d'information stratégiques

Comme de nombreux observateurs, Jean-Marie Descarpentries, Président Directeur Général du groupe Bull, dans ses conférences, souligne le retard croissant que prennent les pays européens sur les entreprises américaines, dans le domaine des systèmes d'information stratégiques, ces systèmes d'information centrés sur les clients qui décuplent l'efficacité des entreprises.

Aux États-Unis, l'investissement informatique est de 3,1 % du produit intérieur brut. En Europe, il n'est que de 1,7 %. Ce retard, qualifié de dramatique (sans doute en adoptant ce faux-ami **dramatic** qui en anglo-américain signifie **spectaculaire,** sans atteindre le tragique du mot français) s'accroît ; il est passé de 22 % en 1990 à 44 % en 1995.

Alors que l'investissement en système d'information est de 6,2 % du chiffre d'affaires aux États-Unis, celui de l'Europe n'est que de 4,3 %. Le taux américain bondira à 8,2 % en 2001.

Que sera, alors, le taux européen ?

Pendant que les entreprises étrangères (américaines, japonaises, etc.) vont poursuivre la domestication de leurs systèmes d'information au service de leurs clients, les européens vont devoir ralentir leurs applications stratégiques pour s'adonner à la mise en équation des passerelles entre monnaies.

Cet effort européen du passage à l'euro peut-il être considéré comme un investissement productif qui portera ses fruits dans plusieurs années ?

L'avenir nous le dira. ▲

Alain Coulon

Quelques références

L'euro passera-t-il à la période transitoire, par William Vantien
Logiciels&Systèmes n° 17 d'avril 1997

L'euro, c'est parti, par Colette Sauvant
01 informatique n° 1444 du 14 mars 1997

L'échéancier de l'euro, par Charles de Laubier
Le Monde informatique n° 713 du 14 mars 1997

Le concept de monnaie unique, par François Buonomo
La Tribune du 12 février 1997

L'euro 1997-99 - L'heure des préparatifs, par Didier Cahen
Éditions d'organisation

Ré-ingénierie des données

Objet de la solution

La ré-ingénierie des données a pour objet la re-conception et la ré-implémentation d'une collection de fichiers de données provenant de sources anciennes (en anglais *legacy* : héritage), vers en général une base de données relationnelle. Elle vise d'abord à améliorer de manière significative la qualité et l'accessibilité des données. Mais de plus en plus, elle vise à fournir des données ou des bases de données qui répondent à de nouvelles exigences des entreprises.

Celles-ci veulent en effet moderniser leur système d'information. Les traditionnelles applications sur ordinateurs centraux ont évolué et grossi durant ces dernières décennies. Elles mettent en œuvre des milliers de fichiers de données et des millions de lignes de code. Elles ont subi des corrections sans que les documentations n'aient été mises à jour. Les entreprises veulent maintenant des systèmes qui reposent sur des technologies modernes et flexibles.

La ré-ingénierie des données est donc une nécessité nouvelle. Elle devient fondamentale compte tenu du rôle vital des données et du volume qu'elles représentent.

La solution proposée dans le présent document comprend :

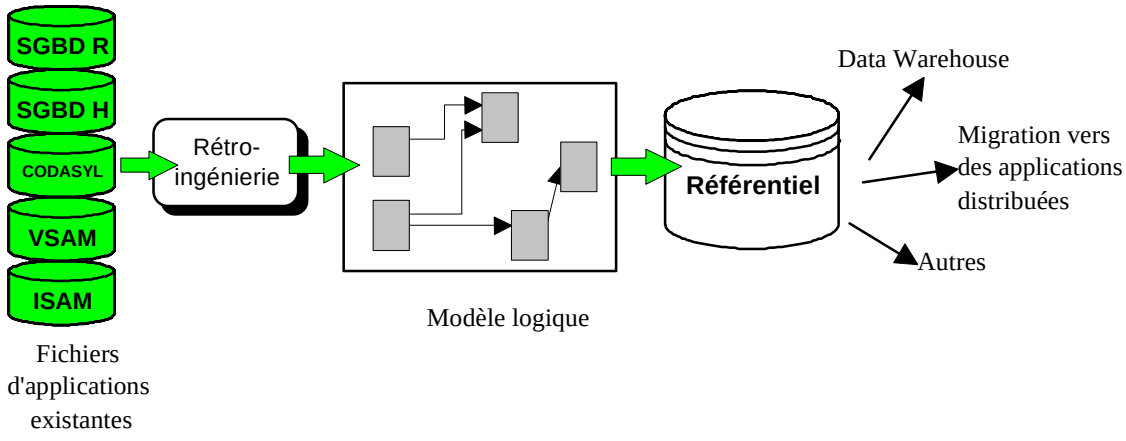
- une phase de rétro-ingénierie de bases de données adaptée aux données anciennes et hétéroclites ;
- une phase de consolidation à l'aide d'un référentiel. Celle-ci conduira à la capitalisation des méta-données (les données descriptives sur les données).

Une telle solution répond à des besoins tels que :

- la migration des parcs applicatifs vers des applications distribuées ou vers des progiciels,
- la consolidation et l'intégration des données,
- l'intégration des applications,
- la construction et la maintenance d'un Data Warehouse.

Champ de la solution proposée

La solution proposée a été conçue pour la ré-ingénierie de sources de données anciennes pour lesquelles on ne dispose pas de modèle logique (ni a fortiori de modèle conceptuel).



Elle est particulièrement adaptée aux situations où les applications présentent une complexité difficilement maîtrisable. Par exemple :

- lorsque les applications ont été développées sans coordination ;
- lorsque les codes source sont incomplets ou ne sont plus à jour. Les équipes de développement ne sont plus présentes et de ce fait, la connaissance sur les données a été en grande partie perdue ;
- lorsque les données sont fragmentées et disséminées dans de nombreux fichiers hétéroclites et supportés par des technologies très variées (fichiers plats programmés en COBOL, bases de données hiérarchiques, navigationnelles ou relationnelles).

Pourquoi cette solution

Il est nécessaire de comprendre les bases existantes

La re-conception des données nécessite de bien connaître leur structure à des niveaux très fins. Toutefois cette opération est rendue complexe et difficile.

En effet, les applications ayant été développées sans coordination, chacune d'elles comporte des définitions de données qui lui sont propres. De ce fait, les données présentent des redondances, des incohérences ou des ambiguï tés. Des données similaires peuvent comporter des noms différents, ou même être décrites avec des types différents. Pour être exploitables, elles doivent être rationalisées : les synonymes et homonymes doivent être détectés, les synonymes doivent être fusionnés et les homonymes doivent être séparés.

Il n'est pas seulement nécessaire de rationaliser les noms et les types de données. Il faut également comprendre les relations structurelles entre elles si on veut les exprimer selon une formulation relationnelle à l'aide de clés primaires, de clés étrangères ou autres contraintes d'intégrité.

Or celles-ci ne sont pas explicites dans les sources de données origine. Dans les fichiers plats programmés en COBOL, les bases de données hiérarchiques ou navigationnelles, on a plutôt l'habitude de manipuler, à la place des index, des pointeurs ou des procédures logiques.

Ceux-ci n'apparaissent pas explicitement dans les Copy Book COBOL ou les DDL qui décrivent les données. Ils sont en général noyés dans les codes sources des programmes. Aussi, les traditionnelles analyses, basées sur les codes sources, même si elles sont nécessaires, ne peuvent pas permettre de récupérer ces relations structurelles.

De plus, le transfert des données sources vers la base de données cible relationnelle nécessite de construire les correspondances entre les anciennes structures et le modèle logique de la base de données cible.

En conséquence, il est nécessaire d'effectuer la rétro-ingénierie des modèles de données. Elle doit permettre de reconstituer les règles qui régissent les données et les dépendances entre elles.

La difficulté de l'analyse manuelle

Dans ce contexte, une résolution manuelle s'avère hasardeuse.

En effet, la rationalisation des données et la résolution des dépendances multiples et complexes mettent en jeu de nombreux fichiers non structurés et éparpillés. Elle sera longue et coûteuse. Elle mobilisera de nombreux analystes et spécialistes des bases de données. De plus, elle comportera des risques d'erreurs préjudiciables.

Il est donc nécessaire de disposer de techniques et d'outils d'aide à l'analyse des données qui permettent de les rationaliser et de constituer de nouveaux modèles de données.

La phase de rétro-ingénierie de bases de données

Elle a pour objet d'automatiser la découverte de la structure des bases de données et de constituer un modèle logique. Elle permet ensuite l'exploration et la manipulation de ce modèle par un concepteur pour obtenir un modèle cible (sous forme relationnelle) en fonction des objectifs de l'entreprise.

Le processus mis en œuvre rend par la suite plus aisée la conversion des données sources vers la base de données cible à l'aide d'outils tiers.

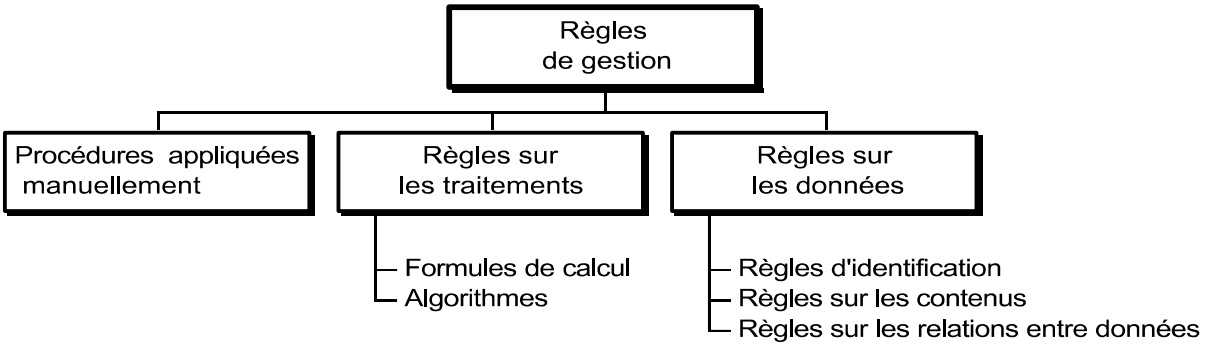
L'analyse des données par les valeurs des occurrences

Contrairement aux techniques de rétro-ingénierie qui d'habitude opèrent à partir du code source, celle que nous proposons ici utilise le contenu des fichiers ou des bases de données comme source première de connaissance des systèmes applicatifs existants.

Elle est basée sur une technique originale d'inférence qui effectue des analyses sur les valeurs de ces données afin de découvrir les règles de dépendance cachées.

Les règles sur les données

Les systèmes d'information renferment des règles qui décrivent les décisions de gestion que l'entreprise a adoptées. Elles sont de différents types :



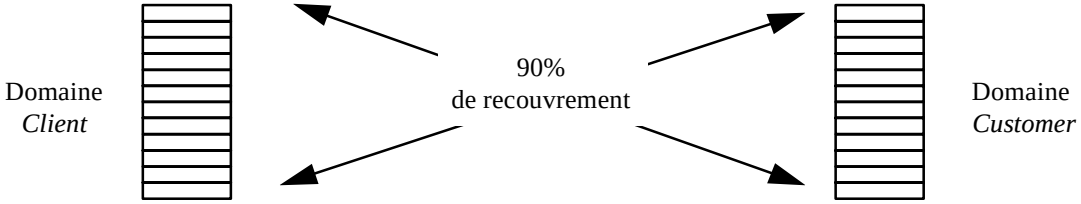
Les règles sur les traitements sont contenues dans les applications sous forme d'algorithmes comportant des instructions 'if-then-else' ou des formules de calcul.

Les règles sur les données, explicites dans les bases de données relationnelles, sont par contre implicites dans les fichiers plats ou les bases de données plus anciennes.

La démarche proposée consiste à découvrir les règles sur les données. Pour cela, elle effectue sur les données une analyse tri-dimensionnelle :

- **1)** Analyse de toutes les valeurs de chaque colonne afin d'identifier le domaine de valeurs de chaque champ de donnée. Les informations identifiées sont :
  - Les types de données
  - Les plages de valeurs
  - Les éventuelles listes de valeurs de données autorisées
  - La possibilité qu'une occurrence puisse être NULL
- **2)** Analyse deux à deux des colonnes d'un même fichier afin de découvrir les règles de dépendance fonctionnelle entre elles. Une telle règle est décrite par un ou plusieurs déterminants et au moins un dépendant. Il y a dépendance fonctionnelle si, pour une combinaison de valeurs des déterminants correspond une valeur unique d'un dépendant. Ces règles permettront d'identifier les clés candidates pour les futures clés primaires.
  - N° de commande -> Client
  - N° d'article -> Fournisseur
  - N° de commande , N° d'article -> Client, Fournisseur**,** Quantité

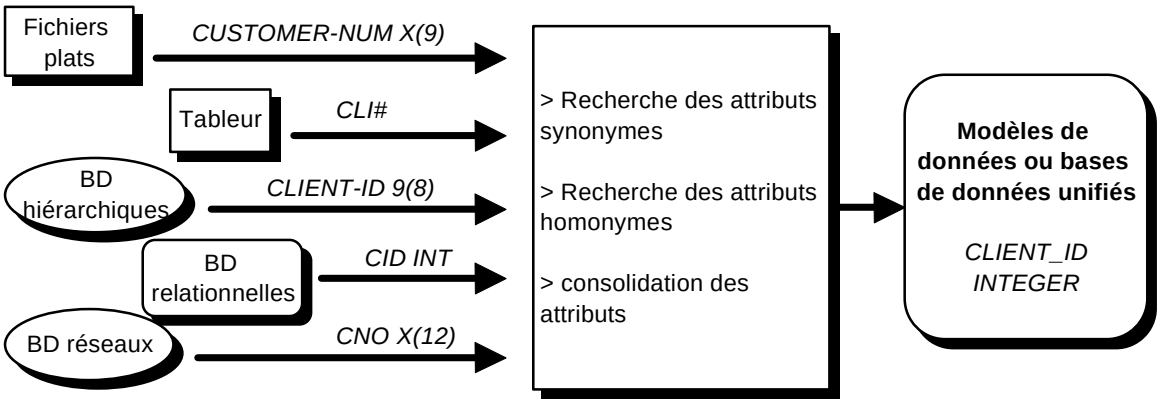
- **3)** Comparaison croisée des valeurs des domaines entre les tables afin d'identifier ceux qui sont similaires.



Elle permet, par l'analyse du recouvrement, de déduire les synonymes et les homonymes potentiels. Elle permet également de mettre en évidence les éventuelles incohérences sur les domaines de valeurs.

La rationalisation des données

Ainsi, la découverte des règles sur les données permet de mettre en évidence les données identiques. Celles-ci peuvent avoir été décrites au départ avec des noms différents, des types différents ou des méthodes d'accès différents comme l'illustre le schéma suivant :



Le résultat : un modèle représentant les bases existantes

Après le travail d'analyse sur les données, l'étape de rétro-ingénierie reconstitue un modèle logique qui représente la collection de données physiques. Il incorpore toutes les règles de données. Il représente donc la manière dont les données sont stockées et reliées entre elles. Il s'exprime selon la notation relationnelle.

La rétro-ingénierie est en général conclue par la transformation du modèle en 3ème forme normale qui sera le point de départ de toute nouvelle conception de base de données.

Toutefois, la constitution du modèle logique peut être un objectif en soi. Elle permet déjà de comprendre les données et d'orienter les prises de décision.

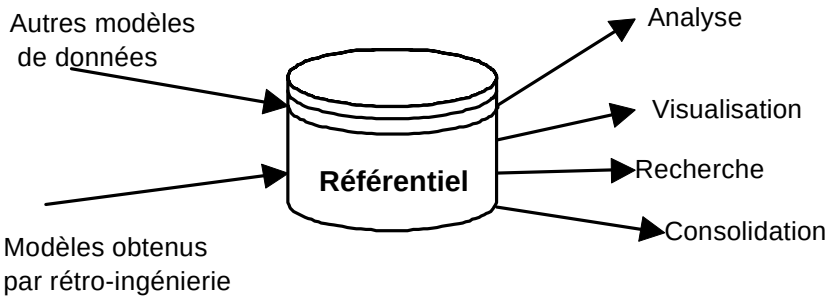
Intégration avec le référentiel

Pourquoi un référentiel

La technologie de rétro-ingénierie que nous avons exposée permet d'obtenir un modèle pour chacune des bases de données prise isolément. Elle s'avère insuffisante pour obtenir une vision globale des données du système d'information, ou même d'un secteur d'activité particulier, ce qui pourtant peut s'avérer nécessaire pour une ré-ingénierie ou un Data Warehouse efficace.

De plus, d'autres modèles de données auront été développés avec des outils modernes de conception pour des bases de données plus récentes.

On cherchera donc à mettre ensemble les modèles disponibles quelque soit leur source et de les rapprocher dans le but d'obtenir des visions plus globales ou contextuelles des données. Cette opération est nécessaire si on veut pouvoir les interpréter correctement et les réutiliser. C'est bien le rôle d'un référentiel.



Rôles du référentiel

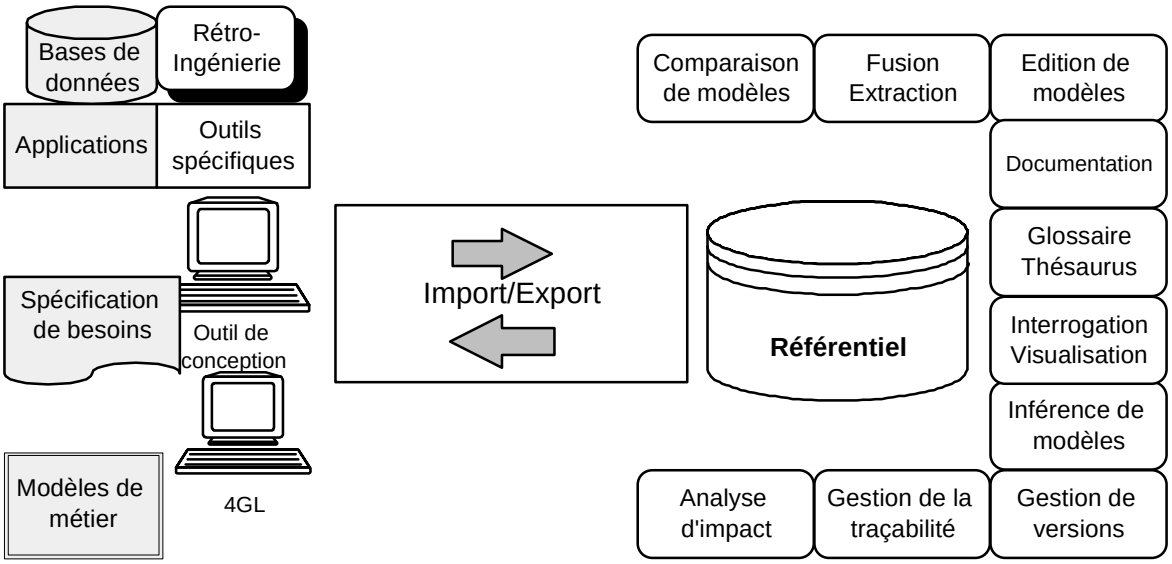
Le référentiel est un environnement d'administration de données qui doit permettre de :

- gérer l'inventaire des sources de données,
- capturer les modèles de données au fur et à mesure qu'ils sont disponibles (par la rétro-ingénierie ou par une autre source),
- les organiser, les consolider, les documenter,
- effectuer des interrogations, des recherches, des analyses, les visualiser sous forme de cartographie,
- les mettre à la disposition des différents acteurs du système d'information (utilisateurs, responsables de développement, responsables d'exploitation).

Le référentiel devient alors le moyen de communication entre les acteurs. Il assure une communication cohérente et permet d'éviter les confusions, les ambiguï tés, les équivoques.

Ainsi, le référentiel se positionne dans la solution décrite comme un outil d'intégration après la technologie de rétro-ingénierie, pour contribuer à la ré-ingénierie du système d'information et pour intégrer les besoins nouveaux liés à l'évolution des métiers, à la stratégie de l'entreprise ou aux technologies nouvelles.

Les grandes fonctions du référentiel



Import/Export

Le référentiel doit offrir une fonction de transfert et de traduction qui permette, d'une part de capturer les données produites par des outils de génie logiciel hétérogènes, d'autre part d'exporter les données vers d'autres outils de conception, des outils de génération ou des SGBD. Cette fonction d'interopérabilité permet ainsi la coopération des outils par l'échange des méta-données.

Administration des méta-données

Le référentiel doit offrir les fonctions qui permettent à une organisation informatique d'intégrer ses méta-données quelque soit leur origine, et de constituer de proche en proche un modèle d'entreprise.
Il doit permettre la gestion de l'information de manière uniforme quelque soit son origine, offrir des fonctions d'organisation et de classement qui permettent à un utilisateur de retrouver facilement des méta-données, ainsi que des fonctions de contrôle de cohérence et de consolidation telles que la comparaison, la fusion, ou l'extraction de modèles.

Consultation et utilisation

Les utilisateurs disposent de fonctions puissantes de consultation et de recherche.
Le référentiel doit fournir les fonctions de :

- Édition de modèles,
- Documentation des modèles et des objets permettant de gérer les informations descriptives (définition, libellés) et administratives (auteur, dates, droits d'accès, ...),
- Glossaire Thesaurus qui permettent la recherche de méta-données par ordre alphabétique ou selon des mots-clés,
- Navigation dans l'organisation des méta-données,
- Visualisation graphique de modèles,
- Interrogation en grande masse par construction de requêtes,
- Inférence permettant de construire des modèles logiques à partir de modèles conceptuels,
- Édition de rapports,
- Archivage de sécurité et d'intégrité.

L'information doit toujours être présentée de manière homogène, indépendante des méthodes ou des outils qui ont servi à la produire.

Gestion des évolutions

Les méta-données d'une entreprise doivent accompagner les évolutions des métiers, de la réglementation, ou simplement du système d'information.

Le référentiel doit permettre le maintien de la cohérence de l'ensemble des méta-données lors des évolutions. Il fournit des fonctions de :

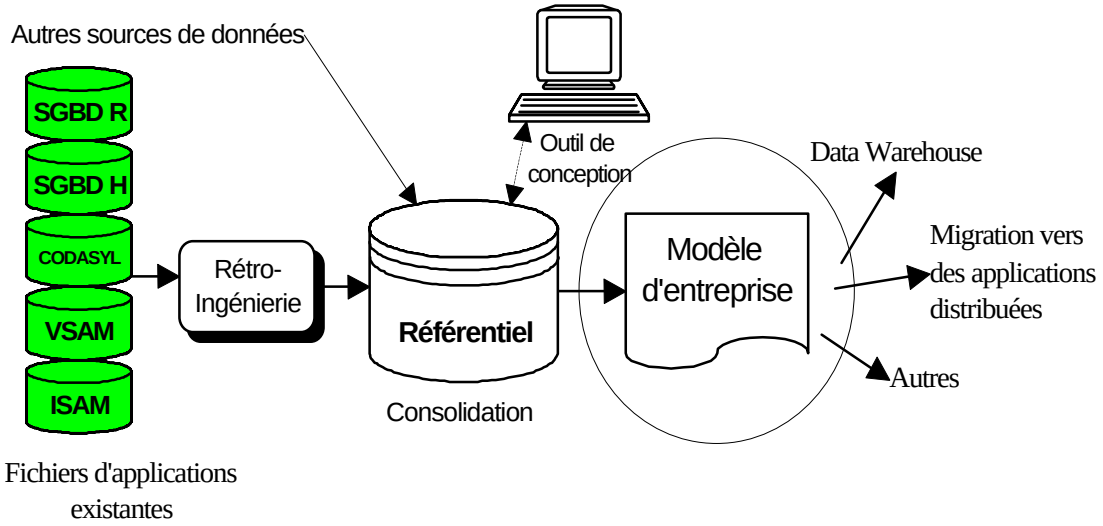
- Gestion de versions et de configurations,
- Gestion de la traçabilité, permettant de maintenir les relations de dépendance ou de dérivation entre les objets,
- Analyse d'impact pour déterminer les méta-données concernées par une éventuelle modification.

La consolidation des données de l'entreprise

Les entreprises réalisent que l'évolution de leur système d'information est rendue plus complexe par la fragmentation des données et leur morcellement en centaines de fichiers. Chacun d'eux a été développé dans le contexte d'une application spécifique, à l'aide d'une technologie spécifique, à une époque donnée, et souvent par des équipes différentes. Les éléments de données qui représentent la même information portent des noms différents à différents endroits, et aucune correspondance n'est disponible. L'entreprise n'est même pas en mesure d'évaluer l'ampleur de la redondance sur les données.

Pour aplanir cette complexité, les entreprises décident alors, à partir des applications existantes, de construire un ensemble de modèles représentatifs de leur système d'information, qui sera le modèle d'entreprise.

La solution proposée permet à l'entreprise de constituer et de gérer ce modèle de données. Cette opération sera effectuée de manière progressive en couvrant les métiers les uns après les autres.



- La rétro-ingénierie produit au fur et à mesure, et à partir de chaque ensemble de fichiers locaux, des modèles qui représentent les sources de données « telles qu'elles sont ».

- Le référentiel gère et administre les modèles. Il capture également tout autre modèle de données produit avec des outils de conception dans le cadre d'applications plus récentes. Il permet de les consolider, d'établir progressivement la cohérence globale des méta-données et de les modifier.

- Un outil de conception produit éventuellement de nouveaux modèles « tels qu'ils sont recherchés » en fonction des besoins (les modèles de données cible). On constitue ainsi le modèle d'entreprise de proche en proche.

- Le référentiel est par la suite utilisé pour les études d'évolution. Il permet d'effectuer les analyses d'impact sur les modifications de données.

L'existence d'un tel modèle de données met l'entreprise dans une meilleure position pour décider des stratégies à adopter : quelles applications migrer, quelles données réutiliser pour de nouvelles applications, construire un Data Warehouse qui exploiterait les données existantes, ou consolider l'ensemble du parc applicatif d'une manière plus efficace et plus maintenable.

La migration vers les applications distribuées

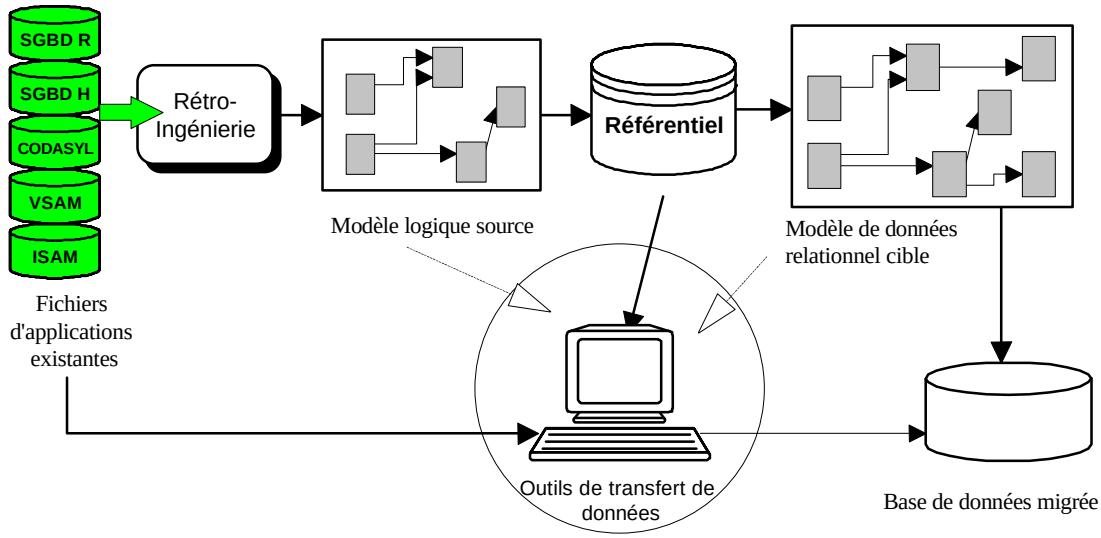
Une migration solide et durable des applications et de leurs bases de données vers une architecture distribuée exige de résoudre plusieurs difficultés :

- Les anciennes applications ayant été développées de manière indépendante sont très peu corrélées entre elles.
- Il faut identifier les règles de gestion les plus significatives, qui représentent au mieux la réalité, qui sont indépendantes des critères technologiques, et qui résisteront aux évolutions de l'entreprise.
- Les systèmes de gestion de bases de données relationnelles doivent être exploités au mieux. Ils doivent gérer les données de la manière la plus complète possible, ce qui n'était pas le cas avec les technologies précédentes.

- Il ne faut pas bloquer le futur du système d'information avec les modes de raisonnement rivés sur les vieilles technologies.

La solution proposée, en reconstituant l'ensemble des règles de gestion sur les données, permet d'obtenir des modèles de bases de données robustes qui exploiteront au mieux les SGBDR cibles. Elle permet de reconstituer les clés, les contraintes d'intégrité ou les déclencheurs, qu'il serait difficile de reconstituer autrement car ils sont noyés dans le code des programmes applicatifs.

Cette étape de migration de données est un préalable à la migration des applications.



Le référentiel est interfacé aux outils de transfert ou de nettoyage des données pour transmettre les modèles logiques sources, le modèle relationnel cible, ainsi que les correspondances entre données sources et données cibles.

Les programmeurs pourront alors utiliser pour développer leurs applications les outils modernes tels que les L4G ou ceux orientés objets. La migration des données sera facilitée par le maintien des correspondances entre l'ancienne structure de base de données et la nouvelle.

La solution permet donc aux programmeurs de se concentrer sur ce qui est important pour l'utilisateur comme les interfaces homme/machine, l'accès aux données ou l'architecture afin de leur fournir des applications plus efficaces.

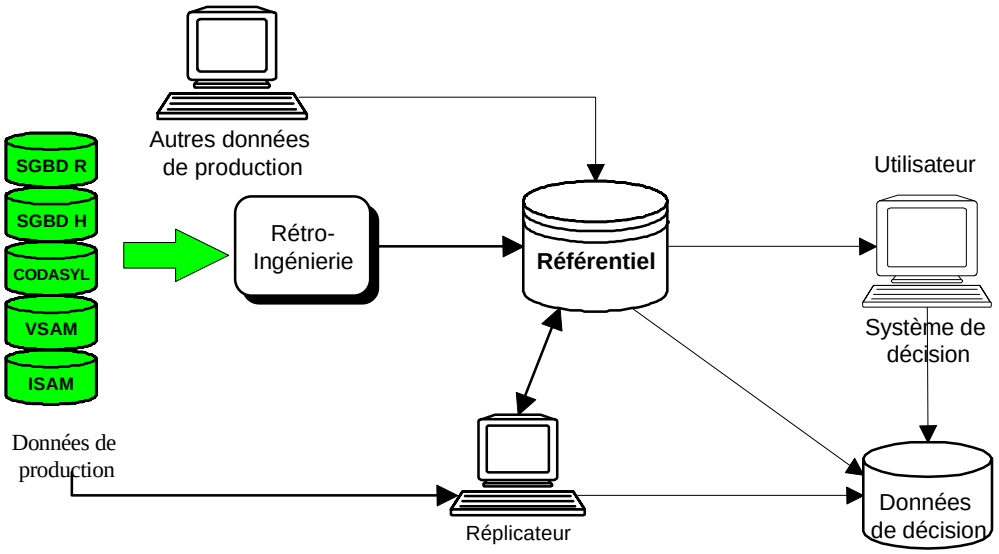
La constitution et la maintenance d'un Data Warehouse

Alors que la migration vers des structures distribuées se limite à une application, la constitution d'un Data Warehouse exige des vues horizontales à travers les données de l'entreprise.

Les données dont se nourrit le système de Data Warehouse sont éparses et nombreuses. Elles proviennent de différents lieux géographiques, de l'ensemble des services de l'entreprise, de l'ensemble des applications, elles ne sont pas forcément cohérentes entre elles. Ainsi le concept de *client* peut être différent selon les services et les applications répartis dans l'entreprise.

Le problème fondamental à résoudre est de rationaliser de multiples sources de données hétérogènes en un modèle unifié. C'est l'apport de la technologie de rétro-ingénierie.

Le référentiel joue le rôle de la pierre angulaire du Data Warehouse. Il aide à la conception, l'alimentation, la construction des requêtes et à la gestion des évolutions.



Un administrateur de données capture les méta-données et les modèles qui décrivent les bases de données de production. La technologie de rétro-ingénierie est utilisée pour reproduire des modèles de certaines bases de données de production lorsque les méta-données ne sont pas disponibles ou ne sont pas à jour.

- L'administrateur de données gère et administre les modèles à l'aide du référentiel. Il établit leur cohérence globale. Il constitue de proche en proche un modèle consolidé indépendant de la localisation des données.

- Le concepteur de Data Warehouse exploite ce modèle pour produire les modèles de données du Data Warehouse (de la base de données de décision). Il utilise éventuellement un OGL externe. Les modèles du Data Warehouse feront partie du référentiel et seront gérés par lui.

- Le responsable de l'alimentation du Data Warehouse réalise les modèles de transformation des données des bases de données de production vers les bases de données de décision. Il introduit les règles de transformation des données.

- Le référentiel communique avec les réplicateurs qui assureront l'approvisionnement du Data Warehouse. Il exporte les modèles de transformation. Il aide au monitoring des réplications.

- Le référentiel aide à l'exploitation du Data Warehouse en assistant l'utilisateur à la construction des requêtes d'interrogation. Celui-ci utilisera le glossaire, le thesaurus ou les outils d'interrogation.

- Le référentiel permet de maintenir le Data Warehouse en cas d'évolution des données. Il aide à effectuer les analyses d'impact en cas de modification amont des modèles.

Avec la solution proposée, les applications de Data Warehouse dynamiques sont plus rapidement disponibles, évolutives et maintenables, alimentées en données toujours cohérentes entre elles, plus rapides, plus simples, et font parties intégrantes du système d'information.

En conclusion

La solution proposée permet de construire des environnements complets de ré-ingénierie.

Elle apporte des gains en productivité et en qualité grâce la compréhension, la restructuration et la ré-ingénierie des bases de données existantes, et la conception de nouvelles bases de données relationnelles.

Elle permet aux analystes et aux concepteurs d'effectuer de manière précise et fiable :

- la construction d'un modèle d'entreprise ;
- la consolidation de bases de données hétérogènes ;
- la migration des applicatifs vers des environnements distribués ;
- les transformations sur les données qui permettent d'extraire des données de bases opérationnelles vers un Data Warehouse.

La solution proposée est ouverte : les utilisateurs peuvent intégrer les outils qui sont adaptés au contexte de leur système d'information.

De plus, elle résiste aux évolutions du système d'information dont la pérennité est assurée. Le patrimoine d'outils peut évoluer tout en préservant le capital informationnel de l'entreprise. ▲

Rémy Levy

☛ *Pour toute information complémentaire :*

☺ *Rémy Levy, Consultant Senior, Transtar*
☎ *01 46 94 77 00*
01 46 94 77 66
remy.levy@transtar.fr
✉ *Transtar - 14 rue de la Ferme - 92100 BOULOGNE*

Techniques et logique

Les nouvelles techniques interfaceront-elles notre logique ?

Nous nous sommes tous frottés aux ergonomies des nouvelles IHM (Interfaces Homme Machine).

Qui n'a jamais voué aux gémonies le distributeur de billets SNCF qui s'entête à exiger que vous choisissiez votre place (couloir ou fenêtre, sens de la marche ou sens inverse) alors que votre train n'attendra pas la fin de votre dialogue pour démarrer ?

Qui n'a jamais renoncé à faire le plein d'essence, face à une pompe qui accorde 20 secondes pour choisir un carburant, sans dire où sont les touches de sélection ?

Qui n'a jamais fulminé contre Word qui cache, dans les replis d'un incertain menu, la commande que l'on a pourtant exécutée la veille et que l'on est incapable de retrouver ?

La structuration hiérarchique

De Descartes à Warnier, nous avions appris à structurer nos informations de façon hiérarchique.

> Une feuille appartient à un rameau ; un rameau appartient à une branche ; une branche appartient à un tronc.
>
> Une page appartient à un document, un document à un dossier, un dossier à un classeur. Le classement des dossiers des secrétariats répondait à cette règle qui amenait quelquefois à dupliquer physiquement certains documents qui appartenaient à deux arborescences (par exemple, l'une commerciale, l'autre technique).

Cette structure hiérarchique, transcrite dans les systèmes informatiques, commande une recherche par arborescence de menus.

> Chaque menu du système de recherche propose une liste de possibilités. Il suffit de sélectionner celle qui ouvre une liste de possibilités plus fines, et ainsi de suite, jusqu'à la découverte de l'adresse du précieux objet de convoitise.
>
> Par rapport aux dossiers manuels, le système informatique nous soulage de la redondance des documents. En effet, l'adresse du même document électronique (qui n'est conservé qu'une fois sous une seule forme) peut être atteinte à partir de plusieurs arborescences.

En cas de mauvais aiguillage, il suffit de remonter, étage par étage, dans l'arborescence jusqu'à l'embranchement qui ouvre la bifurcation que l'on désire emprunter.

L'hypertexte

Depuis quelques années, de nouveaux outils informatiques nous affranchissent de la démarche hiérarchique. Il est désormais possible de se déplacer dans une base documentaire en naviguant selon des analogies.

L'utilisateur n'est plus obligé de connaître les chemins d'accès ; il procède en rebondissant sur des mots qui sont censés l'aiguiller vers l'objet de sa recherche. Il peut ainsi se promener, selon son inspiration, et butiner dans une forêt de connaissances.

Que choisir ?

La structure hiérarchique constitue un filtre à plusieurs niveaux. Les critères d'un niveau doivent être rigoureusement complémentaires, de façon à déterminer strictement la sélection à activer. La recherche est systématique et suit un algorithme infaillible.

A priori, la seconde approche, plus intuitive, apparaît plus séduisante ; elle n'impose pas de balayer tous les étages d'une hiérarchie. Mais, alors que la structure hiérarchique rend tous les chemins équiprobables, l'hypertexte canalise les trajets par des mots attrayants judicieusement choisis, disposés dans des emplacements stratégiques, qui favorisent certaines destinations.

Dans cette démarche, l'utilisateur emprunte des parcours qui ont été imaginés par celui qui a conçu la documentation. Son trajet est ainsi orienté par les balises placées par l'auteur « Point de vue à 200 m. », « Restaurant gastronomique » qui suggèrent au promeneur des détours prometteurs.

Dans tous les cas, l'efficacité de la recherche provient de la complicité entre l'auteur et l'utilisateur.

La première démarche est bien adaptée lorsqu'on cherche une réponse à une question précise et que l'on connaît exactement la signification des indicateurs de l'arborescence. C'est quelquefois long et fastidieux :

- lorsque l'on ne maîtrise pas encore la nomenclature des critères de sélection ;
- lorsque, déjà familiarisé, il faut passer en revue tous les niveaux (mode débutant) ou apprendre des raccourcis (mode expert).

La seconde démarche est plus satisfaisante :

- lorsque l'on cherche à découvrir un domaine encore inconnu, sous la conduite d'un guide discret mais efficace ;
- lorsqu'on connaît les petits cailloux proches de la destination souhaitée.

Pourrait-on aller plus loin ?

Terminons sur une boutade. L'idéal serait sans doute un système de recherche documentaire qui aura su apaiser l'angoisse exprimée par Francis Blanche dans un célèbre sketch.

Lequel demandait au Sâr Rabindranath Duval (alias Pierre Dac) : « Vous qui êtes si malin, pouvez-vous me dire, pouvez-vous me dire ? ..... la question que je dois vous poser maintenant ?».

Et le système lui aurait simplement répondu : « Oui, ... je peux vous le dire ! ».

Merci à Olivier Le Gendre, l'un des inspirateurs de cette réflexion par sa lettre du 14 mars 1997 intitulée « Événementiel contre hypertexte ». ▲

Alain Coulon

Le réseau sémantique universel (3ème partie)

L'intégration de quelques concepts permet d'étendre considérablement les possibilités de modélisation. Voici la suite - et la fin - de l'analyse de cette intégration.
L'auteur vous renouvelle ici ses excuses pour la densité et la non-exhaustivité de cette étude (et pour les égratignures faites à MERISE), mais il s'agit d'une suite d'articles à considérer comme un condensé – et un résumé – d'une étude plus complète qui sera publiée dans un proche avenir.

> Erratum : dans le schéma paru dans LA LETTRE n°27, page 31, la patte "possède" entre **OBJET** et l'association **Posséder** doit être fléchée vers **OBJET** (identification relative de **ATTRIBUT**) (Suite à un déplacement graphique, l'extrémité du tracé s'est retrouvée derrière **OBJET**, masquant ainsi la flèche).

Rappel des parties précédentes

Nous avons vu dans la première partie de cet article (La Lettre n° 26) qu'il était possible d'utiliser un seul et unique symbolisme pour modéliser pratiquement n'importe quel système, ceci ayant été présenté avec sept systèmes : Modèle Conceptuel de Données (MCD) de Merise ; Modèle Conceptuel de Traitements (MCT) de Merise ; Système à Base de Connaissance (SBC) ; Tableur (considéré comme un croisement entre une Base de Données et un SBC) ; Système NeuroMimétique (SNM) ; Modèle de Traitements en Temps Réel (TTR) ; Programmation Orientée Objets (POO).

Bien que le symbolisme utilisé soit - à la base - celui des MCD de Merise, il a été sensiblement élargi, prenant en compte les concepts suivants :

- **Objets** (modèles, populations, occurrences, singularités) ;
- **Attributs** (propriétés simples, faits, tableaux, agrégats, méthodes ; appartenant à des objets, des associations, ou des actions) ;
- **Associations** (modèles, populations, particularités, héritages) ;
- **Actions** (les traitements) ;
- **Prédicats** (attributs contribuant à un héritage par spécialisation ou au déclenchement d'actions) ;
- **Pattes** (reliant entre eux objets, associations, actions, prédicats).

Dans la deuxième partie de cet article (La Lettre n° 27), nous avons bâti un métamodèle - incomplet, mais cependant utilisable - qui permet de définir les entités décrivant un modèle, et ajoute diverses notions comme :

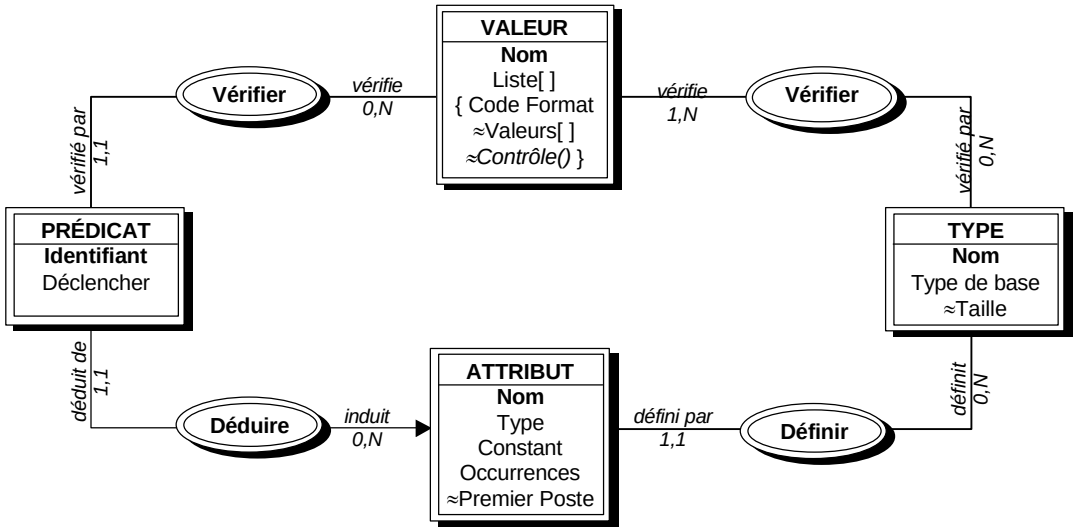
- **Types** (description des attributs-types) ;
- **Valeurs** (contrôle du contenu des attributs et des prédicats) ;
- **Règles** (Actions de type règle de gestion ou de Système à Base de Connaissances) ;
- **Méthodes** (Actions de type méthodes associées à des objets, fonctions, routines...) ;
- **Tâches** (Occurrences de Règles et/ou Méthodes activées pour exécution) ;
- etc.

Nous allons maintenant élaborer un modèle physique qui, implanté en tant que SGBD entité-relation, permettra de concevoir, prototyper et réaliser n'importe quelle application sans programmation.

Ce qui est décrit ci-après correspond à des concepts qui ont été, soit déjà testés dans divers prototypes, soit effectivement mis en œuvre sous des formes approchées dans diverses applications ; il ne s'agit donc pas d'un exercice de style de théorie pure.

Exercice

Avant de plonger dans cette élaboration, un petit rappel. Dans la partie précédente, quand nous avons modélisé les prédicats, nous avions obtenu proposé un exercice[1] concernant le sous-ensemble suivant :



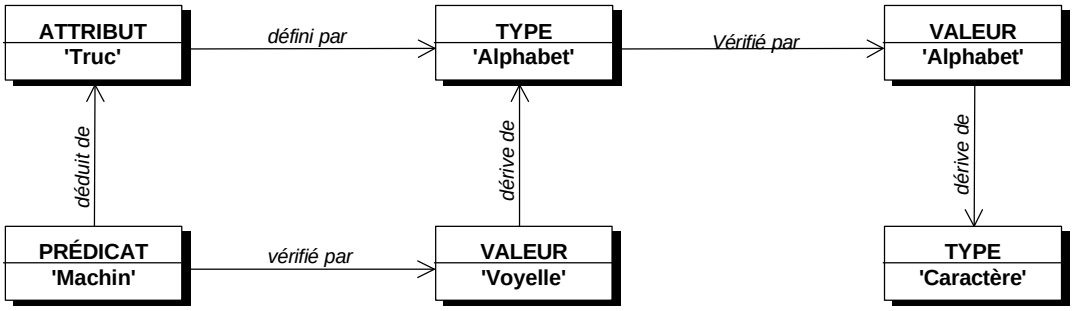
Nous avions dit «*L'association **Vérifier** avec **VALEUR**, qui permet d'associer une liste de valeurs à un prédicat, est sémantiquement légèrement différente de son homonyme avec **TYPE**, car elle définit en fait un sous-ensemble des valeurs associées au type de l'attribut. Les occurrences de **VALEUR** concernées ne sont donc pas identiques, mais il existe entre elles une contrainte, dans le détail de laquelle nous n'entrerons pas maintenant*». Eh bien, voici comment implémenter cette contrainte.

C'est en fait très simple. Il suffit que lors de la création de la listes de valeurs associées au prédicat, une action de validation s'assure que toute valeur fournie soit une valeur possible pour l'attribut.

Comment ? Nous sommes dans un métamodèle, donc nous allons associer à chaque individu VALEUR un TYPE qui définit le format et les valeurs autorisées pour l'attribut VALEURS.

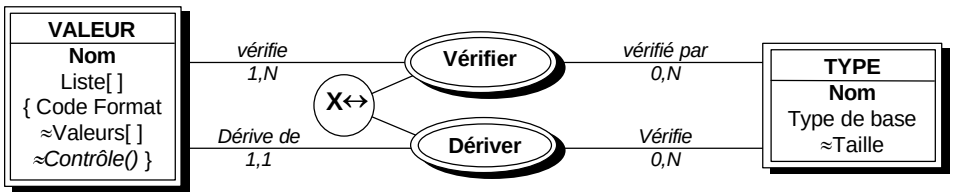
Par exemple, si un attribut TRUC est de type ALPHABET, à ce type sera associée une liste de valeurs limitant la saisie aux valeurs de 'A' à 'Z', elle-même associée au type CARACTÈRE (qui lui n'a pas besoin de liste de valeurs, puisque format primaire). Si l'attribut TRUC a un prédicat qui déclenche une action (ou définit un sous-type de l'objet contenant l'attribut) quand sa valeur correspond à une voyelle, ce prédicat sera relié à une liste de valeurs qui sera elle-même associée au type ALPHABET. Lors de la définition du prédicat, il sera donc impossible de saisir comme valeur un caractère qui ne soit pas alphabétique.

Pour plus de clarté, schématisons cet exemple (sous forme non standard) avec les individus correspondants :



[1] *Cf note 17 page 26 de LA LETTRE n°27 :* «Les amateurs pourront essayer de modéliser cette contrainte (objets "objet valeur" associés à des objets "attribut valeur", etc.). Réponse dans la 3ème partie de ce dossier.»

Il nous suffit donc d'enrichir le métamodèle avec une association **Dériver** entre **VALEUR** et **TYPE** , en interdisant les chemins en boucle entre **Dériver** et **Vérifier** :



Certains objecteront que le TYPE n'est ici pas associé à l'objet ATTRIBUT "Valeurs", donc au modèle, mais à l'objet VALEUR, ce qui semble anormal.

En fait, cette "bizarrerie" est associée à un format d'attribut spécial, utilisé dans un certain nombre de L4G et langages objet[2] : le format **VARIANT**. Il s'agit d'un format métamorphe : un attribut de ce type peut effectivement contenir n'importe quel type d'information : son format est celui de son contenu et est résolu non pas à la compilation, mais à l'exécution, et change en fonction de ce qu'on y met.

Bases de raisonnement

Pour élaborer le métamodèle, nous étions partis d'un premier niveau de généralisation (modèle) pour aboutir à un niveau de généralisation supérieur (métamodèle). En principe, nous devrions continuer cette généralisation jusqu'à aboutir à un ensemble suffisamment simplifié pour être réalisé physiquement (à force de paramétrer les paramètres, vient un moment où il faut bien programmer "en dur").

En fait, nous allons suivre maintenant un cheminement inverse : nous allons partir du composant physique le plus simple - ou presque - pour, allant de spécialisation en spécialisation, obtenir une représentation de notre métamodèle. Et, contrairement à ce qui avait été fait pour le métamodèle, c'est-à-dire l'élaborer au fur et à mesure du raisonnement, nous allons ici montrer des ensembles complets et les commenter, la démarche de conception étant de peu d'intérêt.

Notre point de départ sera donc constitué de l'ensemble qu'on retrouve dans tout système manipulant des données : un **Article** (ou enregistrement, segment, record, ligne...) contenant des **Champs** (ou éléments, données, fields, rubriques, colonnes...).

L'article

Qu'est-ce qu'un article ? C'est le plus petit regroupement d'informations accessible au niveau de la base de données physique : un article est toujours physiquement écrit et lu en entier.

Cependant, nous sommes ici dans un environnement complètement objet, et notre base de données est susceptible de contenir les ensembles les plus divers, certains n'existant qu'en un seul exemplaire. Il nous faut donc, pour pouvoir maîtriser ceci, faire au niveau de l'article ce que les SGBD relationnels font au niveau de la table : associer à chaque article un descripteur permettant le décodage de son contenu par une fonction générique.

Chaque article sera donc constitué ainsi :

| Longueur | Type | Numéro | Descripteur | Champs |

Longueur : la taille de l'article, puisqu'elle peut différer pour chacun.

Type et numéro : permettent d'identifier chaque article de façon unique (nous verrons les types plus loin) et de le retrouver directement dans la base sans index intermédiaire.

Descripteur : suite de codes prédéfinis permettant de connaître exactement le contenu de la partie champs, c'est-à-dire pour chaque champ : son type, sa taille (s'il y a lieu). Des codes

[2] *Dbase et Clipper (format implicite) ; Visual Basic (type "Object") ; Delphi (type "Variant") ; etc.*

permettent également d'indiquer l'emplacement de champs de longueur nulle, et de savoir pourquoi ils sont absents (valeur inconnue, non renseignée, etc.). De par sa construction, ce descripteur est auto-délimité. La fonction qui lit un article commence donc par décoder ce descripteur, ce qui lui permet de créer une table de description du contenu de la partie champs. En écriture, elle utilise cette table (remplie par une fonction de niveau supérieur) pour assembler le descripteur et constituer l'article. Les codes des formats de base sont "en dur", certains formats élaborés d'usage général peuvent se voir associer un code pour améliorer les performances du système.

Champs : le contenu proprement dit de l'article. Selon son type, un champ pourra correspondre à un attribut, un poste d'index, un pointeur vers un autre article, etc.

Réaliser un programme gérant un fichier constitué d'une telle suite d'articles est très simple. On disposera en fait ici d'une couche logicielle de base, purement technique, qui prendra en charge lecture et écriture des articles, gestion de la configuration de la base, gestion de l'espace disque et des adresses des articles, adressage des sous-ensembles à travers le réseau, partage entre utilisateurs, sécurité et recouvrement, réplication[3]...

Nous n'avons ici qu'un système de gestion de fichiers amélioré. Nous devons donc maintenant, d'une part différencier plusieurs types d'articles de base, qui auront chacun une pré-structure type, et être en mesure de naviguer simplement dans la base, avec un système d'adressage simple. Définissons maintenant ces différents types.

Types d'articles

Commençons par un bref inventaire :

- Tout accès à des données d'un SGBD entité-relation passe par un ou plusieurs objets. Le premier type d'article est donc l'**ENTITÉ** (appelé ainsi pour ne pas faire de confusion avec la classe OBJET, qui définit ceux-ci, mais un article ENTITÉ est bien l'équivalent d'un individu-objet).
- Pour accéder à un objet donné, il faut pouvoir le retrouver en fonction de son identifiant. Nous utiliserons pour ce faire des articles INDEX D'ENTITÉ (que nous appellerons **INDEX-E**).
- Dans certains cas, des objets auront plusieurs identifiants physiques. Nous disposerons donc d'articles **ALIAS-E** et **ALIAS-R** (parce que certaines relations, ayant également un identifiant physique, pourront également avoir des alias).
- Un objet a - généralement - des attributs autres que son identifiant. Ils seront stockés dans des articles **DONNÉES**.
- Certains attributs, de par leur taille et leur nature ne seront pas groupés, mais stockés dans des articles spécifiques : les articles **BLOB**[4].
- Les objets sont reliés entre eux avec des associations. Ceci implique de disposer d'articles **RELATION** (appelés ainsi pour ne pas faire de confusion avec les ASSOCIATIONS, qui sont physiquement constituées de RELATIONS et d'INDEX-R, mais chaque article RELATION est bien l'équivalent d'une occurrence d'association).
- À partir d'un objet, il faut pouvoir retrouver une association ou en obtenir la liste. On ira de l'un aux autres à l'aide d'articles INDEX DE RELATION (que nous appellerons **INDEX-R**).

D'autres types d'articles sont définis, notamment pour le paramétrage et la gestion technique du SGBD, mais ils ne sont pas importants dans la description condensée du présent document.

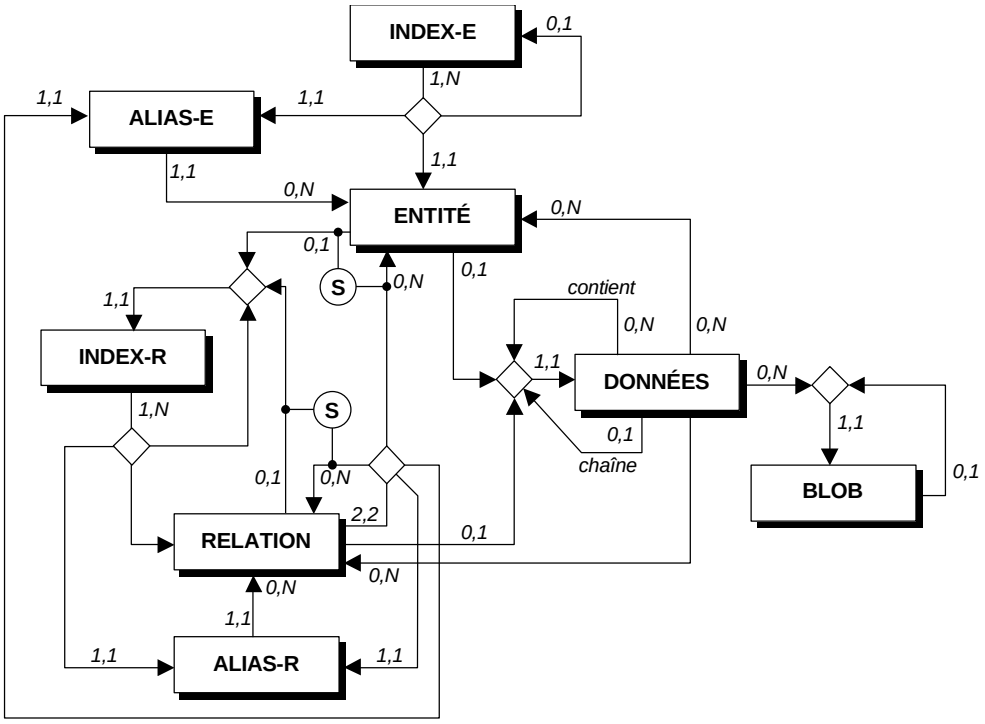
Avant de détailler chacun de ces types d'articles, un Modèle Logique de Données expliquera plus sûrement que deux pages de baratin comment on peut naviguer entre eux. Rappelons simplement que les losanges correspondent à des "pattes" alternatives : on peut ainsi voir qu'un article **DONNÉES** est

---

[3] *Il faut savoir que la simple description complète de cette couche et de la suivante fait l'objet d'un dossier EPHITEQ d'une centaine de pages. Ce qui est présenté ici est donc considérablement abrégé... et simplifié.*

[4] *BLOB = Binary Large OBject, pouvant être un texte, un son, une image, etc.*

pointé, soit par un article **ENTITÉ**, soit par un article **RELATION**, soit par un article **DONNÉES (contient)**, soit par un article **DONNÉES (chaîne)**, et par un et un seul d'entre eux.



Les flèches indiquent le sens de parcours, et chacun pourra s'assurer qu'à trois articles-type près (exceptions qui seront explicitées plus loin), toutes les liaisons ont une cardinalité de départ maximale de 1 ou 2, et correspondent physiquement à des pointeurs stockés dans les articles de départ.

Tout pointeur contient simplement l'identifiant de l'article vers lequel il pointe.

Vous avez bien sûr remarqué la présence de deux contraintes de simultanéité. Explication : une entité (resp. relation) qui pointe sur un index de relation a donc des associations. Dans ce cas il existe obligatoirement au moins une relation qui pointe vers cette entité (resp. relation) pour gérer le parcours inverse. CQFD.

Détaillons maintenant chaque type d'article avec ses liaisons.

L'article ENTITÉ

Tout article ENTITÉ a la structure suivante :

| Lg clé | Niveau | Classe | Langue | Champ(s) clé | Ptr→index-R | Ptr→données | Données |

Lg clé : la taille de l'identifiant, puisqu'elle n'est pas la même pour toutes les entités.

Niveau : un code d'état, permettant de différencier des niveaux et/ou versions différentes d'une même entité, sans duplication superflue, et de protéger des domaines. Par exemple, un niveau "*Système*" est attribué aux entités définissant le métamodèle de base, le protégeant de toute altération intempestive ; un autre niveau "*Test système*" sera vu comme l'union du contenu de "*Système*" et des changements effectués, invisibles quand on utilise directement "*Système*".

Classe : un code permettant d'identifier l'appartenance d'une entité, c'est-à-dire l'objet-type. On aura une classe définie pour chaque type d'objet au sens Merise, le nom de la classe étant associé au code.

Langue : un code permettant d'identifier à quelle langue est associé l'identifiant. Par exemple, "OBJET" sera associé au français, "OBJECT" à l'anglais, les deux étant synonymes (par alias). Une valeur spécifique existe évidemment pour spécifier "toutes langues" (l'individu "DUPONT Marcel" ne change pas d'identifiant si on travaille en anglais…).

Champ(s) clé : le ou les attributs constituant l'identifiant "normal" de l'entité.

Ptr→index-R : un pointeur (facultatif) vers l'index répertoriant les associations de l'entité.

Ptr→données : un pointeur (facultatif) vers les attributs de l'entité.

Données : si l'entité a peu d'attributs, ils peuvent être stockés directement dans l'article entité au lieu d'être dans un article séparé, ce qui améliore les performances. Dans ce cas, **Ptr→données** est nul.

Navigation : On peut retrouver un article entité, soit à partir d'un (et un seul) article index-e, soit à partir d'un ou plusieurs articles alias et/ou relations et/ou données (ce dernier lien n'est pas évident a priori, il sera explicité plus loin).

Note : Les champs Niveau et Langue - effectivement implémentés dans les prototypes déjà mis en œuvre - ont été décrits ici pour montrer les possibilités d'un tel système. Par manque de place et leur non-fondamentalité dans les principes présentés, nous ne décrirons pas leur mise en œuvre dans cet article.

L'article INDEX-E

Nous n'allons pas détailler ici ce type d'article. Disons simplement que ces articles constituent une arborescence organisée pour optimiser la recherche d'entités, classées dans l'ordre de leurs identifiants.

Ajoutons que si une entité, un alias ou un index-e est adressé par un et un seul index-e, il existe un (et un seul) article index-e qui n'est pas adressé ainsi, mais dont l'adresse fait partie de la configuration : il s'agit de l'"index-maître", point d'entrée de toute la base.

L'article DONNÉES

Un article DONNÉES est l'article le plus standard. Il suffit de savoir que quand on a affaire à un tableau ou un agrégat, son contenu fait l'objet d'un article distinct, et que l'article le contenant possède à l'emplacement de cet ensemble un pointeur (DONNÉE contient DONNÉE). Il en est de même si l'attribut est un BLOB.

Dernier cas particulier, un objet ou une relation peut être vu comme un attribut d'un autre objet et/ou relation. Pour ce faire, il existe dans un article DONNÉES un champ qui est un pointeur vers l'entité ou relation correspondante (nous verrons plus loin comment retrouver l'attribut à partir de l'objet ou de la relation). Bien sûr, on pourrait gérer ceci au moyen d'une association "normale" entre entités et/ou relations, mais ce raccourci correspond au sens quasi-exclusif de la navigation ; donc, autant simplifier celle-ci.

Un ensemble d'attributs pouvant contenir plusieurs agrégats, tableaux, etc., il peut donc exister plusieurs pointeurs à partir d'un même article, d'où les cardinalités 0,N des liens vers DONNÉES, BLOB et ENTITÉ.

Sachant qu'un objet, une relation ou un agrégat peut contenir un grand nombre d'attributs, le pointeur DONNÉE chaîne DONNÉE permet de répartir les attributs sur plusieurs articles.

Navigation : On peut retrouver un article données à partir d'un (et un seul) article entité ou relation ou données.

L'article BLOB

Ce type d'article est très simple, puisque ne contenant que tout ou partie d'un seul attribut. L'ensemble de cet attribut est contenu, soit dans un seul article s'il est taille suffisamment petite, soit réparti dans plusieurs articles chaînés.

Navigation : Un BLOB contenant un attribut, on accède à celui-ci exclusivement à partir d'un (et un seul) article DONNÉES (ou BLOB en cas de chaînage pour un gros attribut).

L'article RELATION

Tout article RELATION a la structure suivante :

| Lg clé | TypeG | TypeD | Champ(s) clé | Ptr→entitéG | Ptr→entitéD | Ptr→index-R | Ptr→données | Données |

Lg clé : la taille de la clé de la relation. Cette clé est constituée du type (TypeG ou TypeD, selon le chemin d'arrivée) et d'un certain nombre de champs permettant de différencier une occurrence spécifique parmi d'autres et/ou de les trier selon une séquence définie.

TypeG : un code permettant d'identifier le type de la relation par rapport à la première entité (ou relation) reliée, c'est-à-dire la patte gauche.

TypeD : un code permettant d'identifier le type de la relation par rapport à la seconde entité (ou relation) reliée, c'est-à-dire la patte droite.

Champ(s) clé : le ou les attributs constituant la clé de la relation. Dans le cas où on a affaire à une relation qui permet de retrouver un attribut pointant une entité, cette clé contient, soit le numéro de l'attribut de l'entité concernée, soit l'ensemble des numéros permettant de localiser cet attribut en cas d'agrégat(s) et/ou tableau(x).

Ptr→entitéG : un pointeur vers la première entité (ou relation) reliée.

Ptr→entitéD : un pointeur vers la seconde entité (ou relation) reliée.

Ptr→index-R : un pointeur (facultatif) vers l'index répertoriant les associations de cette association.

Ptr→données : un pointeur (facultatif) vers les attributs de l'association.

Données : si l'association a peu d'attributs, ils peuvent être stockés directement dans l'article relation au lieu d'être dans un article séparé, ce qui améliore les performances. Dans ce cas, **Ptr→données** est nul.

Navigation : On peut retrouver un article relation, soit à partir d'un (et un seul) article index-r, soit à partir d'un ou plusieurs articles relation dans le cas d'associations d'associations.

L'article INDEX-R

Même usage que les articles index-e, sauf que l'index-maître est obligatoirement adressé par une (et une seule) entité ou relation.

Signalons simplement qu'à chaque pointeur vers une relation est associé un indicateur permettant de savoir par quelle patte on arrive (gauche ou droite).

Les articles ALIAS-E et ALIAS-R

Il s'agit de variantes des articles ENTITÉ et RELATION, qui ont pour particularité de n'avoir ni données ni pointeurs, à l'exception d'un pointeur sur l'entité ou la relation dont ils sont l'alias :

Navigation : On peut retrouver un article alias, soit à partir d'un (et un seul) article index-e ou index-r, soit à partir d'un (et un seul) article relation (dont le code type identifie un lien vers un alias) appartenant à l'entité pointée par l'alias.

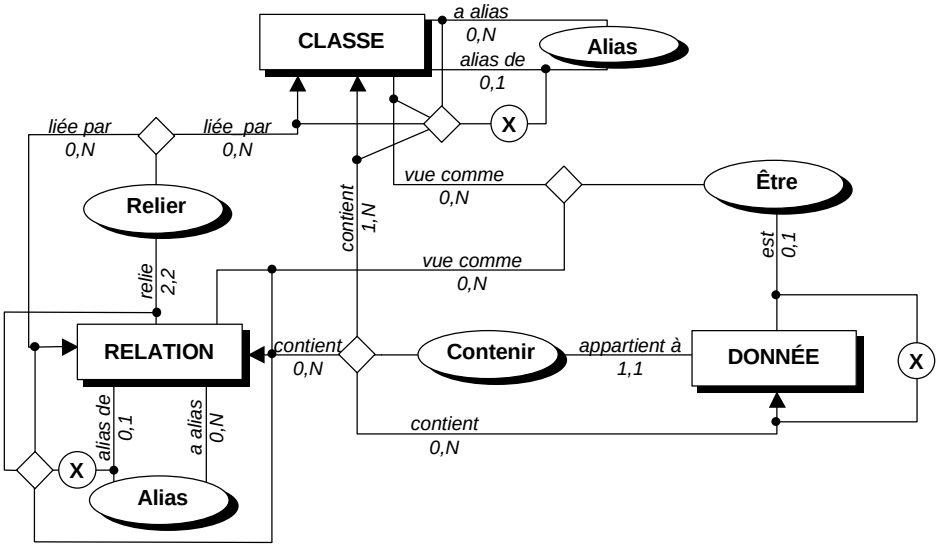
Synthèse

La gestion de ces différents articles est faite par un ensemble de fonctions qui permettent la consultation, navigation, ajout/insertion, modification, suppression de ces différents articles. Au-dessus de cette deuxième couche, on ne voit plus ni pointeurs ni articles en tant que tels, mais uniquement des objets, des attributs et des associations, cependant sous une forme encore très technique (pas de mnémoniques mais des codes binaires, données en format interne, etc.).

Nous disposons maintenant d'un substrat que nous allons commencer à alimenter avec des informations qui nous permettront ensuite d'y définir le métamodèle élaboré dans LA LETTRE n°27.

Du modèle physique au méta-2-modèle[5]

Nous allons maintenant définir dans le système décrit un modèle qui sera la conceptualisation de ce système, c'est-à-dire le MCD suivant :



La correspondance avec le modèle physique est simple : nous pouvons ignorer les index, qui sont ici "implicites", et considérer chaque article DONNÉES comme un regroupement physique d'objets DONNÉE, un BLOB étant lui-même une donnée ; un alias est une entité ou une relation n'ayant qu'une et une seule association ; la classe correspond à un champ de l'identifiant de chaque article entité.

Nous avons par ailleurs utilisé le mot CLASSE à la place d'ENTITÉ, car chaque individu CLASSE est une ENTITÉ-modèle. Vous allez comprendre tout de suite, car nous allons maintenant commencer à alimenter la base.

Toute la difficulté (et pas seulement à expliquer ou à comprendre) vient du fait qu'on dispose d'un contenant qu'on va remplir avec un contenu qui va se décrire lui-même[6], en plus de faire les correspondances entre le niveau physique et le méta-2-modèle. Nous n'allons donc pas entrer ici dans le détail (la simple liste des articles à créer pour définir ce métamodèle, en prenant en compte les notions de niveau et de langue, occuperait plusieurs dizaines de pages, à raison d'une ou deux lignes par article…), mais seulement montrer quelques exemples simples afin que vous vous rendiez compte que "ça peut marcher…"[7].

On utilisera une "classe" spéciale (le n°0) pour distinguer les entités et relations contenant les codes binaires des entités et relations contenant les mnémoniques.

Nous avons ci-dessus 3 entités et 9 relations. Nous allons donc créer les entités correspondantes (le caractère "•" sépare des données différentes) :

- Le premier groupe est | 1 | CLASSE | , | 1 | RELATION | et | 1 | DONNÉE |, avec respectivement comme alias : | 0 | 1•1 |, | 0 | 1•2 | et | 0 | 1•3 | (le premier chiffre est la zone **Classe** des articles Entité). Ceci signifie simplement que la classe n°1 s'appelle **Classe**, la classe n°2 s'appelle **Relation** et la classe n°3 s'appelle **Donnée**.
  (Pour éclaircir, l'individu **Classe Relation** décrit le modèle de la *classe* **Relation**, dont chaque individu décrit le modèle d'une *relation*…)

---

[5] *Ou méta-métamodèle, c'est-à-dire modèle du métamodèle.*

[6] *C'est pourquoi, si le MCD ci-dessus représente des individus (cadre simple), on aurait pu tout aussi bien y représenter des modèles (cadre double), les deux coexistant (le modèle d'un objet est un individu d'une classe de modèles).*

[7] *Et je puis vous certifier que ça marche vraiment !*

- Le second groupe contient, entre autres, | 2 | Contenir |, qui a avec | 1 | CLASSE | la relation | 3 | 3 | contient•gauche | (ayant elle-même l'alias | 1 | 2 | 0•1•4•3 |), et avec | 3 | DONNÉE | la relation | 3 | 3 | appartient_à•droite| (qui a l'alias | 1 | 1 | 0•3•1•1 |). Ce qui signifie que l'*entité* **Relation "Classe Contenir Donnée"** décrit la relation du même nom ; que la patte qui part de **Classe** s'appelle **contient** et a le type 4 (alias "0.1.4.3" = de classe 1, 4ème relation, vers classe 3), celle qui part de **Donnée** s'appelle **appartient à** et a le type 1. On a utilisé pour cela deux relations **"Classe Relier Relation"** qui sont de type 3 depuis Classe (**liée par**) et de type 3 depuis Relation (**relie**).
  Pour mieux comprendre, avec des commandes mnémoniques on aurait (par exemple) :

```
Créer_entité Relation Nom="Contenir" ;
Créer_relation Classe "Classe" liée_à Classe "Relation" Nom="contient" Gauche=oui
    alias=(Nom=x'00' & x'01' & x'04' & x'03') ;
Créer_relation Classe "Donnée" liée_à Classe "Relation" Nom="appartient_à" Gauche=non
    alias=(Nom=x'00' & x'03' & x'01' & x'01') ;
```

Le même processus permettra de définir les données, leur format (qui est lui-même une donnée des entités de la classe Donnée…), leur appartenance, leur numéro relatif par rapport à cette appartenance... Dans un système opérationnel, d'autres classes de base sont bien sûr indispensables (langue, niveau, mots-clés, paramètres système…).

Bref, les fonctions qui gèrent ceci sont désormais capables de convertir le nom de classe **Relation** en code classe **2** (et vice-versa), savent que si on cherche les données appartenant à une entité ABC (on cherche la *relation* **Classe contient Donnée**) il faut partir de l'*entité* **Classe ABC** et lister les relations ayant le code gauche **2**, etc.[8]

À partir de cet ensemble, on utilisera désormais un ensemble de fonctions qui s'occuperont elles-mêmes d'allouer des numéros aux diverses entités, relations et données. Quand on demandera "Créer Classe Prédicat", que le système lui alloue le n°25 ou bien le n°36453 nous est complètement indifférent…

Implémentation du métamodèle

L'intérêt de ce système de métamodèles réflexifs, c'est que :

a) On définit une information de la même manière qu'on l'alimente (à la différence du système relationnel dans lequel création et modification des tables n'utilisent pas les mêmes commandes que leur mise à jour) ;
b) Toute information définie est immédiatement utilisable, comme vous pourrez le constater dans les exemples ci-après.

Avec le méta-2-modèle ci-dessous, qui nous permet désormais de travailler avec des mnémoniques, nous allons enrichir la base pour définir le métamodèle élaboré dans LA LETTRE n°27.

Pour vous simplifier la compréhension, des mots-clés seront utilisés avant d'être définis (nous les mettrons en italiques dans les commandes). D'autre part, l'implémentation de base des classes, données et relations étant "en dur", il est possible d'utiliser un langage de commande[9] plus puissant : on pourra ainsi dire :

```
Créer_entité Relation "Objet Identifier Attribut" Gauche=(Nom="identifié par" Cardinalités=(1,1))
    Droite=(Nom="identifie" Cardinalités=(0,)) ;
```

Ce qui créera l'entité | 2 | Identifier | et ses relations | 5 | 7 | identifié par•gauche | 1•1 | (avec **Classe "Objet"**) et | 5 | 7 | identifie•droite | 0 | (avec **Classe "Attribut"**).

---

[8] *Si on cherche par "Donnée appartient_à Classe", la recherche se ferait en partant de l'entité* Donnée XYZ *sur les relations ayant le code droit à 1. Vous suivez ?*

[9] *Ceux qui verraient une troublante ressemblance entre le langage utilisé ici et celui du DB/DC Data Dictionary d'IBM n'auraient pas tort : je m'en suis fortement inspiré - tout en l'enrichissant fortement.*

Pour commencer, les types **Objet**, **Attribut** et **Association** du métamodèle correspondent évidemment aux classes **Classe**, **Donnée** et **Relation**. Définissons-les :

```
Créer_alias Classe "Objet" alias_de Classe "Classe" ;
Créer_alias Classe "Attribut" alias_de Classe "Donnée" ;
Créer_alias Classe "Association" alias_de Classe "Relation" ;
```

Le type **Objet** a des attributs :

- *Unique* : unicité des identifiants (booléen)[10] ;
- *Alias* : peut avoir des alias (booléen) ;
- *Type* : type d'objet (singularité, occurrence, population) ;
- etc.

Nous allons les créer et les relier (avec des commandes élaborées[11]) :

```
Créer_entité Attribut "Objet"."Unique" ;
Créer_entité Attribut "Objet"."Alias" ;
Créer_entité Attribut "Objet"."Type" ;
```

… puis les renseigner :

```
Modifier_entité Classe "Objet" Unique=oui Alias=oui Type=population ;
Modifier_entité Classe "Attribut" Unique=non Alias=oui Type=population ;
Modifier_entité Classe "Association" Unique=oui Alias=oui Type=population ;
```

Un Objet a aussi des associations, dont certaines ont des attributs. Par exemple :

```
Créer_entité  Association  "Objet Hériter Objet"  Gauche=(Nom="hérite de"  Cardinalités=(0,N))
     Droite=(Nom="parent de" Cardinalités=(0,N)) ;
Créer_entité Attribut "Objet Hériter Objet"."Type héritage" ;
```

Pour pouvoir définir un attribut, l'objet correspondant a lui-même des attributs. Définissons les principaux :

```
Créer_entité Attribut "Attribut"."Type" ;
Créer_entité Attribut "Attribut"."Constant" ;
Créer_entité Attribut "Attribut"."Occurrences" ;
Créer_entité Attribut "Attribut"."Premier poste" ;
```

À partir d'ici, nous supposerons déjà définis - et renseignés - les types d'objets **TYPE** et **VALEURS** (sinon nous y sommes encore dans quinze pages…). Renseignons donc les attributs que nous avons déjà définis pour les entités **Attribut** existantes :

```
Modifier_entité Attribut "Objet"."Unique" Type=champ Constant=non ;
Créer_association Attribut "Objet"."Unique" défini_par Type "Booléen" ;
Modifier_entité Attribut "Objet"."Alias" Type=champ Constant=non ;
Créer_association Attribut "Objet"."Alias" défini_par Type "Booléen" ;
Modifier_entité Attribut "Objet"."Type" Type=champ Constant=non ;
Créer_association Attribut "Objet"."Type" défini_par Type "Type objet" ;
Modifier_entité Attribut "Objet hériter Objet"."Type héritage" Type=champ Constant=non ;
Créer_association Attribut "Objet hériter Objet"."Type héritage" défini_par Type "Type héritage" ;
Modifier_entité Attribut "Attribut"."Type" Type=champ Constant=non ;
Créer_association Attribut "Attribut"."Type" défini_par Type "Type attribut" ;
Modifier_entité Attribut "Attribut"."Constant" Type=champ Constant=non ;
Créer_association Attribut "Attribut"."Constant" défini_par Type "Booléen" ;
Modifier_entité Attribut "Attribut"."Occurrences" Type=tableau Constant=non Occurrences=2 Premier_poste=1 ;
Créer_association Attribut "Attribut"."Occurrences" défini_par Type "Entier" ;
Modifier_entité Attribut "Attribut"."Premier poste" Type=champ Constant=non ;
Créer_association Attribut "Attribut"."Premier poste" défini_par Type "Entier relatif" ;
```

[10] *Indispensable, notamment pour les identifiants relatifs.*

[11] *Qui créent l'entité* Attribut "xxxx*" et le relient à* Classe "yyyy*" avec la relation* "Classe contient Attribut*".*

On définit également d'autres types d'objets. Par exemple :

```
Créer_entité Classe "Action" Unique=oui Alias=oui Type=occurrence ;
Créer_entité Attribut "Action"."Nom" Type=champ Constant=oui ;
Créer_association Attribut "Action"."Nom" défini_par Type "Chaîne" ;
Créer_entité Attribut "Action"."Code" Type=champ Constant=non ;
Créer_association Attribut "Action"."Code" défini_par Type "Code source" ;
…
Créer_entité Action "Créer tâche" Code="…code source…" …
…
```

…et ainsi de suite.

Bref, le métamodèle élaboré dans LA LETTRE n°27 contenant 14 types d'objets, 25 types d'associations, plus de 30 types d'attributs, des prédicats… - et il est très incomplet -, on procède successivement à la définition d'objets, associations, attributs, de leurs associations internes, puis on enrichit les entités et associations - nouvelles et existantes - avec les nouveaux attributs et associations…

En résumé : on définit, on utilise pour définir et enrichir, etc.

Il reste là-dessus à programmer "en dur" les méthodes et règles de base qui permettront d'exploiter les données - sans oublier l'action qui se chargera d'exécuter les actions définies par les entités du même nom.

Quant aux objets et/ou attributs qui sont des capteurs ou des effecteurs (c'est-à-dire qui correspondent aux entrées-sorties), ils sont associés à des actions de type "pilote"[12] vers/depuis les périphériques connectés (par exemple, pour les accès en saisie et/ou affichage sur votre écran, les pilotes sont tout simplement les fonctions idoines de l'A.P.I.[13] de Windows - ou d'Unix).

Commentaires

Résumons-nous. Dans LA LETTRE n°26, nous avons montré qu'on pouvait modéliser n'importe quel type d'application avec un symbolisme unique. Dans LA LETTRE n°27, nous avons élaboré un métamodèle permettant de définir n'importe quel modèle d'application. Et nous avons maintenant élaboré un SGBD qui est capable d'exploiter directement ces modèles pour générer et utiliser des applications.

Obtenir une application opérationnelle directement en élaborant son modèle conceptuel, ça c'est du R.A.D[14] !

D'accord, d'accord, vous avez plein de questions et de critiques : comment définit-on un identifiant relatif, une contrainte d'intégrité, un prédicat ? Où sont la sécurité, le réseau ? Comment être portable ?[15] Je l'ai dit : les presque cinquante pages - réparties sur 3 LETTRES - que vous avez pu lire ne sont qu'un résumé très succinct, et beaucoup de choses y ont été omises. Et inutile de me contacter pour faire des remarques sur des points de détail, je vous ai présenté un principe dans ses grandes lignes.

En-dehors de la rapidité de mise en œuvre d'une application (y compris le temps réel, je le rappelle), voyons quelques avantages d'un système entité-relation.

---

[12] *"Driver" pour les anglophiles.*

[13] *Application Programming Interface, ou routines d'interface entre l'application et le système d'exploitation.*

[14] *Rapid Application Development.*

[15] *Grâce à un langage idoine pour le code source des actions, ça a été dit dans LA LETTRE n°27.*

La non-redondance

Ce qui permet à un SGBD entité-relation comme celui-ci d'être performant malgré sa complexité interne (sans rapport avec la simplicité - pour ne pas dire l'indigence - du modèle relationnel), c'est qu'on ne duplique **jamais** un identifiant : si Mlle DUPONT devient Mme DURAND, on change son nom dans son entité, et tout ce qui fait référence à cette personne la connaît immédiatement sous son nouveau nom, puisque tout se fait par des pointeurs. On a donc une excellente garantie d'intégrité du système.

La réflexivité du langage vs l'orientation objet

Si le langage utilisé dans cet article pour définir et renseigner la base de données n'est qu'un exemple, vous aurez pu constater qu'il est à la fois très simple et très riche, puisqu'il s'enrichit lui-même au fur et à mesure des définitions, permettant à des choses au départ explicites de devenir ensuite implicites.

Prenons un exemple : si on définit un type d'attribut Description et un type d'objet Mot Clé, on peut y associer une action qui scannerait toute description créée ou mise à jour pour relier l'entité contenant cette description aux entités Mot Clé correspondant aux termes trouvés. Par la suite, il suffit, lors de la définition d'une classe, d'y associer un attribut de type Description pour disposer automatiquement d'un thésaurus.

Autre exemple : on a définit un agrégat Adresse (contenant donc nom du site, n°, rue, code postal, localité…) et on a défini une action qui s'assure de la cohérence entre code postal et localité. Quelque soit par la suite le type d'objet utilisé, s'il a quelque part une Adresse, la saisie et la mise à jour de celle-ci sera contrôlée.

Bref, on dispose d'une réutilisabilité complète.

De plus - ce qui ne gâte rien - on est débarrassés une fois pour toute des programmes à recompiler dès qu'on altère un tant soit peu la structure d'une table relationnelle : toutes les actions utilisent automatiquement les attributs dans leur format réel (il suffit de gérer un contrôle d'intégrité pour éviter les incompatibilités lors de l'altération du format).

La gestion des requêtes

L'une des grandes forces des SGBD relationnels actuels est le langage SQL, qui permet d'effectuer des requêtes parfois très complexes. Or, que fait une requête SQL, sinon une navigation pour identifier les différents objets qui répondent aux critères de recherche ?

Cette force est en même temps une faiblesse, car certaines requêtes complexes sont consommatrices de temps machine, et sont à refaire entièrement à chaque fois.

Le système entité-relation que nous avons décrit permet de conserver cet avantage en en évitant les inconvénients. Il suffit pour cela de disposer d'une classe Requête, dont chaque occurrence sera reliée aux objets répondant aux critères de la recherche correspondante. Une requête contiendra, tant par attributs que par associations, prédicats et actions, le descriptif du traitement.

Quand une requête sera créée, elle sera exécutée. Par la suite, toute nouvel accès à cette requête donnera immédiatement le résultat de la recherche.

Mais s'il y a eu des changement dans la base ? demanderez-vous. Pas de problème, puisque les actions permettant de relier les objets à la requête seront activées lors d'une mise à jour d'un objet/association/attribut concerné et maintiendront automatiquement les liens à jour.

Une autre question vous brûle les lèvres : n'y a-t-il pas risque d'inflation à force de créer des requêtes de toutes sortes ? Effectivement. C'est pourquoi il faut également prévoir une action chargée de supprimer les requêtes qui n'auraient pas été utilisées depuis un certain temps.

Objectif à court et moyen terme

Résumé historique

Le système présenté au cours de ces trois articles a considérablement évolué au cours du temps :

- La version 1, élaborée entre 1977 et 1984, était conçue pour gros système IBM (Système d'exploitation MVS, langages assembleur IBM/370 et PL/1), et n'a existé - hors papier - que sous forme de sous-ensembles séparés, destinés à vérifier la faisabilité et les performances.
- La version 2, élaborée entre 1985 et 1991, était conçue pour PC (MS/DOS, langages turbo-assembleur 1 et turbo-pascal 5.5). Le prototype réalisé était capable de gérer des entités avec leurs index, ainsi qu'une partie des fonctionnalités données et relations.
- La version 3 (1992-1994), conçue multi-plates-formes (MS/DOS avec XMS, Windows, OS/2) a été - toujours partiellement - réalisée en turbo-assembleur 3 et Borland C++ 3.1.
- La version 4 (1995), conçue pour Windows 95, a intégré le multitâche et l'utilisation en réseau, mais n'a pas dépassé le stade du papier.
- La version 5, dont vous venez de lire un bref résumé du concept, sera conçue pour Windows 95 et NT, et le prototype réalisé en Delphi 3.

Le projet

Une commission «***Logiciel d'évaluation des AGL***» a été créée au sein de l'ADELI fin 1996. Le prototype de cette application sera réalisé avec une implantation physique proche de la description ci-dessus - mais très simplifiée, certaines fonctions étant (temporairement) codées "en dur".

Le SGBD sera implémenté, non pas de toutes pièces, mais sur un substrat relationnel (20 champs, 4 tables et 5 index suffisent). Évidemment, ceci se fera au détriment des performances (accès aux articles par clé interne et index au lieu de pointeurs directs), mais accélérera considérablement la réalisation.

Le MCD du logiciel d'évaluation étant lui-même très simple, il pourra très facilement être implémenté sur ce support. À l'heure où vous lisez ces lignes, les travaux ont déjà commencé.

Par la suite, les versions suivantes de l'applicatif (et d'autres produits à faire) profiteront d'un niveau d'interfaçage toujours plus important, jusqu'à parvenir à une implémentation complètement paramétrée.

Perspectives pour demain…

Le SGBD que nous avons conçu tout au long de ces trois articles est basé sur les sept systèmes cités au début, et permet donc de les implémenter tous[16]. Mais bien d'autres possibilités sont envisageables.

Traitement de texte

Tout document issu d'un traitement de texte est structuré de manière arborescente : sections et/ou chapitres, cadres et tableaux, paragraphes, texte, chacun ayant ses propres attributs :

- pour les chapitres et sections : format de page, en-tête et pied de page…
- pour les tableaux : colonnage, en-tête (et parfois en-pied), bordures…
- pour les cadres : dimensions, positionnement, chaînage…
- pour les paragraphes : style, espacement, alignement, puce ou numérotation…
- pour le texte : police, taille, graisse, décalage, couleur…

…avec la possibilité d'insérer d'autres objets : images, etc.

Il serait donc très simple de définir un type d'objet DOCUMENT, avec une arborescence d'attributs (agrégats et tableaux) qui contiendraient les différentes caractéristiques de chaque type de composante,

[16] *Y compris toutes les formes de SGBD "classiques" (hiérarchique, relationnel, codasyl...). Si, si, essayez...*

associées au contenu (les attributs pointant directement sur d'autres objets permettant l'incorporation d'autres objets par liaison[17]). Un document pourrait donc contenir d'autres documents de tous types.

Hypertexte

Un attribut qui pointe sur un autre objet, ou une information (texte, image…) à laquelle on associe un lien (signet, URL ou autre), et voilà, on surfe dans le SGBD. Fini, les formats spécifiques de fichiers d'aide, HTML et autres…

Graphisme

Tout comme un document issu d'un texteur, une image vectorielle (qu'elle soit 2D ou 3D) se compose de plans, dans lesquels se superposent des objets et groupes d'objets, constitués de lignes, courbes, surfaces, texte…

Une image bitmap constitue également un fichier structuré, surtout quand elle est compactée.

Alors, qui me proposera une structure d'objet "graphisme" renvoyant aux oubliettes les formats BMP, GIF, JPG, PNG, CDR, DXF, CGM, WMF, etc, etc, etc. ?

Multimédia

Qu'est-ce qu'un document multimédia (diapositives, animation…) sinon un ensemble d'objets associés dans un sur-ensemble structuré ? Alors, en avant pour remplacer les formats AVI, MOV, GIF animé, MPEG, PPT et consorts…

Mémoire de masse

Qu'il s'agisse d'Unix, Ms-Dos, Windows NT ou beaucoup d'autres, les données manipulées par un système d'exploitation sont stockées sous forme de fichiers, regroupés dans des répertoires… arborescents. Ici aussi, notre SGBD pourrait, plutôt qu'être implémenté comme un fichier ou un ensemble de fichiers, directement remplacer systèmes FAT, NTFS…[18]

…et après

Finalement, au vu de ce qui précède, au lieu d'énumérer ce qu'un SGBD comme celui que nous avons décrit peut faire, il serait peut-être plus simple de chercher ce qu'il ne peut pas faire (le café ? Pourquoi pas, avec les interfaces domotiques et robotiques idoines connectés à des capteurs et effecteurs…).

À long terme, il est certain que se standardisera au niveau mondial un système intégré dans lequel chaque utlisateur aura son espace - sans distinction pour lui entre mémoire vive et mémoire de masse - qu'il pourra consulter et utiliser comme une banque de données et/ou un "Datawarehouse", dans lequel il pourra naviguer (dans la limite des autorisations des différents domaines), piocher, etc.

Peut-être en ai-je décrit les premières bases, au cours de toutes ces pages ?

Et si un "grand" du logiciel décide de créer ou de faire évoluer son produit en s'inspirant des idées publiées ici, S.V.P., contactez-moi plutôt que de vous servir directement : ce serait plus sympa… et je n'ai pas tout écrit ici, loin s'en faut ! ▲

Jean-Luc Blary

---

[17] Comme tout objet est géré par les actions qui lui sont associées, indépendamment de toute liaison externe, le petit parfum d'OLE que nous avons là rend ce dernier primitif et obsolète (sorry, Bill).

[18] Par exemple, les fichiers "basiques" peuvent être manipulés comme un seul objet (1 enregistrement = 1 attribut), avec l'avantage de pouvoir accepter des insertions/modifications/suppressions d'enregistrements intermédiaires sans réécriture de tout le fichier...

Sources documentaires

MERISE, support de cours
ABOUHAIR G.
IBSI, Paris 1988, 1991, 1992

EPHIBASE, SGBD entité-relation orienté objet, dossiers de conception et prototype
BLARY J-L.
EPHITEQ, Caëstre 1989-1990, 1993, 1995, 1996

ODESYS, Outil d'aide à l'évolution du Système d'Information, dossiers de conception
BLARY J-L., THELLIEZ Ph.
EPHITEQ, Caëstre 1993-1996

MERISE & OBJETS, support de cours
BLARY J-L.
EPHITEQ, Caëstre 1995

MERISE, support de cours
BLARY J-L.
EPHITEQ, Caëstre 1996

Le Réseau Sémantique Universel (1ère partie)
BLARY J-L.
La Lettre de l'ADELI n°26, janvier 1997

Le Réseau Sémantique Universel (2ème partie)
BLARY J-L.
La Lettre de l'ADELI n°27, avril 1997

De la modélisation systémique au réseau sémantique
BRES P-A., ROCHFELD A., TABOURIER Y., SIBERTIN-BLANC C.
Conférence-débat de l'AFCET, Paris 15/11/1990

SGBD avancés : bases de données objet, déductives, réparties
GARDARIN G. VALDURIEZ P.
Eyrolles, Paris 1990

Merise vers une modélisation orientée objet
MOREJON J.
Les Éditions d'Organisation, Paris 1994

Réseaux de neurones
NADAL J-P.
Armand Colin, Paris 1993

Le langage C++
STROUSTRUP B.
Addison-Wesley, Paris 1992

Moteurs de systèmes experts
VOYER R.
Eyrolles, Paris 1987

☛*Intéressés, critiques, puristes, adversaires, partisans... pour en savoir plus ou participer :*

☎ *0.660.602.702*
0.328.402.702
jlblary@nordnet.fr
✉ *EPHITEQ - Château Vallée - 59190 CAËSTRE*

La mémoire de l'eau...

Dans les colonnes de la presse de vulgarisation scientifique, une vive polémique oppose, depuis quelques années, Jacques Benveniste et ses détracteurs sur la thèse de « la mémoire de l'eau ».
Jacques Benveniste avance une hypothèse révolutionnaire qui pourrait expliquer l'homéopathie.
La structure moléculaire de l'eau se transformerait au contact de ses solutés (corps dissous) et cette modification survivrait à une dilution infinie.

Il s'appuie sur une expérimentation (dont la rigueur est contestée par ses contradicteurs). On crée une solution aqueuse par dissolution d'un corps dans l'eau pure. Une agitation énergique rend la solution parfaitement homogène. Puis on fait un prélèvement d'une partie de cette solution que l'on dilue avec une grande quantité d'eau pure. On refait cette opération un grand nombre de fois. Au-delà d'un certain niveau de dilution, il ne peut subsister matériellement aucune molécule du corps dissous.
La solution est alors une eau chimiquement pure mais qui ne se comporte plus comme de l'eau pure ! En effet, certaines propriétés semblent provenir du souvenir de cette union provisoire avec un autre corps, dont on a fait disparaître toute trace.

Je ne me prononcerai pas sur cette hypothèse scientifique. En revanche, j'affirme, haut et fort, que le texte a une mémoire et je vais vous le prouver. Voulez-vous vous prêter à l'expérience suivante ?

Créez la mixture initiale

Écrivez, d'un premier jet, selon votre inspiration, sans vous censurer, un texte direct, en n'évitant ni les formules à l'emporte-pièce, ni les termes crus.
Puis relisez votre prose. Il n'est pas possible de soumettre un tel texte à la lecture d'un destinataire non averti. Celui-ci risquerait d'être choqué par la rugosité de votre franchise.

Diluez-la progressivement

Polissez votre texte. Remplacez chaque mot trop net par un synonyme. Utilisez le langage de la diplomatie. Amusez-vous à jongler avec la polysémie des mots-caméléons - ces mots qui peuvent présenter plusieurs significations distinctes en fonction de leur contexte et des intentions de l'auteur. Changez de temps ou de mode, conjuguez au futur ou au conditionnel.
Après chaque passage correctif, procédez à une nouvelle lecture. Si le texte est encore susceptible d'égratigner, recommencez consciencieusement cette opération en éliminant, peu à peu, toutes les aspérités.

Servez-la tranquillement

Puis, quand aucun mot ne dépasse plus des conventions d'un discours politiquement correct, diffusez ce texte de bon aloi.
Vous serez le premier surpris par son impact. Le destinataire retrouvera immédiatement le sens initial de votre texte. Ainsi, vos sentiments auront été véhiculés par les phrases du texte, alors que les mots actuels ne gardent aucune trace physique des mots initiaux.

Alors, pour expliquer ce phénomène, il faut admettre que le texte a une mémoire. Le texte s'imprégnerait des mots mis à son contact et conserverait toujours la trace de la première rédaction qu'il a accueillie. Il suffirait de placer ce texte, véritable cryptogramme de vos sentiments, dans un environnement qui connaît votre comportement, pour reconstituer la totalité du message que vous vouliez transmettre.

Un lecteur troublé serait bien en peine de formuler un reproche. Il sera d'autant plus perturbé qu'il lui sera matériellement impossible d'appuyer ses soupçons sur des faits précis. Aucune preuve (ni sémantique ni syntaxique) ne lui permet d'accuser l'auteur d'un acte volontaire.

Le seul recours du destinataire serait de répondre, en maîtrisant la même technique ! ▲

Alain Coulon

Post-scriptum : Ce texte a été écrit en utilisant la technique qu'il décrit. Je remercie la fonction « synonymes » de mon fidèle traitement de texte qui propose à la fois les synonymes d'un mot et son éventuelle polysémie.

Protégez-vous !

Vous avez l'intention de vendre, à prix très élevé pour en accroître la crédibilité, un rapport d'étude destiné à guider les pensées de nos élites.

Commencez par le traduire en anglais, pour renforcer votre impact sur le marché français.

Puis, pour éviter tout ennui, n'oubliez pas d'inclure un dernier paragraphe, écrit en petits caractères, qui :

- protégera la propriété littéraire du texte, contre toute tentative de plagiat;
- dissuadera toute victime d'une erreur éventuelle d'exercer à votre encontre le moindre recours.

Pour ceux qui ne liraient pas encore couramment l'anglais, en voici la version française :

En décodant la langue de bois, vous en percevrez l'essentiel.

Nous n'avons pas pris la peine de vérifier des informations prélevées, ici et là, chez des gens qui nous ont paru sérieux. Il se peut que ces informations soient imprécises, tronquées et fantaisistes. En conséquence, les informations, que vous avez achetées, circulent désormais à vos seuls risques et périls. En outre, nous nous autorisons à changer d'avis, sans crier gare, quand bon nous semblera. ▲

imaginé par Alain Plagion