MASTER EN SCIENCES INFORMATIQUES

Spécification et génération de serveurs de Business objects

Wigny, Xavier

Award date:
2000

FACULTÉS UNIVERSITAIRES NOTRE-DAME DE LA PAIX,

NAMUR

INSTITUT D'INFORMATIQUE

RUE GRANDGAGNAGE, 21, B-5000 NAMUR (BELGIUM)

Spécification et génération de serveurs de Business Objects

Xavier Wigny

Mémoire présenté en vue de l'obtention du grade de
Maître en Informatique

Année Académique 1999 - 2000

Résumé

Actuellement, la communauté objets est en train de modifier l'utilisation qu'elle fait des objets. Jusqu'à présent les concepts représentés par les objets des diverses applications restaient au niveau informatique. On utilisait un objet pour la gestion de l'affichage, un autre pour stocker temporairement de l'information de la base de données... La tendance actuelle est à la représentation, à l'aide d'objets, de notions qui ont un sens dans le processus d'affaires. Dans cette optique, un objet représente par exemple un client, ses commandes et les règles d'affaires qui régissent son comportement (elles sont décrites sous forme de méthodes). Ces objets d'un type particulier sont appelés des business objects.

Ce mémoire présente un méthode de développement d'applications utilisant les business objects et les bases de données relationnelles. Cette méthode débute par la création de prototypes d'écran et l'identification des business objects qui composent les écrans. C'est à partir de ces business objets que sont générés des sous-schémas conceptuels. Ces derniers seront intégrés ensemble pour former un schéma conceptuel global qui servira de base à la création de la base de données. Ensuite, le business object proprement dit va être créé, c'est-à-dire la structure d'information, les méthodes qui règlent son comportement ainsi que les requêtes qui permettent d'échanger l'information contenue dans l'objet avec la base de données. Pour terminer, les écrans finals seront construits à partir des prototypes et des changements engendrés par les étapes précédentes. Le tout sera intégré dans une architecture distribuée. Divers outils complémentaires à l'atelier logiciel DB-MAIN ont été développé afin d'automatiser au maximum cette méthode.

Abstract

Currently, the « object » community is working on modifying the utilization it makes of « objects ». Until now, the concepts represented by the objects of the different application software's were restricted to the « computer science » domain. Some object was used for display management, some other object for temporary storage of database information... The current tendency relates to representing, via objects, notions that have some sense within the business process. With this in mind, an object represents, for example, a client, his orders and the business rules (described through methods) that govern its behaviour. These specific objects are called « business objects ».

This dissertation presents a development methodology for software applications relying on business objects and relational databases. This methodology begins by creating screen prototypes and identifying the business objects that compose the screens. In turn, the previously identified business objects permit to produce conceptual sub-schemas. The latter's will be integrated in order to form a global conceptual schema, which will be the basis for creating the database. After that, the actual business object will be created, i.e. the information structure, the methods controlling its behaviour as well as the queries allowing exchanging the information contained in the object between the object and the database. Finally, the final screens will be built based on the prototypes and the changes produced by the previous steps. All the parts will be integrated into a distributed architecture. Several complementary tools to the DB-MAIN CASE tool have been developed in order to maximize the automation of this methodology.

Avant - Propos

Je remercie vivement Monsieur le Professeur Jean-Luc HAINAUT qui a supervisé ce travail pour sa disponibilité et ses conseils avisés.

Je tiens également à remercier Virginie et Steve sans qui ce mémoire n'aurait probablement jamais existé. Merci également à Olivier pour sa compréhension ainsi qu'à toutes les personnes qui, de près ou de loin ont contribué à la réalisation de ce travail.

Pour mes parents et Virginie...

Chapitre 1

Introduction

1.1 CONTEXTE

La création de plus en plus rapide d'applications toujours plus complexes et aussi proches que possible des besoins de l'utilisateur voilà sans doute l'objectif que de nombreux informaticiens tentent d'atteindre depuis des décennies. Pourtant tout avait assez mal commencé avec les langages informatiques de bas niveau tel que le code machine ou l'assembleur. Heureusement, ceux-ci se sont rapidement développés jusqu'à gagner le qualificatif « évolués » offrant un niveau d'abstraction certain par rapport au code machine. Ces langages sont divisés en trois grandes familles : les langages procéduraux (C), les langages orientés-objets (C++) et les langages orientés-listes (ProLog). Chacune de ces familles a été accompagnée de diverses méthodes de conception d'applications.

Un système d'information de qualité apporte à une entreprise un avantage compétitif indéniable. L'informatique d'une société est devenue au fil des ans un élément essentiel à sa survie, à tel point qu'aucune ne peut plus prétendre pouvoir subsister si elle devait en être privée un long moment. De plus, une organisation qui possède une infrastructure informatique adéquate (matérielle d'une part mais essentiellement logicielle) pourra rapidement profiter des nouvelles opportunités d'affaires ainsi que s'adapter aux changements des marchés sur lesquels elle évolue. En règle générale pourtant, les systèmes d'information des entreprises sont très complexes et trop rigides pour permettre une adaptation rapide.

L'orienté objet existe depuis plus de vingt ans (le C++ a été développé par Bjarne Stroustrup dans les laboratoires Bell au début des années 1980 [ENC98]) et pourtant il n'a commencé à être largement utilisé qu'au début des années 1990. Les objets ont été popularisés avec le développement des interfaces homme-machine graphiques où l'utilisateur pouvait interagir avec eux. Ce type d'IHM a été développé par les laboratoires Xerox avant d'être popularisé par Apple.

Derrière le terme « orienté-objet » se cache un concept de programmation structurée autour d'un ensemble de composants interagissant entre eux et appelés objets. Dans ce courant de programmation, un objet peut être vu comme un élément composé de deux parties complémentaires. La première est une structure de données tandis que la seconde se compose de méthodes définies par « l'interface ». Cette interface spécifie ce qu'il est possible de faire avec l'objet et comment l'utiliser. Le but des langages orientés-objets est donc de regrouper en une entité autonome l'ensemble des données et des fonctions qui réalisent une même tâche. Par rapport à un langage procédural classique les langages orientés-objets se trouvent à un niveau d'abstraction supérieur puisqu'une fois défini, un objet se présente au programmeur comme une boîte noire fournissant divers services.

Uniquement utilisée par des programmeurs marginaux[1] à ses débuts, la technologie objet fait maintenant partie de tous les développements importants d'applications. Elle permet de créer des programmes dont certaines parties sont

[1] Ce terme n'a ici rien de péjoratif.

réutilisables ou sont déjà réutilisées ce qui fournit l'assurance qu'elles sont pratiquement sans erreur.

Paradoxalement, le modèle objet n'a été que peu utilisé pour représenter les véritables éléments du processus d'affaires alors qu'il y convient parfaitement. Un objet peut représenter un client, une commande, une facture ou même des évènements du processus d'affaires tel qu'un achat qui regroupe une commande de produits faite par un client et qui nécessite une facture. En fait, ces objets peuvent être l'abstraction de tout élément qui a un sens dans le processus d'affaires. Ils portent le nom de « business objects »

1.2 LES BUSINESS OBJECTS

L'OMG (Object Management Group) définit un business object comme « Une représentation d'une chose active dans le domaine d'affaires qui possède au moins des attributs, un comportement, des relations et des contraintes. [...] Des exemples typiques de business objects sont : un employé, un produit, une commande. » Cette définition souligne l'importance d'une correspondance entre l'élément réel présent dans le domaine d'affaires et sa modélisation informatique.

1.3 BUT ET ETENDUE DU TRAVAIL

L'objectif de ce travail est de décrire et surtout d'implémenter une méthode de conception d'applications reposant d'une part sur les business objects et d'autre part sur les bases de données. L'originalité de la méthode vient du fait que celle-ci est plus ou moins automatisée selon les étapes et qu'elle vient se greffer à l'atelier logiciel DB-MAIN dont elle en étend ainsi les possibilités déjà nombreuses. La conception assistée d'applications est un sujet tellement vaste que son étude complète nécessite bien plus qu'un simple mémoire. Aussi, dès le début deux directions étaient envisageables. La première était de choisir une étape du processus de création d'applications, de la décrire en détail et de proposer un outil informatique performant capable de gérer tous les cas pouvant être rencontrés par l'utilisateur. La seconde consistait à couvrir l'étendue de la démarche même si avant d'avoir commencé nous savions pertinemment bien que tout ne pourrait être abordé en profondeur. C'est pourtant cette deuxième route que nous avons choisie. Tout d'abord elle apporte au lecteur une vision globale d'un processus qui est encore peu connu et nous espérons susciter chez lui l'envie d'ajouter une brique aux fondations que nous avons érigées ou alors d'en modifier une existante afin que l'édifice soit plus solide.

En quelques mots, la méthode dont il est question ici prend son origine à partir de la conception de maquettes d'écran comme c'est le cas lors de développement classique avec des outils comme Delphi. A partir de ces maquettes sont générés des sous-schémas conceptuels et les business objects de notre application sont identifiés. L'étape suivante est l'intégration de ces sous-schémas en un schéma conceptuel global normalisé et le développement d'une base de données relationnelle. La phase suivante définit l'objet technique qui représente une instance du business object en mémoire. C'est également elle qui rédige les requêtes SQL qui garnissent cet objet à partir de la base de données que nous venons de créer. Pour terminer, il faut rédiger les méthodes à attacher aux business objects et intégrer le tout dans une architecture de distribution.

Le chapitre 2 définit formellement le concept de business objects, introduit ses proches parents que sont les frameworks et les patrons d'ingénierie. Dans ce chapitre sont aussi chiffrés les apports d'un développement d'applications en orienté-objet. Il se termine en positionnant les business objects dans l'ensemble du système d'information.

Le chapitre 3 explique de manière théorique l'ensemble de la méthode utilisée tout au long de ce mémoire. Celle-ci se compose de onze étapes qui s'étendent de la création de maquettes d'écran jusqu'aux tests finaux et la documentation en passant par la conception de la base de données.

Le quatrième chapitre décrit les étapes nécessaires à l'implantation d'une architecture business object dans un environnement client/serveur. La technologie retenue pour cela est RMI. Ses principes en sont rapidement présentés et une application basique mais complète reposant sur cette technique y est développée. A ce point du travail, une réflexion est menée sur l'influence du design des objets distribués sur l'efficacité du système.

Le chapitre 5 explique en détail les différentes fonctions d'une application Voyager qui permettent de transformer une maquette d'écran en un sous-schéma conceptuel. Dans ce chapitre sont définies les règles à respecter lors de la conception de ces maquettes. Le code de cette application se trouve à l'annexe 1.

Lors de l'intégration de sous-schémas conceptuels, de nombreux changements sont effectués et il est impossible de connaître ce qu'est devenu un objet du schéma. Ce détail est très gênant si l'on désire qu'à l'exécution de l'application ses business objects soient enregistrés dans les tables qui leur sont attribuées. Le sixième chapitre décrit comment fonctionne une application Voyager qui résout le problème. Le code de cette application est présent à l'annexe 2.

Le septième chapitre est consacré à l'étude d'un générateur de composeur/décomposeur de business objects. Le composeur/décomposeur a pour fonction d'instancier en mémoire les business objects et de gérer l'information qu'ils contiennent (i.e. transférer les données de la base de données vers le business object et vice-versa). Ce chapitre s'attarde particulièrement sur un prototype d'applications qui produit automatiquement des requêtes SQL.

Le chapitre 8 propose une étude de cas qui au travers de la création d'une application simple de gestion d'ouvrages littéraires met en pratique la méthode qui est décrite de façon théorique au chapitre 3. Les programmes construits et décrits lors des trois chapitres précédents (5, 6 et 7) sont utilisés pour automatiser certaines phases du développement de l'application de cette étude.

Finalement, le chapitre 9 tire les conclusions de ce travail et introduit les éventuelles suites à lui donner.

Chapitre 2

État de l'art

Objectif :

L'objectif de ce chapitre est d'expliquer et définir le concept de business objects ainsi que leur environnement, de montrer leurs apports et de décrire une architecture qui les utilise.

2.1 L'IDÉE FONDATRICE DES BUSINESS OBJECTS

Le concept de spécifications business, c'est-à-dire l'explicitation du processus d'affaires dans les termes du domaine d'applications, a été introduit pour faciliter la description et la compréhension des ces processus d'affaires et ce indépendamment de toute plate-forme informatique. Les spécifications business établissent un lien clair et non ambigu entre les spécialistes du domaine et les informaticiens chargés de son automatisation. De plus, elles apportent aux responsables du business une vision formelle et globale d'un processus d'affaires qu'ils ne maîtrisent pas toujours bien.

La conception d'un logiciel passe par différentes étapes. Celles-ci se regroupent en deux grandes familles, la première est centrée sur la définition des besoins des utilisateurs tandis que la seconde vise le système informatique final. Ces étapes de conception trouvent leurs origines dans la modélisation, de plus en plus précise, des connaissances du domaine qui sont intégrées dans le schéma conceptuel du système d'information. Ce schéma sera par la suite transformé en un système opérationnel. Chaque étape peut avoir comme origine un modèle et comme destination un autre modèle. Le développement se fait alors en étapes incrémentales et il est possible à chaque moment de revenir en arrière si la voie choisie n'est pas la bonne.

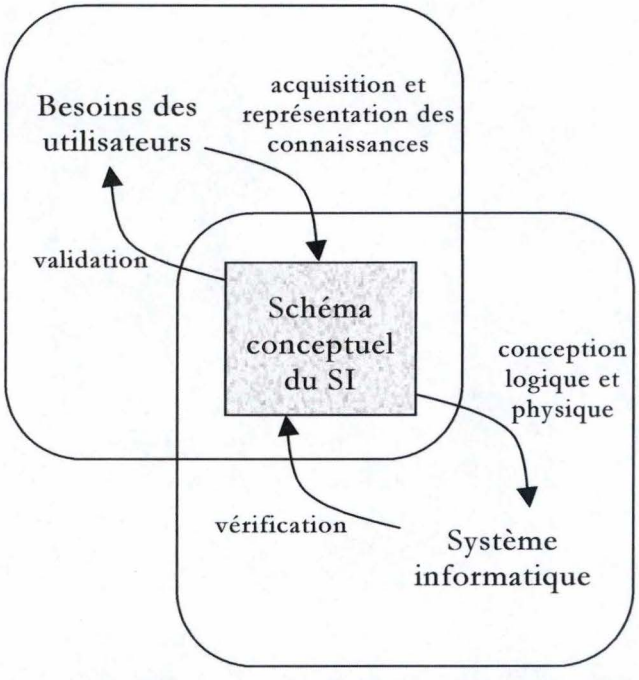


Figure 2.1 – Ingénierie d'un Système d'Information *[OUS99].*

Les business objects (BO) sont l'implémentation des spécifications business. Ils se composent de différentes classes et fournissent un ensemble de services. De par leur nature, les business objects rassemblent la connaissance du domaine mise en évidence par les spécifications (et conceptualisée dans le schéma conceptuel du système d'information) et comme l'indique leur nom, sont des objets (au sens technique du terme). Ce sont donc des entités quasi autonomes qui ont, entre autres caractéristiques, celles de pouvoir communiquer entre elles (via l'envoi de messages) ainsi que l'abstraction par les classes et l'encapsulation. La programmation objet facilite la conception grâce à la réutilisation de composants qui vise trois objectifs : la diminution des coûts de développement et de maintenance ainsi que leurs délais et l'amélioration de la qualité des logiciels puisqu'ils se basent sur des éléments maintes fois éprouvés. La ré-ingénierie et le développement rapide de fonctionnalité métier entraînent de nombreux et importants changements dans les méthodes de conception et sur le marché des outils CASE.

2.2 BUSINESS OBJECTS, PATTERN ET FRAMEWORK

L'étendue des objets réutilisables est grande et s'étend des composants très techniques jusqu'aux composants très orientés « métier de l'utilisateur ». Par composant technique, il faut entendre par exemple un objet de gestion de liste chaînée, un module de tri (quick sort), etc... Le grand nombre de composants réutilisables pose le problème de leur archivage (dans des bibliothèques) et de leur mise à disposition. Internet a supprimé cet obstacle et c'est ainsi que l'on assiste au développement de site web[2] dont le métier est de vendre des objets métiers (business objects). Généralement, de telles bibliothèques de composants n'existent que pour la phase qui se trouve entre le schéma conceptuel du système et son implémentation (i.e. le code). Beaucoup de progrès sont encore à faire pour l'automatisation des phases d'analyse et de conception. C'est justement pour combler ce manque que le concept de patron (pattern) a été introduit.

2.2.1 Les patrons

Le but des patrons est de proposer des composants (ré)utilisables à partir de l'analyse des besoins. Pour ce faire chaque patron réunit un problème d'un domaine (qui se produit fréquemment) et au moins une architecture de solution qui doit être facilement implémentable pour chaque occurrence (parfois légèrement différente) du problème. Cette adaptabilité nécessite des moyens fournis avec le patron qui permettent de particulariser la solution à un problème spécifique. On peut alors définir un patron d'ingénierie comme « une **solution** à un **problème** dans un **contexte** ». Généralement, les patrons sont orientés vers un niveau du processus de conception d'un logiciel ainsi parle-t-on de patrons d'analyse, de patrons de conception ou de patrons d'implémentation.

---

[2] Par exemple, le projet IBM SanFrancisco propose un ensemble de composants Java destinés aux développeurs d'applications côté serveur. http://www-4.ibm.com/software/ad/sanfrancisco/about.html

La réutilisation de solution confronte le monde de la création de logiciel à deux processus de développement complémentaires :

- le développement *pour* la réutilisation qui oblige de penser les objets créés directement en composants réutilisables (identifier, spécifier et développer).
- le développement *par* la réutilisation qui impose de développer des applications en utilisant au mieux les objets créés ce qui parfois est moins commode que de les développer « from scratch ».

2.2.2 Les frameworks

Un framework peut être vu comme un patron d'étendue plus large (il peut être composé de plusieurs patrons). Le framework fournit une solution particulière à un problème qui appartient à une classe de problèmes classiques. Un framework, fidèle à sa définition[3], offre une ossature globale qui peut être assemblée avec d'autres. L'implantation d'un framework donne un logiciel exécutable ce qui n'est pas le cas d' un patron. Les patrons peuvent s'utiliser tout au long du processus de développement de logiciel alors que les frameworks sont généralement utilisés à partir de l'étape de conception comme le montre le graphique ci-dessous.

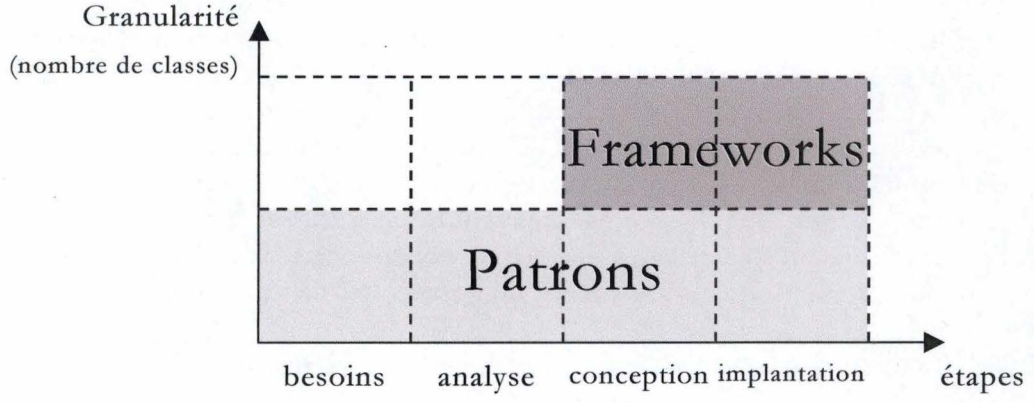


Figure 2.2 – Installation des composants dans le processus de conception *[OUS99].*

2.2.3 Les business objects

La conception de frameworks et de patrons est la base de la construction de systèmes de business objects. Notons, en passant, que la réutilisation de composants sera favorisée si l'on base la conception d'applications en terme de problèmes et non en terme de solutions.

L'approche objet est réputée comme étant la méthode qui gère le mieux la complexité, l'extension et l'adaptation, mais la réalité est différente : les applications développées de cette manière ont parfois une faible flexibilité et il peut être difficile de les adapter. Il y a principalement deux raisons à ce problème :

---

[3] Selon « Le Robert & Collins », framework = charpente, carcasse, ossature, encadrement, châssis.

- un objet est défini strictement alors que la réalité est plus nuancée. Le développement d'une hiérarchie de classes et sous-classes est une contrainte rigide.
- certains besoins peuvent ne pas être identifiés initialement (spécifications en cours de développement, incrémentielles) ou alors mal définis (nécessiteront une reformulation).

Définition : Un business object est défini comme étant la représentation informatique (il doit être supporté par une infrastructure technique) d'un élément actif du domaine d'affaires et qui a un sens pour les personnes compétentes dans le secteur.

2.3 APPORT DES BUSINESS OBJECTS

Le marché actuel (on cite souvent le concept de « mondialisation » et de « nouvelle économie ») est un environnement où les changements se font de plus en plus rapidement. Les organisations, principalement les entreprises, doivent régulièrement revoir, enrichir ou remodeler leur processus d'affaires si elles veulent encore croître ou tout simplement survivre. Ces modifications dans leur business doivent après acceptation par l'ensemble des responsables, être intégrées aux outils informatiques, nouveaux ou existants, que toute entreprise moderne se doit d'utiliser. C'est lors de cette étape de traduction des règles propres au processus d'affaires vers l'outil informatique que les business objects se révèlent particulièrement utiles et efficaces. Ils rassemblent les deux mondes que, malheureusement, tout tend à séparer : le monde réel et sa modélisation informatique. Le monde réel est le monde où existe le processus à modéliser et qui est généralement bien maîtrisé par ses membres (les décideurs, leurs collaborateurs, ...). L'autre monde, le monde informatique n'est historiquement compréhensible que par quelques initiés. Ces business objects, s'ils sont bien conçus, sont facilement adaptables aux changements du processus d'affaires.

2.4 LES BUSINESS OBJECTS DÉFINIS FORMELLEMENT

Pour le *Business Object Management Special Interest Group* un business object est la représentation d'un élément actif du domaine d'affaires qui possède au moins un nom, une définition (issue du domaine), des attributs, un comportement, des relations et des contraintes. La représentation d'un tel objet peut se faire en langage naturel, avec un langage de modélisation (UML) ou encore à l'aide d'un langage de programmation. Notons qu'il est préférable pour faciliter l'échange d'informations avec les personnes du domaine d'applications de n'utiliser que le langage naturel qui peut, à la limite, être complété par un langage de modélisation.

- Nom issu du secteur analysé : le nom d'un business object doit être le terme le plus utilisé, par les initiés du secteur, pour décrire le concept qu'il automatise. Un business object doit également posséder un identifiant unique. Le nom seul ne suffit généralement pas à l'identifier. Prenons l'exemple d'une entreprise qui désire gérer ses clients (des particuliers). Elle crée un B.O Client puis pour élargir son marché, se lance dans le commerce électronique orienté vers les entreprises (business to business). Elle possède maintenant des entreprises

comme clients et doit recréer un nouveau B.O Client orienté vers les entreprises (elle aura alors deux B.O Client). La valeur de l'identifiant utilisé doit de préférence être sans sémantique.

- Définition: la définition d'un business object doit se faire avec précision en collaboration avec les personnes coutumières du secteur. Elle doit se baser sur les définitions des données (dictionnaire des données) établies par le data administrator. Elle doit informer l'utilisateur sur l'étendue des fonctions apportée par l'objet métier. La définition d'un B.O. ordinaire (common business object[4]) doit expliquer la fonction de celui-ci dans l'ensemble des organisations sans se soucier des détails externes au B.O. et typiques à une organisation. Par exemple, un objet ordinaire de gestion des clients définie par une compagnie aérienne, doit gérer des clients et pas particulièrement des passagers.
- Attributs: les attributs relèvent des faits pertinents du business object qui lui permettent de réaliser son rôle dans le métier pour lequel il est défini.
- Comportement: la définition du comportement d'un business object est parfois appelée « business rules ». Les comportements conflictuels et exceptionnels doivent aussi être définis.
- Relations: par relations, on entend les relations entre différents business objects (dépendance, référence, encapsulation, ...).
- Contraintes: les contraintes limitent le champ d'applications du business object.

2.5 LES BUSINESS OBJECTS TECHNIQUEMENT

Techniquement, les business objects encapsulent un ensemble d'objets traditionnels qui chacun implémente une fonction de base du processus d'affaires. Par exemple ajouter une commande, modifier une commande ou encore consulter le solde du compte, etc...

2.5.1 L'architecture « business objects »

Lorsque le modèle d'affaires subit des modifications, les logiciels (de traitements, de gestion, d'analyse, etc...) doivent aussi être adaptés. Une architecture basée sur les business objects apporte une solution efficace pour une mise à jour rapide de l'outil informatique. Cette architecture prône la réutilisation du code existant et est généralement écrite en langages visuels ou à l'aide de langage objet classique. Le problème des langages visuels vient de la trop grande cohésion entre les procédures ou les fonctions et les éléments graphiques, ce qui ne nous met plus en présence d'une part d'un objet et d'autre part de son interface mais d'un tout indissociable. Ce problème ne se rencontre pas avec des langages objets classiques qui se prêtent donc mieux au développement d'architectures objets.

---

[4] Les common business objects apportent un ensemble d'objets de base aux développeurs d'applications qu'ils pourront adapter aux besoins de l'organisation qui les emploie. Ces objets peuvent être définis de manière large si le concept qu'ils représentent varie fortement d'une organisation à l'autre ou alors définis de manière stricte s'ils informatisent une notion stable à travers les entreprises.

2.5.2 Accroissement de la productivité

En programmation classique, les tentatives de ré-ingénierie du processus d'affaires (BPR) ont rapidement montré leurs limites. Ainsi, aux États-Unis, 80% des BPR ont échoué et même quand ils réussissent, leur coût est élevé puisque l'on cite le chiffre de 7$ par an pour la maintenance d'une seule ligne de code. 31% des projets sont stoppés avant même d'être terminés et 52,7% de ceux qui sont terminés dépassent leur budget de près de 200%. C'est la crise du logiciel que le développement objet tente de résoudre.

Le but d'une architecture centrée sur les business objects est d'améliorer aussi bien la vitesse de développement des nouveaux composants que celle de la mise à jour d'éléments existants. A ce niveau, comme le montre l'étude du groupe Software Productivity Research, c'est essentiellement le langage de programmation utilisé qui importe. Ce dernier conditionne évidemment la possibilité ou non de développer facilement des objets. L'étude montre ainsi que le taux de réutilisation des fonctions écrites en C est de 15%, ce qui est égal à celui de la plupart des autres langages de troisième. Seul Smalltalk améliore ce pourcentage de réutilisation de ses composants avec 50% de réutilisés dans les deux ans et 80% pendant la troisième année d'existence des objets. Qui dit réutilisation dit bien entendu diminution des coûts mais surtout diminution du temps de développement. Cette diminution de temps intéresse particulièrement les organisations qui, même si le coût est plus élevé, peuvent entrer les premières sur le marché, avoir une position enviable et généralement un retour sur investissement agréable. Prenons l'exemple de Microsoft qui a développé Internet Explorer en un temps record grâce à la réutilisation du système de navigation d'Encarta, ce qui lui a permis de ne pas se faire distancer par la concurrence (principalement Netscape).

L'autre avantage d'une architecture business objects c'est la faible dépendance qui existe entre les objets. Ainsi un objet qui n'est plus assez efficace pour assumer la montée en charge pourra être complètement reconsidéré sans compromettre le design général de l'architecture ou les autres objets.

2.5.3 Construction

Un business object est un rassemblement d'objets. Ce groupe doit se comporter comme un processus d'affaires de haut niveau et ses interactions (les données fournies en entrée, les résultats de l'exécution et les exceptions) avec l'extérieur doivent utiliser le langage du milieu d'affaires. Le fonctionnement interne du B.O doit être caché pour le monde extérieur (voir *Figure 2.3*).

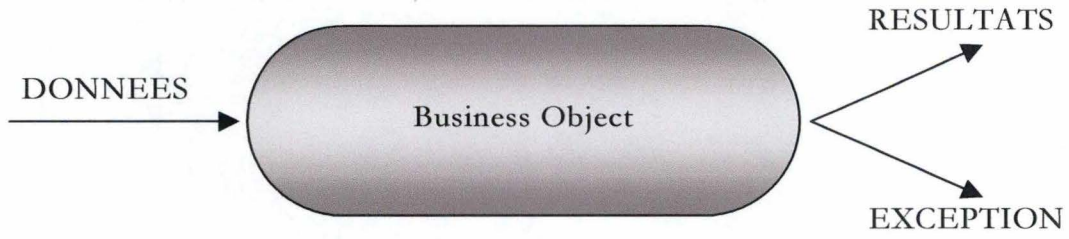


Figure 2.3 – Vue externe d'un business object.

Pour permettre la réutilisation de composants d'un B.O, ceux-ci doivent être encapsulés dans deux directions : l'extérieur ne doit rien connaître de l'intérieur des composants et le composant ne doit connaître de l'extérieur que les autres composants qui l'intéressent (voir *Figure 2.4*). Les bases de données externes doivent aussi être encapsulées dans un B.O pour que leur réutilisation soit facile et que toute migration d'un SGBD vers un autre soit possible. L'interface avec l'utilisateur doit aussi être séparée des autres objets, toujours pour des raisons de réutilisation et d'évolutivité. Grâce à cela, il est aussi possible de personnaliser les interfaces pour chaque fonction du processus d'affaires voire même pour chaque utilisateur.

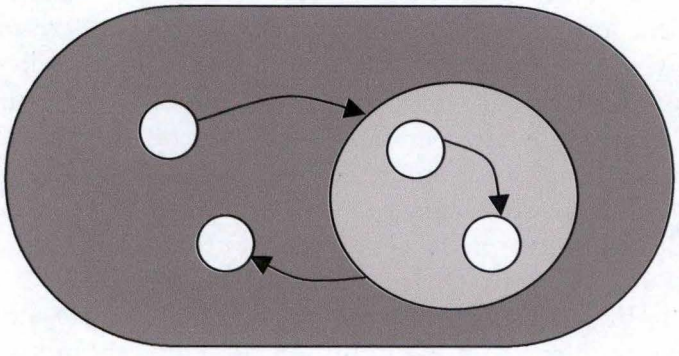


Figure 2.4 – Un business object et ses composants.

En résumé, une application classique[5] devrait se composer de trois couches distinctes : la couche interface homme-machine, la couche des composants modélisant le processus d'affaires et une couche d'accès aux bases de données qui sépare le cœur de l'application du type de base de données ainsi que la gestion des communications réseaux. Le chapitre 4 s'attarde à ce type d'architecture (principalement deux et trois tiers).

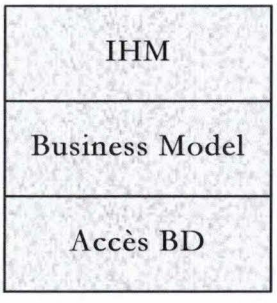


Figure 2.5 – Les couches d'une application classique.

[5] Par application classique il faut comprendre application centrée sur l'informatique de gestion.

2.5.4 Intégration des business objects dans le système d'information

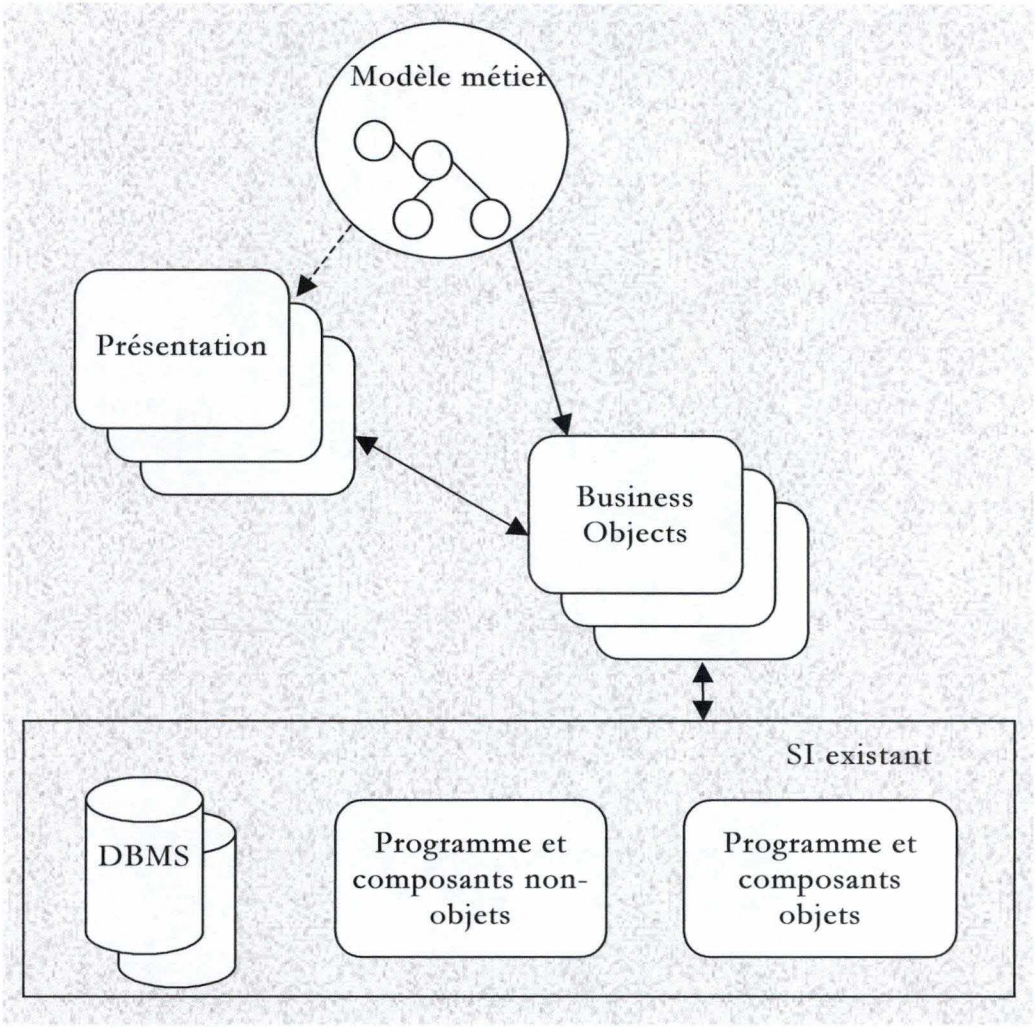


Figure 2.6 – Situation des business objects dans le système *[CAS]*.

Le modèle métier est le processus que le business object va essayer de modéliser. Une fois correctement décrit, il va servir de référence sur la manière dont est organisé et opère le processus d'affaires.

Les business objects représentent le modèle métier sous forme informatique. Il encapsule les règles relatives au métier. Certains business objects utilisent des fonctions qui existaient avant dans le **système informatique (SI)**. Ces fonctions peuvent être de type **objet ou non**. Quand un business object utilise une base de données (**DBMS**) il se positionne naturellement entre l'interface et la base de données de telle manière que le système est vu comme orienté objet et multi-tiers (trois tiers généralement).

La **présentation** permet de voir et de manipuler le(s) business object(s).

Chapitre 3

Méthodologie

Objectif :

Ce chapitre a pour but de présenter une méthode, plus ou moins automatisée, de création d'applications basées sur les bases de données relationnelles et les business objects introduits au chapitre précédent.

3.1 DESCRIPTION DE LA DÉMARCHE ADOPTÉE

Le but de la partie pratique de ce mémoire a été de développer différents outils complémentaires à DB-MAIN[6] qui automatise au maximum la création d'applications utilisant des bases de données. Ce processus permet d'un côté de diminuer sensiblement le temps nécessaire au développement d'applications mais de l'autre a le désavantage de fournir des logiciels parfois moins performants puisque les parties créées de manière automatique ou semi-automatique n'ont pas les mêmes prétentions que si elles avaient été écrites spécifiquement par un développeur.

3.2 ÉTAPES THÉORIQUES NÉCESSAIRES À LA CRÉATION D'APPLICATIONS REPOSANT SUR LES BUSINESS OBJECTS

La méthode de création d'applications que nous proposons repose sur les 11 points cités ci-dessous est expliquée en détail juste après :

1. Description des maquettes d'écran
2. Identification des business objects
3. Extraction de sous-schémas conceptuels
4. Intégration des sous-schémas en un schéma conceptuel global
5. Développement d'une base de données relationnelle
6. Un business objet, une vue
7. Aspect technique et implémentation des business objects
8. Méthodes attachées aux business objects
9. Construction des écrans de dialogue
10. Intégration dans une architecture distribuée
11. Tests, évaluation et documentation

---

[6] DB-MAIN est un outil CASE (Computer-Aided Software Engineering) développé par l'Institut d'Informatique des Facultés Universitaires Notre-Dame de la Paix, à Namur. DB-MAIN est un atelier logiciel qui permet (entre autres) à un développeur, de dessiner le schéma conceptuel d'un domaine d'applications et de générer les tables SQL qui lui correspondent.

3.2.1 Description des maquettes d'écran

La première étape de notre démarche est la description des maquettes d'écran. Cette tâche est semi-automatisée puisqu'il existe de nombreuses applications capables de créer des interfaces. Ces interfaces seront celles de la future application que nous sommes en train de développer. Cette étape ne diffère en rien à celle d'un développement rapide d'applications (RAD) classiques.

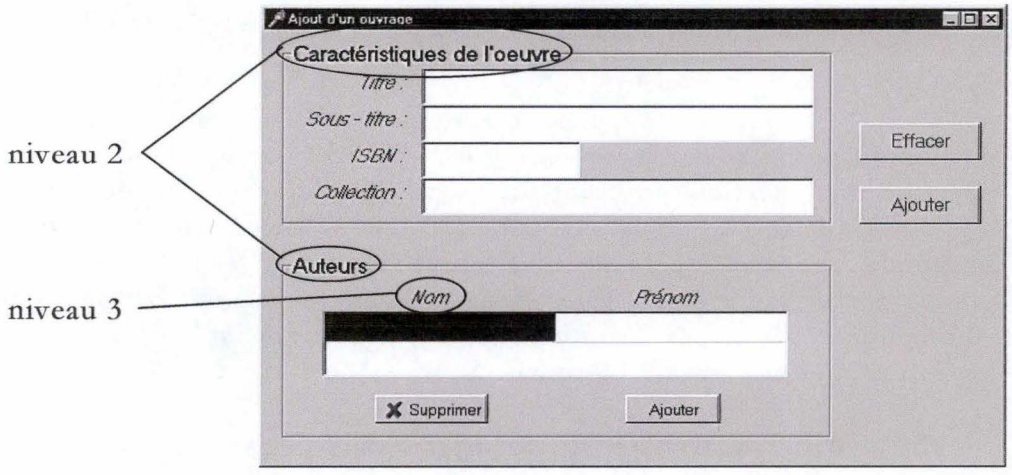


Figure 3.1 – Exemple de maquette d'écran.

Sur la *Figure 3.1*, on distingue deux zones ('Caractéristiques de l'œuvre' et 'Auteurs') définies par des boîtes de regroupement. Chaque zone de regroupement sera traduite automatiquement en un attribut composé de tous les champs (boîte d'édition) qu'elle contient. Cette technique de regroupement permet à l'utilisateur de réunir les objets qui sont sémantiquement proches. Ce regroupement devra si possible être conservé durant la suite du développement. Au travers ces rassemblements il est possible de voir une certaine hiérarchie. Chaque composant de la maquette d'écran est à un niveau connu et chaque composant possède un et un seul supérieur direct. Seul le composant de type « fenêtre » n'en possède pas. Il est de niveau 1 et son équivalent dans un schéma conceptuel est une entité. Dans notre exemple, 'Auteurs' est de niveau 2 et est composé d'un champ nom et prénom tous deux de niveau 3. En général, tous les champs d'un niveau n ($n \geq 2$) caractérisent l'attribut de niveau n-1 qui les contient.

Pour chaque champ, il faut définir précisément les caractéristiques qu'il possède c'est-à-dire son type de valeur et sa sémantique. La sémantique n'est pas obligatoire mais permet aux personnes extérieures au domaine de comprendre plus facilement comment se comporte le programme. Le type des valeurs des champs est quant à lui essentiel puisque c'est grâce à cette information qu'il sera possible d'insérer la valeur qu'il contient dans une table (SQL) de la base de données. Dans notre exemple (*Figure 3.1*), le numéro ISBN est une chaîne de caractères de longueur maximum égale à 10.

Aux événements des composants de l'écran[7] seront assignées soit des fonctions typiques à l'interface (ex : effacer le contenu de la fenêtre, fermer la fenêtre, activer une autre fenêtre) ou alors des méthodes attachées aux business

[7] Les boutons en font évidemment partie.

objects. Ce sont ces dernières qui nous intéressent le plus. Dans l'exemple de la *Figure 3.1* l'utilisation du bouton 'Ajouter' de droite va déclencher l'exécution de la méthode d'insertion d'un nouveau business object de type 'ouvrage' dans la base de données. Cette méthode d'insertion peut implémenter des règles d'affaires plus ou moins complexes. Par exemple l'ajout d'un nouvel ouvrage peut générer l'envoi automatique d'e-mail à tous les clients qui ont manifesté un intérêt particulier pour les nouveautés de cette collection. Ces méthodes font partie au même titre que les règles de cohérence et de comportement des règles d'affaires (business rules). Celles-ci gèrent le comportement du/des business object(s). Certaines règles peuvent être introduites dans DB-MAIN (contraintes d'intégrité, cardinalités minimum et maximum, ...), les autres seront intégrées au code même de l'application. Ces dernières doivent, à l'heure actuelle, toujours être programmées « à la main » même s'il n'est pas improbable qu'il existe un jour un outil capable d'aider l'analyste dans la découverte et l'écriture de ces différentes règles. Nous considérons que cet aspect mérite à lui seul tout un travail et c'est alors naturellement que nous n'investiguerons pas plus dans cette direction.

3.2.2 Identification des business objects

Après avoir créé et décrit les prototypes d'écran, il faut que le concepteur de l'application identifie les business objets de base puisqu'un écran est un business object, souvent complexe, qui peut en regrouper plusieurs plus simples (au sens où ils ne sont plus composés). La recherche des business objects de base possède un double avantage. D'une part, elle permet au modèle informatique de représenter au mieux le réel qu'il implémente (plus facile à comprendre, plus facile à faire évoluer et plus stable dans le temps). D'autre part, les business objets vont être traduits en tables dans le modèle relationnel. Un gros BO, qui représente un conglomérat de plus petits, engendrera toujours un schéma dénormalisé comme le montre le tableau ci-dessous :

Titre	Sous Titre	ISBN	Collection	Nom de l'auteur	Prénom de l'auteur
Java 2	Édition 1999	2212250223	OEM	Mirecourt	Antoine
Delphi	2	2742906754	PC Poche	Rensmann	Jörg
Delphi	2	2742906754	PC Poche	Herbers	Jörg
Delphi	2	2742906754	PC Poche	Herbers	Jens
...	...	...	...	...	...

Figure 3.2 – Table dénormalisée qui enregistre l'information contenue dans un business object composé.

3.2.3 Extraction de sous-schémas conceptuels

Cette troisième étape a pour but de construire un sous-schéma conceptuel à partir d'un prototype d'écran. Pour se faire nous allons utiliser les notions de niveau et de regroupement introduites au point 3.2.1 (page 20). L'extraction fonctionne comme suit : à partir d'une fenêtre bien construite (cette notion est précisée au paragraphe ci-dessous) il faut rechercher quelle zone de texte explique quelle boîte d'édition. Sur la *Figure 3.1* la zone de texte 'Titre :'

explique la boîte d'édition située à sa droite. Il arrive qu'une zone de texte soit présente juste à titre informatif, elle n'est alors en relation avec aucune boîte d'édition. Dans ce cas, elle ne doit pas être prise en compte. Dès que toutes les correspondances sont établies, l'extraction peut-être effectuée. Celle-ci va à partir de la maquette d'écran créer une entité dont le nom est le titre de la maquette d'écran. Des attributs simples de niveau n ($1<n\leq$*nombre de niveaux*) vont être créés à partir de toutes les boîtes d'édition de niveau n. Le nom d'un attribut est celui de la zone de texte qui est liée avec le composant qui a généré cet attribut. Une boîte de regroupement de niveau n va être représentée dans l'entité par un attribut composé de niveau n. Cet attribut va être composé d'attributs de niveau $n+1$ représentant les composants de niveau $n+1$ de la boîte de regroupement et ainsi de suite pour tous les niveaux comme le montre la *Figure 3.3* (ci-dessous).

Chaque prototype d'écran bien construit a toujours un et un seul sous-schéma conceptuel qui lui correspond. Par bien construit il faut entendre le respect de certaines règles. D'une part il y a la règle de bonne utilisation des niveaux. Celle-ci impose que deux composants d'un même niveau ne possèdent pas le même nom. D'autre part le nombre de descriptions des zones d'édition doit être égal au nombre de ces zones. La *Figure 3.3* (ci-dessous) montre sur un exemple simple, l'extraction d'un sous-schéma conceptuel à partir d'une maquette d'écran.

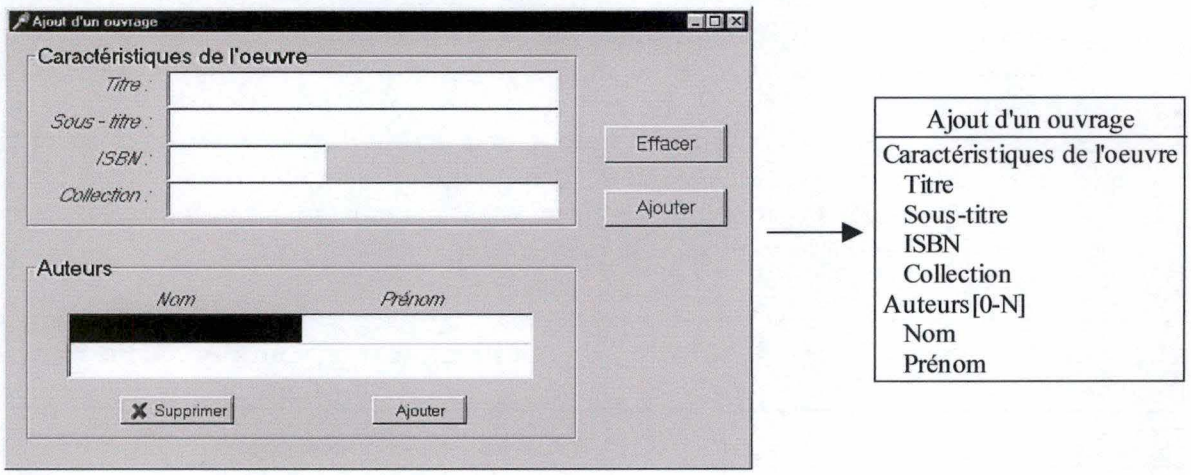


Figure 3.3 – Transformation d'un prototype d'écran en un sous-schéma conceptuel.

3.2.4 Intégration des sous-schémas en un schéma conceptuel global

Après avoir pour chaque prototype d'écran construit son sous-schéma conceptuel qui lui correspond, il faut intégrer l'ensemble de ces sous-schémas en un seul schéma conceptuel global normalisé. DB-MAIN dispose d'un module dédié à l'intégration. Celui-ci propose différentes transformations qui permettent dans un premier temps de fusionner les sous-schémas et ensuite de normaliser le schéma global. Cette intégration reprend l'approche classique d'utilisation des sous-systèmes mais appliquée à des sous-schémas représentant des business objects. Parmi les transformations les plus utilisées il y a la transformation d'une entité en une relation, d'une entité en un attribut, l'introduction d'un super-type, l'éclatement ou la fusion d'entités, l'ajout d'un identifiant technique et

généralement l'inverse de toutes ces fonctions. Pour plus de renseignements sur le module d'intégration de DB-MAIN, le lecteur est renvoyé à l'aide en ligne de DB-MAIN ou à *[HAI99b]* .

3.2.5 Développement d'une base de données relationnelle

Le schéma conceptuel développé au point 3.2.4 n'est pas une fin en soi, et c'est tout naturellement que ce schéma doit être transformé en un schéma relationnel qui représente le mieux la manière dont l'information sera stockée dans le SGBDR. Cette transformation peut être automatique avec DB-MAIN mais il est aussi possible de ne l'automatiser que partiellement pour que le concepteur de la base de données garde un œil sur les transformations effectuées sur son schéma. A partir du schéma relationnel, DB-MAIN peut générer automatiquement le script qui servira à la création des tables et des différentes contraintes (insérées dans le schéma) dans le SGBDR.

3.2.6 Un business objet, une vue

Pour chaque business objet, on calcule la vue relationnelle qui le représente. Cette étape nécessaire est ardue puisqu'elle doit faire face à plusieurs problèmes.

- Premièrement, comme nous l'avons vu au point 3.2.2 il n'existe pas et il n'existera probablement jamais de moyen automatique pour déterminer les business objets de base qui composent un business objet. En effet, c'est la connaissance du domaine d'applications qui permet de savoir si un business object est atomique (=B.O. de base) ou composé. Le business object de base est la plus petite unité d'échanges d'informations entre l'utilisateur final et la base de données. Dans notre démarche, l'identification des B.O de base n'est pas essentielle et pour simplifier, on peut dire qu'un business objet de base est l'ensemble des champs d'un prototype d'écran.

- Deuxièmement, les sous-schémas issus des prototypes d'écran ont été intégrés en un schéma conceptuel global, certains sous-schémas ont été fusionnés, d'autres ont disparus, certains encore ont perdu plusieurs de leurs attributs. D'une manière générale, les sous-schémas ont beaucoup évolué et il faut pouvoir faire la correspondance entre leurs états initiaux (donnés par les fenêtres) et ce qu'ils sont devenus après intégration (et les transformations qui l'accompagnent).

- Troisièmement, le principe de la vue SQL elle-même est problématique. La vue est très utile lorsqu'il est question de lire des informations stockées dans différentes tables. Une vue[8] crée une table virtuelle adaptée aux besoins de l'utilisateur dont seul le format est stocké. Les données issues des tables réelles ne sont pas dupliquées mais uniquement regroupées à la demande (lors des requêtes). Si la vue rend le remplissage des business objects possible, il n'en est pas de même dans l'autre sens, c'est-à-dire lorsque l'on veut insérer de nouvelles informations (nouvelles ou des anciennes mises à jour) dans la base de données. En effet, imaginons que nous avons deux tables, une table client et une table commande. Une instance d'un business object « client »

[8] définition composée à partir de *[HAI94]*, pour plus de détails, nous renvoyons le lecteur à cet ouvrage.

reprend le client ainsi que toutes ses commandes. Les tables client et commande sont liées entre elles par une clé étrangère située dans la table commande et qui reprend l'identifiant du client. Une vue qui nous donne le nom, le prénom et l'adresse du client ainsi que les numéros et le prix total de chacune de ses commandes nous suffit. Nous allons donc créer une vue qui reprend ces informations. Malheureusement une fois créée, cette vue ne nous permet pas d'insérer la nouvelle information issue de notre business object car il nous manque l'identifiant du client qu'il faut absolument insérer dans la table commande pour que la commande puisse référencer le client auquel elle est attachée. La solution est de mettre dans la vue cet identifiant. On découvre ici que le problème n'est pas uniquement un problème d'identifiant mais plus généralement celui des colonnes obligatoires (colonnes NOT NULL) dans les tables. Et des colonnes obligatoires, il y en a beaucoup.

- Le quatrième et dernier problème concerne la réelle perte d'énergie qu'occasionne le passage par une vue SQL. Reprenons l'exemple du point situé ci-dessus. On dispose d'un business object « client » qui contient les données du client (nom, prénom, adresse) ainsi que ses commandes (voir *Figure 3.4*). Le passage par une vue SQL donne comme résultat un ensemble de lignes contenant bon nombre d'informations redondantes et donc inutiles (comme le montre la *Figure 3.5*). De plus, si on suit le cheminement des données lors de l'utilisation d'une vue, au départ les données utiles au business object « client » sont séparées en deux tables. La vue va les regrouper en une seule puis elles vont être de nouveau séparées pour être insérées dans le business object. En d'autres termes, l'insertion des données dans le business objet défait tout le travail de la vue.

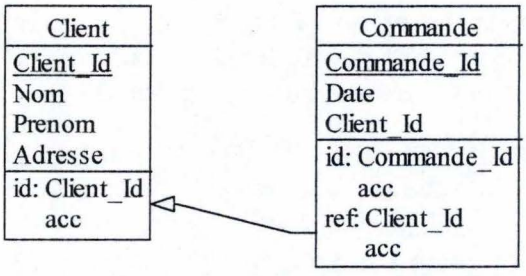


Figure 3.4 – Tables utilisées par le business object « Client ».

Nom	Prénom	Adresse	Commande_Id	Date
Dupont	**Jean**	**12, rue des roses**	**472**	**22/09/97**
Dupont	***Jean***	***12, rue des roses***	**785**	**13/01/98**
Dupont	***Jean***	***12, rue des roses***	**1025**	**09/07/98**
Dupont	***Jean***	***12, rue des roses***	**1358**	**17/11/99**

Figure 3.5 – Résultat de la vue du business object « Client ». Le caractère italique de certaines données indique qu'elles sont superflues.

La solution à tous ces problèmes est de se passer des vues pour la lecture mais aussi pour l'insertion (i.e. l'écriture) d'informations de la base de données. Cela implique la création d'autant de requêtes SQL pour un seul business

object qu'il n'y a de tables qui contiennent l'information que ce dernier utilise. La procédure qui chargera et déchargera l'information dans le business object devra connaître une plus grande partie de la base de données puisqu'elle devra continuellement accéder à différentes tables pour chaque BO. Elle sera de ce fait un peu plus compliquée. Pour le reste, rien ne change.

3.2.7 Aspect technique et implémentation des business objects

Un business object peut être représenté sous une forme d'arbre. C'est un objet et donc il possède des champs où sont stockées des données ainsi que des méthodes pour agir sur ces données. En fait, un business object est une structure d'informations à laquelle sont attachées des méthodes. Cette structure peut être définie à l'aide de différents langages, par exemple en Java.

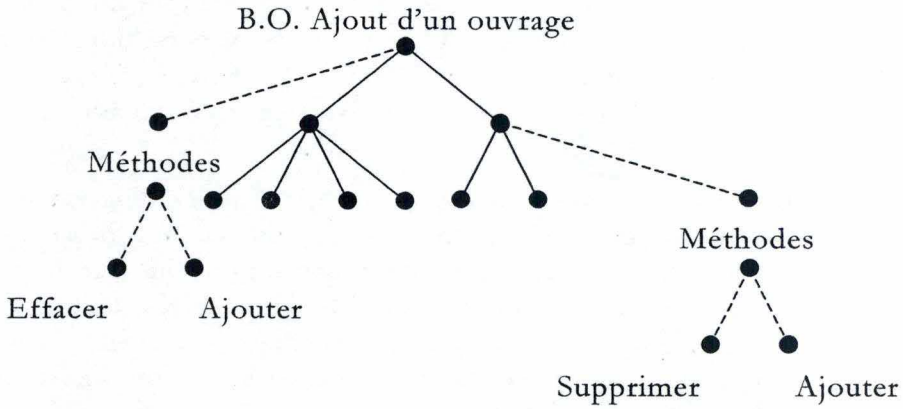


Figure 3.6 – Exemple de représentation de business objects.

Une fois la structure établie, il faut une procédure qui à chaque utilisation d'un business object l'instancie en mémoire. C'est le rôle du composeur/décomposeur. Il garnit également l'objet suivant la technique vue au point 3.2.6.

3.2.8 Méthodes attachées aux business objects

A cette étape, il faut rédiger les méthodes à attacher aux business objects. Ces méthodes sont l'implémentation des règles d'affaires déjà introduites au point 3.2.1. Pour l'instant certaines règles peuvent être décrites à l'aide de DB-MAIN, les autres doivent être écrites « à la main ». Comme règles pouvant être décrites par DB-MAIN il y a entre autres les cardinalités minimum et maximum (une commande doit au moins posséder une ligne de détails de commande), les clés étrangères (une commande doit appartenir à un client), les colonnes obligatoires (un client doit avoir un nom, un prénom et une adresse), etc... Certaines méthodes qui implémentent les règles d'affaires sont assignées à des boutons de l'interface (par exemple ajouter un ouvrage), les autres sont indépendantes de l'interface et sont déclenchées par d'autres règles suivant la situation. Par exemple la vente d'un produit peut faire passer le stock sous son niveau minimum ce qui provoque une commande automatique de ce produit auprès du fournisseur.

3.2.9 Construction des écrans de dialogue

L'étape de construction des écrans de dialogue a pour objectif de générer les écrans de l'application à partir des maquettes d'écran qui ont pu subir différentes modifications au cours des étapes précédentes.

Le but est que les écrans générés soient le plus proche possible des prototypes construits au point 3.2.1. Dans la majorité des cas, ils devraient être identiques. Les modifications peuvent avoir deux origines :

- De mauvaises maquettes d'écran peuvent vite montrer leurs limites. Par exemple un non respect de la hiérarchie dans les maquettes d'écran empêche la génération des sous-schémas conceptuels. Il faut alors retravailler les prototypes d'écran.
- L'utilisation d'outils différents entre la conception des prototypes d'écran et l'application finale. Par exemple un composant peut ne pas exister dans le langage utilisé pour cette dernière alors qu'il était présent pour la construction du prototype d'écran. Dans ce cas, le composant manquant doit être remplacé par un autre ou développé (ce qui est plus coûteux en temps).

La *Figure 3.7*, ci-dessous, montre les étapes déjà effectuées par notre démarche. Sur ce schéma, on remarque que la construction des écrans de dialogue est sur l'autre versant de la démarche de construction d'une architecture basée sur les business objects que la description des maquettes d'écran. Beaucoup de modifications peuvent avoir lieu entre ces deux extrêmes. Néanmoins, il est essentiel que l'écart entre les écrans prototypes et les écrans finals soit minimum.

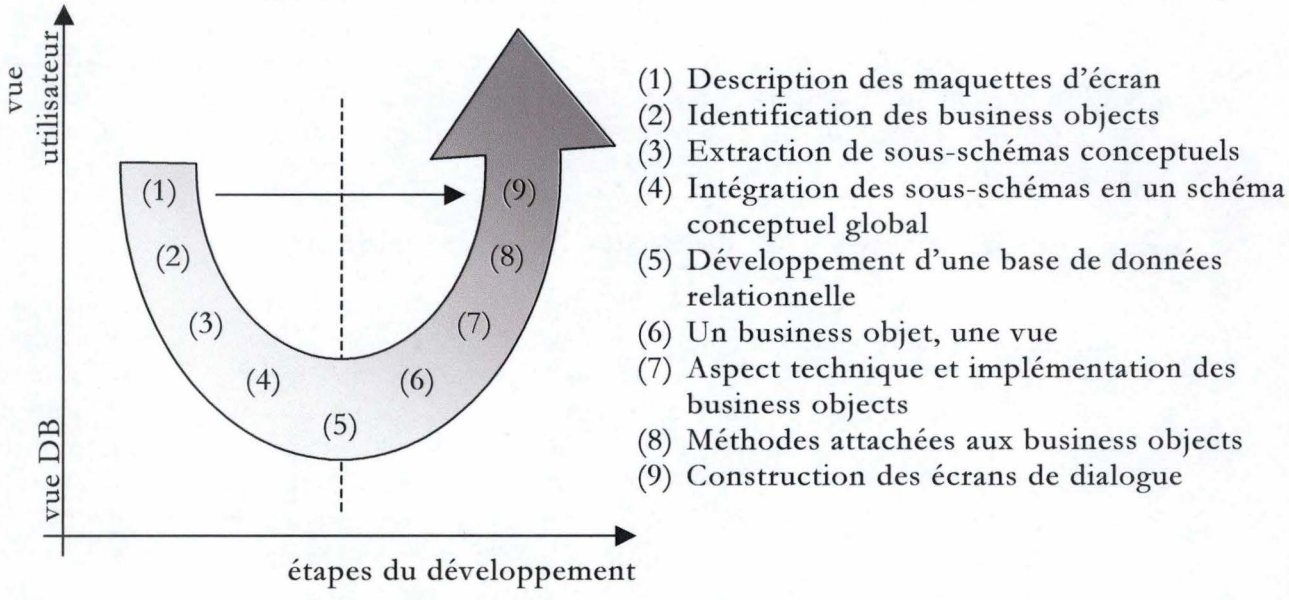


Figure 3.7 – Résumé de notre démarche.

3.2.10 Intégration dans une architecture distribuée

Une application utilisant de grandes bases de données ne peut-être développée sans tenir compte d'un environnement réseau, il en est de même pour les business objects. La structure même d'un business object le rend relativement autonome ce

qui facilite son échange sur le réseau. L'idée est d'intégrer le modèle business object dans un environnement client - serveur. Pour se faire, diverses méthodes existent Corba, DCom, RMI ou les Enterprise Java Beans. Le serveur est le seul à connaître l'existence de la base de données et le business object est exécuté sur le serveur. La seule manière pour le client d'interagir avec le BO est par l'envoi de messages. En retour, il recevra le résultat de sa demande. Pour le client, peu de choses diffèrent par rapport à un environnement où l'objet serait situé sur sa propre machine.

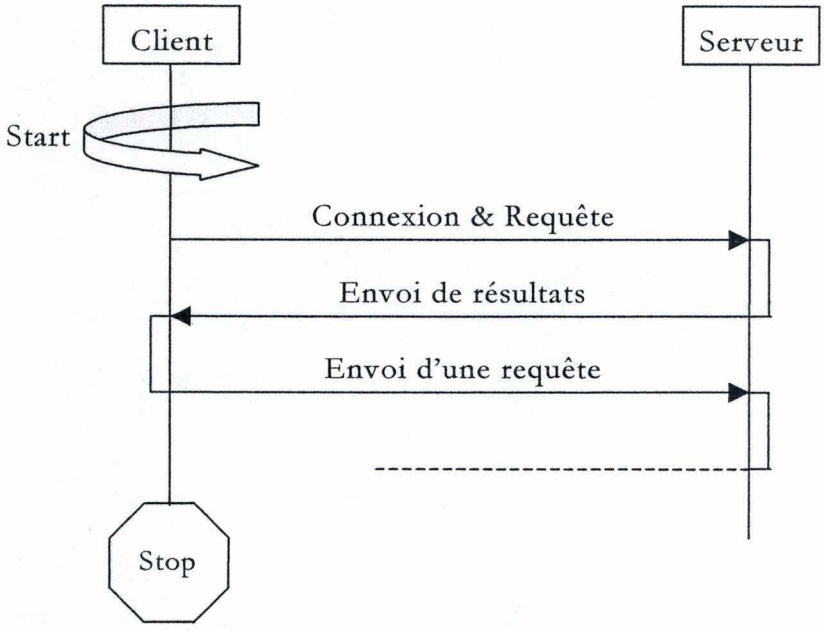


Figure 3.8 – BO dans un environnement client – serveur.

3.2.11 Tests, évaluation et documentation

Les dernières étapes du développement sont communes à tous les développements informatiques, ce sont les tests, l'évaluation et la documentation.

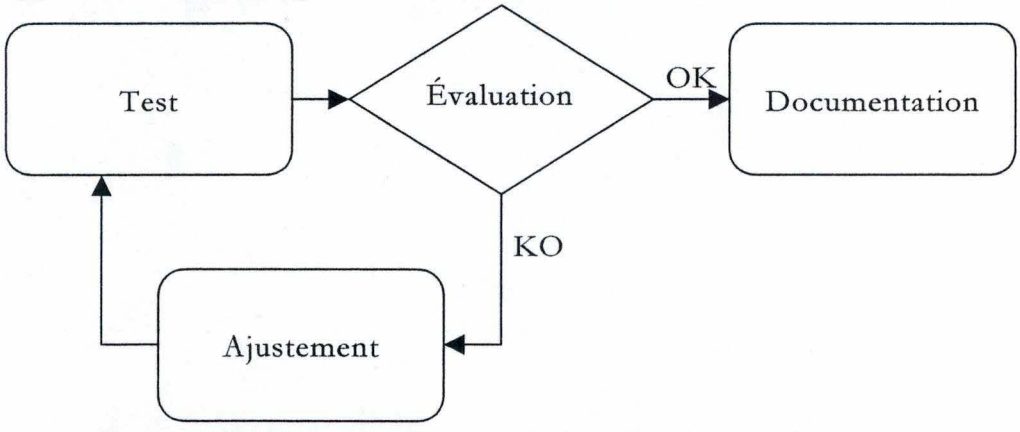


Figure 3.9 – Dernière étape du développement logiciel.

Même si cette étape ne peut pas être automatisée, elle peut assister l'utilisateur dans sa tâche. Lors des tests par exemple, une application d'aide aux tests pourrait indiquer à l'utilisateur quels ont été les business objects testés et ceux qu'il reste à vérifier. Elle pourrait aussi tenir un historique des jeux de valeur déjà utilisés. Ceci permettrait à l'utilisateur de ne pas rejouer deux fois le même

genre de test mais aussi de les relancer après avoir modifier l'application. L'autre point qui pourrait grandement être assisté est la documentation. Lors de la définition des business objects, l'utilisateur a dû décrire tous les types de valeurs qu'ils contenaient. Cette information est pertinente pour la documentation. Le module de documentation pourrait aussi facilement créer l'arborescence de chaque business object comme à la *Figure 3.6*. Cela rendrait la compréhension de l'application beaucoup plus aisée.

Chapitre 4

Architecture

Objectif :

Ce chapitre a pour objectif de proposer une solution d'implémentation de business objects dans un environnement concret et répandu : le client/serveur .

4.1 L'ARCHITECTURE CLIENT/SERVEUR

4.1.1 Caractéristiques d'une architecture client/serveur

A première vue, donner les caractéristiques qui permettent de déterminer qu'une architecture informatique est une architecture client/serveur est évident. Il suffit qu'elle possède au moins une partie client et une autre partie serveur qui s'exécutent sur deux machines reliées par un réseau. Déjà un problème se pose, celui de savoir ce qu'il en est si le programme « client » et le programme « serveur » s'exécutent sur la même machine. Ce petit exemple montre à quel point il est important de décrire de manière formelle (si cela est possible) les infrastructures de type client/serveur, puisque c'est sur elles que va reposer la quasi-totalité des utilisations de business objects.

D'une manière générale, les applications client/serveur[9] ont les caractéristiques suivantes :

- Les serveurs d'une architecture client/serveur ne sont pas uniquement des entrepôts où les différents clients se partagent des fichiers. Les serveurs peuvent exécuter des requêtes décrites en langage de haut niveau (SQL) et renvoyer le résultat au client sous forme structurée.
- Les clients sont généralement connectés au serveur à l'aide d'un réseau. Une bonne architecture des applications client/serveur ne doit pas augmenter anormalement la charge du réseau.
- Bon nombre d'applications client/serveur sont construites pour que seul le serveur connaisse l'organisation physique des données. Alors, le rôle du client « se limite» à ordonner l'exécution de méthodes ou des requêtes et à afficher les résultats à l'utilisateur.
- Le modèle client/serveur repose sur le partage des ressources du serveur tel que les données, les procédures stockées, la mémoire ... et le comportement de chacun des clients influencent les performances globales du système.

Le client/serveur peut être implanté de différentes manières suivant les besoins du département informatique. Il peut être en deux, trois ou n tiers. Nous baserons le reste de ce travail uniquement sur les infrastructures deux et trois tiers.

---

[9] Nous examinons ici plus particulièrement les architectures client/serveur dont le serveur partage au moins un accès vers une base de données, ce qui correspond tout de même à une grande majorité des cas.

4.1.2 Client/serveur deux tiers

Sur le client/serveur deux tiers, seule la base de ***D***onnées a été déportée sur le serveur. Le client héberge la totalité de l'application c'est-à-dire l'***I***nterface ***H***omme-***M***achine et le ***T***raitement. Cette architecture porte le nom de client obèse, plus connu sous son équivalent anglais de « fat client ».

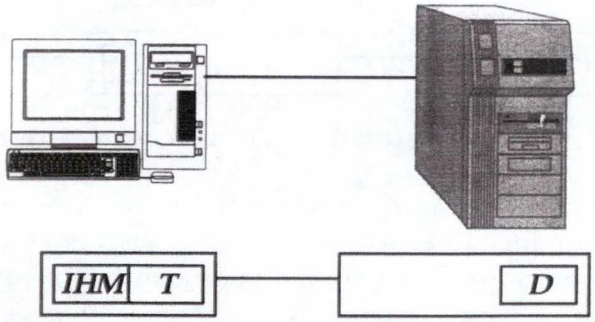


Figure 4.1 – Architecture 2-tiers.

Normalement, les accès aux données dans une architecture deux tiers doivent se faire de haut niveau, c'est-à-dire sans connaissance du schéma physique de la base de données. Cette méthode donne par exemple une plus grande liberté à l'administrateur lors des modifications du schéma ou des index où seules les procédures d'accès côté serveur doivent être mises à jour.

Malgré une mise en œuvre facile, ce modèle a comme principal désavantage (on ne parle pas ici en terme de performance) d'accroître les coûts de déploiement de nouvelles applications et de mise à jour des anciennes puisque ces opérations doivent être répétées sur chacun des postes clients. Néanmoins pour le problème de mise à jour, il existe une solution facilement implémentable en Delphi et dans bon nombre d'autres langages[10] qui consiste à programmer l'application sous forme de processus (thread). L'application va au début de chacune de ses exécutions, démarrer un processus annexe qui va via le réseau, contrôler si une nouvelle version de l'application existe. Si oui, il va tuer l'application qui lui a donné vie, copier la nouvelle version de celle-ci à travers le réseau, l'exécuter et se tuer. S'il n'existe pas de nouvelle version, le processus annexe se termine et la première application continue normalement. L'utilisateur ne remarquera pas ce travail mais à chaque démarrage de l'application utilisera la dernière version.

4.1.3 Client/serveur trois tiers

Dans une architecture 3-tiers, le client héberge l'***I***nterface ***H***omme-***M***achine et une partie des ***T***raitements.

[10] Voir l'équivalence des langages de programmation dans [DEV].

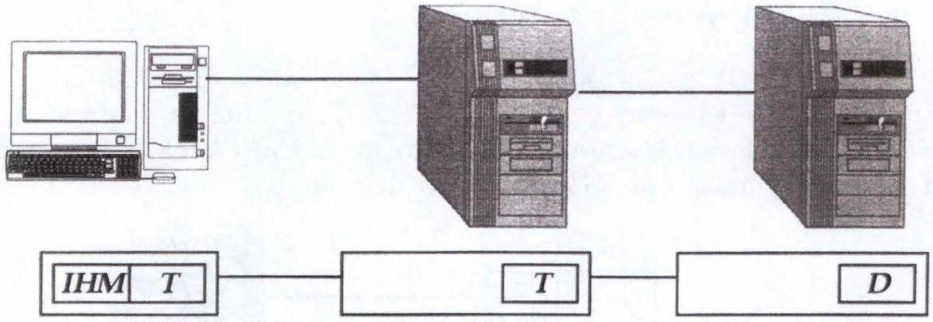


Figure 4.2 – Architecture 3-tiers.

Un serveur d'applications gère l'autre partie des *T*raitements tandis que le serveur de données héberge la ou les bases de *D*onnées et gère leurs accès. Cette architecture porte le nom de « traitement coopératif ». La scission entre les traitements qui se trouvent du côté du serveur et ceux qui sont déportés sur le client dépend du type de traitement. Les traitements proches de l'utilisateur comme par exemple la vérification de la validité d'un format de date sera normalement du côté client. Au contraire, les traitements proches de la base de données seront du côté serveur comme par exemple le calcul de la ristourne faite sur les achats de l'année.

Le transfert de traitements au niveau du serveur permet d'y implémenter des règles d'affaires. La modification de ces règles d'affaires ne devra se faire qu'à cet endroit et non sur toutes les applications des différents clients (*Figure 4.3*). C'est particulièrement utile lorsque cette partie du processus d'affaires est soumise à de nombreux changements comme par exemple dans une banque, le taux d'intérêts.

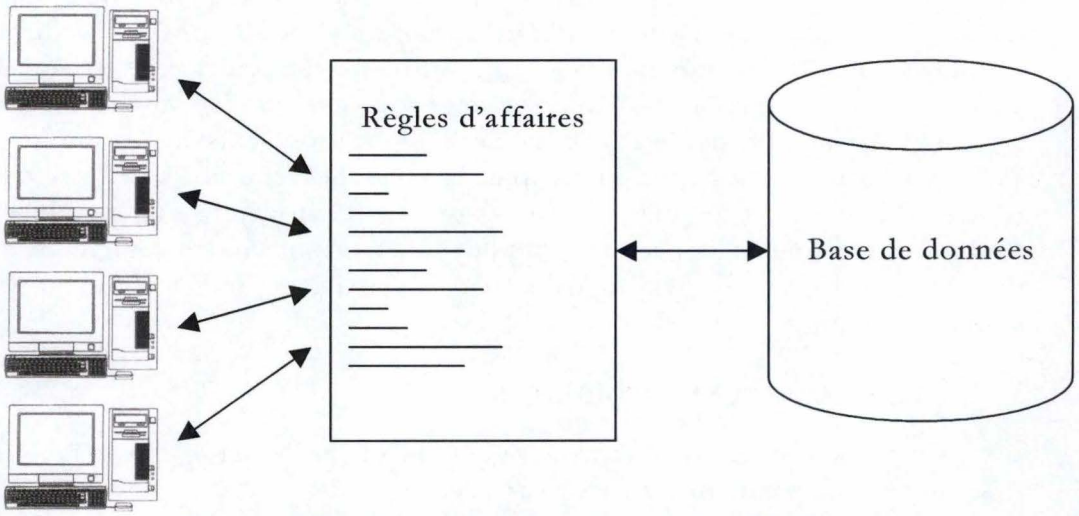


Figure 4.3 – Centralisation des règles d'affaires.

4.1.4 Client/serveur reposant sur la technologie RMI

Dans ce point nous n'allons pas décrire en détail RMI puisque des livres entiers lui sont consacrés mais donner à tout un chacun suffisamment d'informations pour qu'il puisse comprendre les bases de cette technologie utilisée dans la suite de ce travail.

Le but de l'architecture RMI est de permettre la création de systèmes Java utilisant des objets distribués tout en minimisant les différences de programmation entre une application locale et son pendant distribué.

Le principe de base d'une architecture RMI est le concept d'interface. La définition (= l'interface) et l'implémentation d'un service sont deux choses complètement séparées à tel point qu'il est permis qu'elles soient exécutées sur des machines virtuelles différentes. Cette caractéristique convient particulièrement bien pour un usage dans un système distribué où le client se préoccupe essentiellement de la définition des services qui lui sont nécessaires tandis que le serveur se concentre sur la fourniture de ces services.

En RMI donc, les fonctionnalités du service sont définies à l'aide de l'**interface** alors que l'implémentation de celles-ci est réalisée dans une **classe**. En réalité, deux classes implémentent l'interface, l'une du côté serveur qui fournit les fonctionnalités et l'autre du côté client qui joue le rôle de mandataire (proxy) et permet au client d'accéder au service distant comme le montre le schéma ci-dessous.

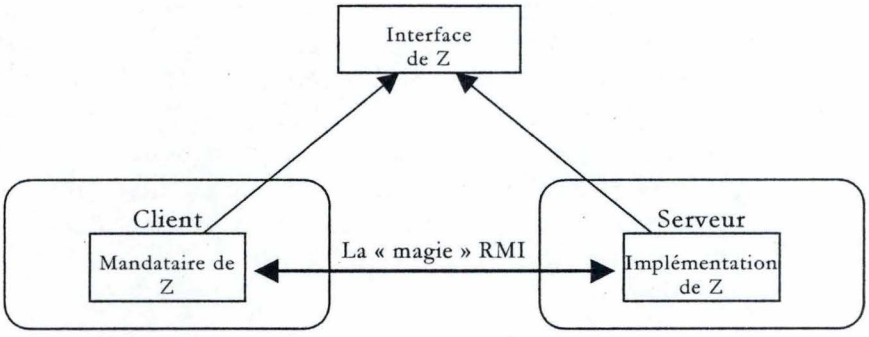


Figure 4.4 – L'idée RMI.

Lorsqu'un client utilise une méthode distante, il invoque la méthode comme si elle était locale et c'est le mandataire qui se charge de la transmettre vers la machine virtuelle distante où elle sera exécutée. Les résultats quant à eux, parcourent le chemin inverse.

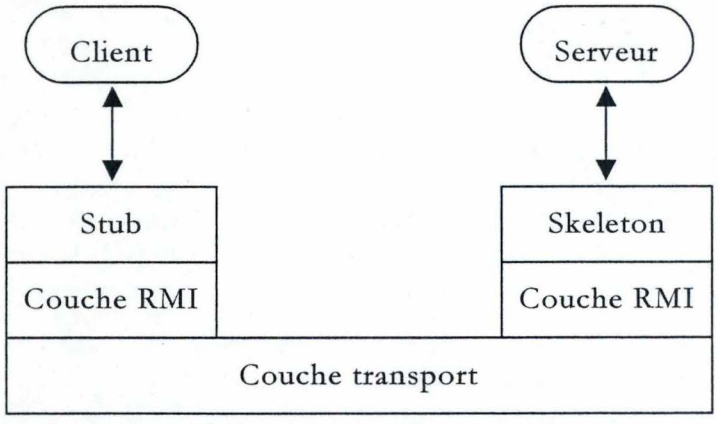


Figure 4.5 – Fonctionnement en couches d'un système RMI

Le stub correspond au mandataire (proxy). Le skeleton communique avec le stub au travers du lien RMI. Le Skeleton reçoit les paramètres pour l'exécution de la méthode, appelle la méthode et récupère le résultat qu'il transmet au stub.

Pour connaître l'emplacement de la méthode distante, RMI utilise un service de nommage situé à une adresse stable et connue de toutes les classes du système distribué. Les serveurs vont y décrire les méthodes distantes qu'ils possèdent et les clients vont l'interroger pour savoir sur quel serveur se trouve une méthode particulière.

Détail amusant et performant, avec Java 2 les connexions ne sont plus uniquement unicast (i.e. d'un point à un autre) mais elles peuvent être multicast (d'un point vers plusieurs points choisis). Ce qui permet à un mandataire de demander l'exécution d'une méthode à plusieurs serveurs simultanément et de ne prendre en compte que la réponse la plus rapide (ce qui améliore les performances).

Les paramètres de type primitif passés à une méthode distante le sont par valeur. Les résultats de ce même type sont également renvoyés par valeur. Lorsque le paramètre est un objet, RMI envoie l'objet lui-même et pas seulement sa référence comme cela se fait habituellement sur une seule machine virtuelle. De la même manière, lorsqu'une méthode distante retourne un objet comme résultat, c'est une copie de l'objet qui est renvoyé au programme appelant. Le passage par valeur et non par référence augmente la charge du réseau ainsi que celle des machines mais est une étape obligatoire si le client veut disposer de l'objet sur sa machine (peut-être que ce transfert coûteux en ressource en épargnera beaucoup juste après). Pour passer un objet d'une machine virtuelle à une autre, RMI utilise un procédé appelé la « sérialisation » qui permet de transformer des objets en un format linéaire qui peuvent être envoyés sur le réseau (ainsi qu'enregistrés sur un disque). Les objets sérialisés peuvent de l'autre côté être dé-sérialisés pour être utilisés de façon classique.

RMI introduit un troisième type de paramètre qui est l'objet distant. Un client peut recevoir la référence à un objet distant en interrogeant le service de nommage ou alors cette référence peut être renvoyée au client comme résultat d'une méthode.

4.2 Développement d'objets dans une architecture distribuée

La construction des objets distribués ne diffère pas énormément de celle des objets d'une application orientée objet classique. Les composants distribués sont de toutes les formes et de toutes les tailles. Ils peuvent être très petits comme par exemple un bouton d'une interface homme-machine, comme ils peuvent implémenter une tâche très complexe, par exemple gérer les comptes d'une banque.

Tout le raisonnement que nous faisons sur les objets distribués est aussi valable pour les business objects. Ceux-ci ne sont en quelque sorte que des objets construits pour automatiser une tâche qui a une signification auprès des individus familiers au domaine d'applications ; même si ces derniers ne sont pas habitués à l'informatique. Notons au passage que l'optique de ce travail est d'essayer d'automatiser au maximum la création d'applications basées sur des business objets dans une architecture client/serveur (pour en diminuer le temps de développement), mais cela ne doit pas être une raison suffisante pour ne pas s'intéresser à l'aspect qualitatif des objets.

4.2.1 Étapes de conception

Le développement d'une application[11] à objets distribués se fait en 4 étapes :

1. Diviser le problème global en modules logiques et identifier les différents blocs tel que celui de présentation, celui d'accès aux données, ...
2. Déterminer le type d'architecture à utiliser (2-tiers, 3 tiers, ...)
3. Diviser les blocs du point 1 en objets et déterminer la place de chacun de ceux-ci dans l'architecture définie au point 2 (i.e. l'objet *x* est à placer du côté client, l'objet *y* est sur le serveur de données ...).
4. Règles de construction des objets :
   - Penser le développement des objets pour en faciliter la réutilisation.
   - Séparer le code stable du code volatile. Toutes les parties d'une application n'ont pas la même stabilité dans le temps. Il faut essayer au maximum de les séparer pour que les fréquents changements appliqués aux uns n'influencent pas les autres.

4.2.2 Influence du design des objets distribués sur l'efficacité

La taille des objets doit influencer les performances des applications mais dans quelle mesure, là est toute la question. Si le client et le serveur doivent s'échanger des objets distribués alors il est clair que plus un objet est gros et plus la charge du réseau est importante, donc les performances générales chutent (moins bonne montée en charge, ...). De l'autre côté, des petits objets rendent plus difficile l'écriture et la gestion des règles d'affaires. Dans le cas où les objets ne seraient pas échangés sur le réseau, le problème existe aussi même si l'aspect charge du réseau a naturellement disparu. Par exemple il n'est pas nécessaire d'utiliser un énorme objet qui reprend toute la comptabilité de l'entreprise pour renseigner l'utilisateur sur le solde d'un client. Ceci aurait pour conséquence de gaspiller de la mémoire, du temps processeur ainsi qu'accroître le nombre d'accès à la base de données nécessaires au garnissage de l'objet. L'emplacement des objets dans l'architecture influence aussi directement les performances, nous en avons déjà parlé.

Le problème est qu'il n'y a pas, à notre connaissance, de théorie complète solutionnant le problème. De plus, le problème de performance ne s'accorde pas bien avec la philosophie objet puisque cette dernière prône la réflexion par rapport au domaine d'applications et non pas par rapport à l'outil informatique sur lequel la solution va être implantée. Ce problème est bien trop complexe pour pouvoir être abordé en détail dans un travail dont ce n'est pas l'objectif principal, d'autant plus que Sun lui-même n'a pas, à l'heure actuelle, de réponse à la question. Néanmoins ceux-ci s'y intéressent puisqu'un projet[12] de mesure de performances d'une architecture à objets distribuées est en cours de

---

[11] Cette méthode est destinée principalement au développement d'applications complexes. Les applications plus simples peuvent facilement être réalisées avec moins de rigueur.

[12] Le projet en question est le projet ECperf dont les dernières informations disponibles datent du 22 juin 2000.

développement. Un tel outil permettra de mesurer l'impact de l'architecture des objets pour une même application et de ces observations peut-être élever une théorie.

4.3 EXEMPLE D'IMPLANTATION D'UNE APPLICATION DANS UNE ARCHITECTURE CLIENT/SERVEUR

Différentes techniques pour la mise en œuvre d'applications business objects dans un environnement client/serveur sont disponibles telles que DCom, Corba, RMI et EJB en ne citant que les plus connues. Le langage de développement choisi étant le Java, deux techniques étaient le mieux à même de réaliser la tâche : RMI et EJB. Finalement, c'est la technologie RMI qui a été choisie car même si la ressemblance entre business objects et beans est forte, cela n'empêche pas RMI d'être plus simple dans son approche, plus facile à comprendre. Le passage de RMI vers d'autres technologies telles que Corba par exemple est assez facile à réaliser. L'utilisation des EJB aurait nécessité l'introduction de concepts comme le conteneur et la gestion du cycle de vie des objets, les différents types de beans (orienté session, sans session ...) ce qui complique un peu le système.

Les étapes de développement d'une architecture RMI sont au nombre de 6 :

1. Écrire et compiler l'interface.
2. Écrire et compiler le code d'implémentation des classes.
3. Générer les classes Stub et Skeleton à partir des classes d'implémentation.
4. Écrire le programme serveur qui va faire exécuter les services définis par l'interface.
5. Écrire le client qui va utiliser les services définis par l'interface.
6. Installer et exécuter le système.

L'exemple que nous allons développer est un des plus simples pour faciliter la compréhension mais a l'avantage d'être complet. L'application réalisée permet à un client d'interroger le serveur où est logée la base de données pour connaître le nom et/ou le prénom d'un citoyen connu par le client grâce à son identifiant (son numéro de carte d'identité ou de registre national par exemple). L'application est destinée à être exécutée sur une architecture trois tiers et ci-dessous sont décrites la partie client et la partie serveur de traitement, ce dernier étant lié à une base de données dont l'adresse est connue (l'URL). Lors de cette implémentation, c'est la clarté de l'application qui a été recherchée et non la performance.

4.3.1 Écrire et compiler l'interface

La première étape consiste à écrire et compiler le code Java correspondant à l'interface. L'interface définit tous les services distants offerts par le serveur.

```
public interface Recherche
  extends java.rmi.Remote
```

```
{
   public String GetNom(String id)
      throws java.rmi.RemoteException, java.sql.SQLException;

   public String GetPrenom(String id)
      throws java.rmi.RemoteException, java.sql.SQLException;
}
```

Il faut enregistrer le fichier sous le nom 'Recherche.java'. Pour compiler le fichier, il faut taper : javac Recherche.java.

4.3.2 Écrire et compiler le code d'implémentation des classes

La deuxième étape est l'écriture du code implémentant les services distants décrits par l'interface.

```
public class RechercheImpl
  extends java.rmi.server.UnicastRemoteObject
  implements Recherche
  {
     //L'implémentation doit avoir un constructeur explicite
     //pour pouvoir déclarer l'exception RemoteException.

     public RechercheImpl() throws java.rmi.RemoteException,
                                   java.sql.SQLException
     {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection c = DriverManager.getConnection(dbUrl, user, password);
     }

     public String GetNom(String id) throws java.rmi.RemoteException,
                                            java.sql.SQLException
     {
        Statement s = c.createStatement();
        ResultSet r =s.executeQuery("SELECT * " +
                                    "FROM citoyen " +
                                    "WHERE ID = " + id);

        if (r.next()) return r.getString("Nom");
        else {return "";}
     }

     public String GetPrenom(String id) throws java.rmi.RemoteException,
                                               java.sql.SQLException
     {
        Statement s = c.createStatement();
        ResultSet r =s.executeQuery("SELECT * " +
                                    "FROM citoyen " +
                                    "WHERE ID = " + id)
        if (r.next()) return r.getString("Prenom");
        else {return "";}
     }
  }
```

Pour compiler le fichier, il faut taper : javac RechercheImpl.java si le fichier a été enregistré sous le nom RechercheImpl.java.

4.3.3 Générer les classes Stub et Skeleton à partir des classes d'implémentation

Pour générer le stub et le skeleton il suffit d'utiliser le compilateur RMI sur la classe d'implémentation des services : rmic RechercheImpl.
Deux fichiers devraient être créés Recherche_Stub.class et Recherche_Skel.class.

4.3.4 Écrire le programme serveur qui va faire exécuter les services définis par l'interface

Les services distants doivent être hébergés sur un serveur.

```
import java.rmi.Naming;

public class RechercheServeur
  {
     public RechercheServeur()
     {
        try {
               Recherche c = new RechercheImpl();
               // enregistrement sur le service de nommage
               naming.rebind("rmi://localhost:1099/RechercheService",c);
            }
        catch (Exception e) {
                                 System.out.println("Problème:"+e);
                               }
     }

     public static void main(String args[])
     {
        new RechercheServeur();
     }
  }
```

4.3.5 Écrire le client qui va utiliser les services définis par l'interface

```
import.java.rmi.Naming;
import.java.rmi.RemoteException;
import.java.net.MalformedURLException;

public class RechercheClient
  {
     public static void main(String[] args)
     {
      try{
            Recherche r = (Recherche)
                Naming.lookup("rmi://remotehost/RechercheService");
            System.out.println(r.GetNom("123"));
            System.out.println(r.GetPrenom("123"));
            System.out.println(r.GetNom("259"));
         }
      catch (MalformedURLException murle)
      {
         System.out.println("MalformedURLException");
         System.out.println(murle);
      }
      catch (RemoteException re)
```

```
        {
            System.out.println("RemoteException");
            System.out.println(re);
        }
       }
    }
```

4.3.6 Installer et exécuter le système

Le système est maintenant prêt à être installé et exécuté. Pour commencer, il faut lancer le service de nommage depuis le répertoire qui contient les classes nouvellement écrites : rmiregistry.

Ensuite vient l'exécution du serveur qui héberge les services distants : java RechercheServer.

Pour terminer, si les deux étapes précédentes se sont bien passées et si une base de données contenant la table 'citoyen' utilisée existe à l'adresse donnée, l'exécution du client fournit à l'écran les résultats attendus. Le client s'exécute en tapant : java RechercheClient.

RMI utilise la pile TCP/IP pour que les différents processus communiquent entre eux. Il est donc indispensable que ce service soit présent sur le ou les ordinateur(s) où le système est exécuté.

Chapitre 5

Premier support outil - Transformation d'un prototype d'écran en une entité

Objectif :

Ce chapitre a pour but de définir les règles de construction de maquettes d'écran ainsi que d'expliquer le fonctionnement du programme qui permet de générer une entité à partir d'une de ces maquettes.

5.1 CONTEXTE

L'extraction de sous-schémas conceptuels à partir de prototypes d'écran est une opération pénible, répétitive et où une seule erreur d'inattention entraîne un blocage de tout le processus en aval. De plus, la moindre application possède souvent plus d'une dizaine d'écrans plus ou moins complexes, ce nombre accroît le charge de travail ainsi que la probabilité d'avoir des erreurs. Heureusement, il n'est pas trop difficile de dresser la liste des règles d'extractions et de les implémenter sous forme d'un programme pour résoudre le problème automatiquement.

5.2 AUTOMATISATION : LE PROGRAMME *DFM2ET*

Le programme DFM2ET transforme de manière automatique un prototype de fenêtre créé sous Delphi[13] en un type d'entité correspondant au sous-schéma conceptuel de ce prototype d'écran. Delphi est utilisé ici dans sa version quatre, les autres versions n'ont pas été testées mais devraient fonctionner. Les règles à suivre lors de la conception des prototypes d'écran et qui garantissent le bon déroulement du processus automatique de transformation sont décrites au point 5.3 de ce même chapitre (page 48).

5.2.1 Explication de l'exécution

Après avoir créé son ou ses prototype(s) de fenêtre de manière standard, l'utilisateur doit les enregistrer sous forme de texte. Cependant Delphi ne propose pas directement de sauver des fenêtres sous forme de texte, par contre le second bouton de la souris enfoncé au niveau de la fenêtre nouvellement créée déploie un menu contextuel où un des choix permet de la voir sous forme de texte. Il faut copier-coller ce code vers un fichier vide et puis sauvegarder celui-ci (avec comme extension « .txt »). Une fois en possession d'un tel fichier, il suffit de lancer le programme « dfm2lun.oxo » qui dès le début de son exécution demande de sélectionner un fichier du même type que celui que nous venons de créer (voir *Figure 5.1*, ci-dessous). Ce fichier va être analysé pour produire l'entité correspondante qui sera stockée dans le schéma DB-MAIN courant. Mis à part le cas où il n'y a pas de schéma ouvert (le programme signalerait alors l'absence de schéma), il produit une entité où les attributs sont les boîtes de saisie de la fenêtre donnée en entrée dont les noms sont issus des labels qui leur sont attachés.

---

[13] Delphi est un produit Borland.

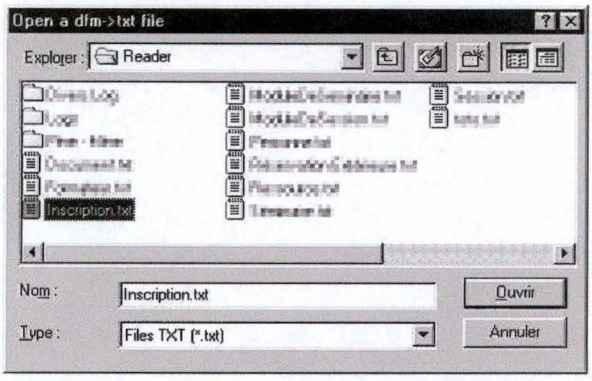
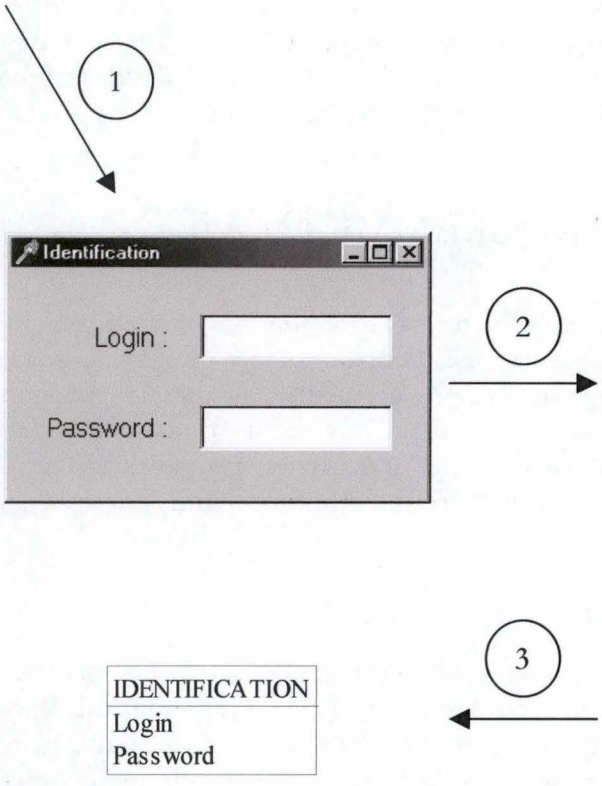


Figure 5.1 – L'unique interaction entre DFM2ET et l'utilisateur.

Les grandes étapes du processus de génération du sous-schéma conceptuel appliquées à un exemple très simple sont :




```
01 :   object Form1: TForm1
02 :     Left = 251
03 :     Top = 171
04 :     Width = 269
05 :     Height = 168
06 :     Caption = 'Identification'
07 :     Color = clBtnFace
08 :     Font.Charset = DEFAULT_CHARSET
09 :     Font.Color = clWindowText
10 :     Font.Height = -16
11 :     Font.Name = 'MS Sans Serif'
12 :     Font.Style = []
13 :     OldCreateOrder = False
14 :     PixelsPerInch = 96
15 :     TextHeight = 20
16 :     object Label1: TLabel
17 :       Left = 52
18 :       Top = 32
19 :       Width = 47
20 :       Height = 20
21 :       Caption = 'Login :'
22 :     end
23 :     object Label2: TLabel
24 :       Left = 22
25 :       Top = 88
26 :       Width = 77
27 :       Height = 20
28 :       Caption = 'Password :'
29 :     end
30 :     object Edit1: TEdit
31 :       Left = 120
32 :       Top = 28
33 :       Width = 121
34 :       Height = 28
35 :       TabOrder = 0
36 :     end
37 :     object Edit2: TEdit
38 :       Left = 120
39 :       Top = 84
40 :       Width = 121
41 :       Height = 28
42 :       TabOrder = 1
43 :     end
44 :   end
```

Figure 5.2 – Les trois étapes de la construction d'un sous-schéma conceptuel.

A la *Figure 5.2*, la première étape correspond à la création de la maquette de fenêtre, l'étape deux représente la sauvegarde sous format texte de la description de la fenêtre et la dernière étape symbolise la transformation automatique réalisée par le programme DFM2ET.

5.2.2 Explication algorithmique[14] du programme DFM2ET

Le programme DFM2ET a pour objectif d'automatiser la troisième étape de la *Figure 5.2*. Il se décompose en cinq grosses fonctions enchaînées l'une à la suite de l'autre, c'est-à-dire que la $n^{ème}$ ($n \geq 2$) fonction prend en entrée le résultat de la $(n-1)^{ème}$. Les fonctions sont dans l'ordre d'exécution :

1) Dfm2List();
2) Align(liste);
3) CrossTab1(liste);
4) Cardinalities(liste);
5) List2ET(liste);

a) Dfm2List()

La première fonction de l'application est Dfm2List. C'est elle qui va lire le fichier texte contenant le code de la fenêtre prototype qu'il est demandé de transformer en entité. Le fichier ne va être lu qu'une seule fois, chacune de ses lignes va être analysée, l'information pertinente en sera extraite et insérée dans un tableau. Cette technique apporte des gains de performance significatifs puisqu'elle minimise au maximum les accès disques qui sont des plus pénalisants en terme de temps.

	1	2	3	4	...
Label	IDENTIFICATION	LOGIN	-	PASSWORD	
Niveau	1	2	2	2	
Left	251	52	120	22	
Top	171	32	28	88	
Width	269	47	121	77	
Height	168	20	28	20	
Type	TForm1	TLabel	TEdit	Tlabel	
Proche	/	3	/	5	
ColCount	/	/	1	/	
NbLiens	/	/	1	/	
Min_rep	/	/	1	/	
Max_Rep	/	/	1	/	

Figure 5.3 – Le tableau temporaire utilisé tout au long du programme DFM2ET.

Dans un tableau comme celui de la *Figure 5.3*, les composants sont stockés en colonnes. En lignes se trouvent les différentes propriétés de ces composants nécessaires et suffisantes à la suite de l'exécution du programme.

Le « Label » stocke l'information du champ 'caption' du composant de la fenêtre si celui-ci est pertinent pour le programme. En effet, certains

[14] Le code complet du programme ne se trouve pas dans ce point, il a été reporté à l'annexe 1.

composants (un bouton par exemple) possèdent également un champ 'caption' mais les boutons sont liés à des méthodes et pour cela ne nous intéressent pas ici. Dans le cas où le composant ne possède pas de 'caption' la ligne est mise à "-".

Le « Niveau » permet de connaître pour chaque composant sa place dans la hiérarchie. Les niveaux de hiérarchie sont spécifiés dans la phase de conception des prototypes d'écran par des boîtes de regroupement. La fenêtre principale est le seul élément de niveau 1, les autres sont de niveau 2 ou plus.

Les champs « left, top, width et height » reprennent les coordonnées des objets qui composent le prototype d'écran. Elles seront utilisées pour calculer quel champ texte est lié à/ou explique quel champ d'édition. Par exemple, à la *Figure 5.4*, elles nous permettent de définir que la boîte d'édition de cette figure doit être représentée dans l'entité par un attribut dont le nom est « Prix ».

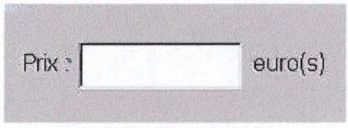


Figure 5.4 – Quel est le champ texte lié à cette boîte d'édition ?

Le champ « Type » enregistre le type de l'objet inséré dans cette colonne. Ce peut être un TForm, TLabel, TEdit, TGroupBox ...

Le champ « Proche » est utilisé pour les labels. Comme on l'a vu à la *Figure 5.4*, les coordonnées permettent de calculer que tel label doit être relié à tel composant éditable. Une fois ce lien calculé, il est stocké dans la cellule « Proche » du composant de type TLabel. Ce lien est enregistré sous la forme de l'indice de la colonne du tableau où se trouve l'objet lié.

La valeur du champ « ColCount » est mise à 1 pour la plupart des composants. Elle indique le nombre de colonnes que possède ce composant. Par exemple un TStringGrid peut être composé de plusieurs colonnes et pour un objet de ce type, il doit y avoir autant de zones de texte expliquant une colonne que de colonnes comme le montre la *Figure 5.5* ci-dessous.

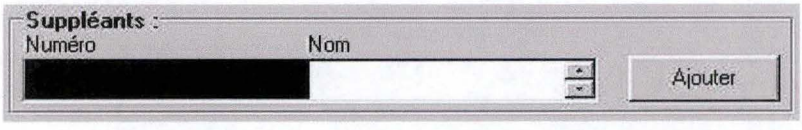


Figure 5.5 – Exemple où un composant possède plusieurs colonnes.

Le champ « NbLiens » enregistre le nombre de zones de texte qui ont un lien avec l'objet courant. Pour un composant éditable une fois le nombre de liens (= NbLiens) égal au nombre de colonnes du composant (ColCount), il est considéré comme complètement expliqué. Il faut assigner à un composant éditable un certain nombre de zones de texte qui expliquent ce que ce composant signifie. En d'autres termes, il doit y avoir autant de zones de texte attachées à un composant éditable que ce composant a de colonnes à expliquer.

Les deux dernières lignes du tableau reprennent les cardinalités minimum et maximum du composant, par exemple un composant de type TStringGrid aura une cardinalité maximum égale à N.

b) Align(liste)

Après avoir extrait du fichier correspondant au prototype d'écran les informations qui nous intéressent, nous allons appliquer à ces informations la fonction Align. Le but de la fonction Align (qui prend le tableau résultat de Dfm2List() comme paramètre d'entrée) est de modifier les coordonnées relatives des composants en coordonnées absolues. De fait, la construction des maquettes d'écran donne aux composants des coordonnées relatives par rapport au père de chaque élément et on les voudrait sous forme absolue (i.e. par rapport à la fenêtre principale). En effet, l'analyse permettant de savoir si telle zone de texte correspond à telle zone éditable va travailler sur les coordonnées. Mais, il se pourrait que les coordonnées des coins supérieurs gauches de deux éléments soient toutes deux (x,y) alors que visuellement ces deux composants sont situés à des endroits bien distincts. Le premier se situe par exemple à un décalage de (x,y) points par rapport au coin supérieur gauche de la fenêtre principale (son père), alors que le deuxième est lui aussi à (x,y) points du coin supérieur gauche de son père, ce dernier n'étant pas la fenêtre principale mais, par exemple, une boîte de regroupement.

Align va parcourir le tableau et pour tous les éléments dont le niveau est au moins égal à 3 (c'est-à-dire que leur père n'est pas la fenêtre principale), elle va ajouter aux coordonnées Left et Top du composant les valeurs des coordonnées Left et Top du père de celui-ci.

c) CrossTab1(liste)

Une fois que les coordonnées sont absolues (grâce à la fonction Align), la fonction CrossTab1 va calculer quelle est la zone de texte à assigner à chaque composant (comme sur la *Figure 5.4*). Pour qu'il y parvienne, il est demandé à l'utilisateur de suivre les règles de construction de prototype d'écran (décrites au point 5.3 de ce même chapitre) et si tel est le cas, le rectangle délimitant le zone de texte sera superposé au composant avec lequel elle doit être liée comme sur la *Figure 5.6*.

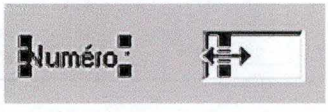


Figure 5.6 – Conception permettant à CrossTab1 de déterminer les liaisons entre composants.

CrossTab1 va pour tous les labels (c'est-à-dire les zones de texte) rechercher si un de ses points extérieurs est inclus dans la zone délimitée par un composant éditable ou si un des coins du composant éditable est inclus dans la zone de texte. Si oui, le lien entre ces composants (un label et un composant éditable) est indiqué dans la cellule du tableau située au croisement de la ligne « Proche » et de la colonne du label. Cette méthode n'est peut-être pas la plus « intelligente » mais elle a le double avantage d'être efficace (elle ne prend pas beaucoup de temps lors du design du prototype d'écran) et fiable (si l'utilisateur construit son écran avec attention, le taux de réussite dans la découverte correcte des liens est de 100%). Si la méthode de recouvrement n'avait pas été utilisée mais à la place de celle-ci une méthode de recherche dans les environs, le programme aurait été beaucoup plus compliqué car il fallait y introduire des règles de recherche. Par exemple d'abord rechercher si le composant ne se trouve pas à

droite (car comme dans la *Figure 5.6*, le composant est souvent à droite du label qui le décrit) puis vers le bas ... De plus comme le résultat n'était pas sûr à 100%, il fallait créer un programme, si possible muni d'une interface graphique agréable, qui permette à l'utilisateur de corriger les erreurs. Le problème (si c'est un problème) vient du fait que l'être humain est doté de facultés de déduction que la machine ne possède pas. En effet, il est facile pour tout un chacun de saisir le numéro d'un document (*voir Figure 5.7*, ci-dessous) dans la case blanche située à la droite de l'intitulé. Dans le groupe « Est composé de » il est tout aussi facile de le faire puisqu'il ne faut pas chercher la zone de saisie à droite mais en dessous de lui. La règle générale est donc claire, regarder à gauche et s'il n'y a rien tout près, chercher en dessous. Mais comme toute règle a généralement sa ou ses exceptions, le problème devient plus ardu quand on analyse le libellé « euros » (toujours sur la même figure). A droite, il n'y a rien donc on se tourne vers le bas. Seulement en bas il y a deux candidats à l'association « label – boîte de saisie ». Comme « taille » va s'associer avec la boîte de gauche la plus proche, il ne restera plus que la TComboBox « disquette(s) » à proximité du libellé « euros » et ils vont tout naturellement s'associer alors que ça n'a aucun sens.

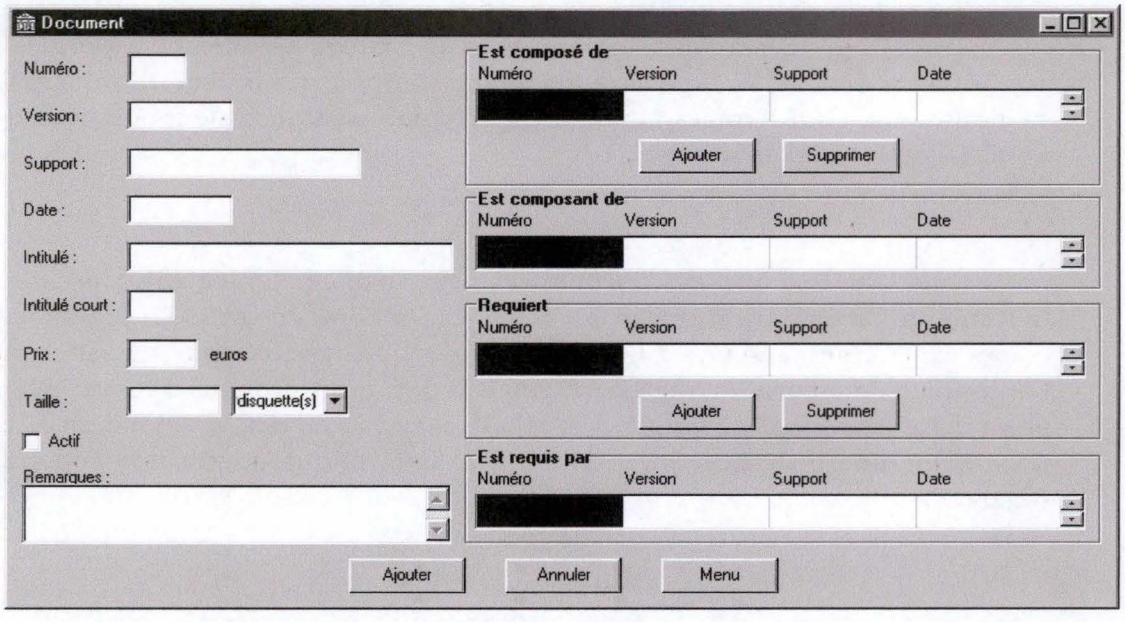


Figure 5.7 – Un cas concret de problème de recherches de liaisons intelligentes.

d) Cardinalities(liste)

La quatrième étape de programme DFM2ET est la fonction qui gère les cardinalités. C'est dans cette fonction que se trouvent toutes les règles qui vont modifier les cardinalités de certains types attributs de l'entité que l'on va créer. Tous ces types d'attributs ont comme point commun d'être issus de composants de type TStringGrid. Jusqu'à présent, toutes les cardinalités des attributs insérés dans les entités étaient mises par défaut à [1-1]. Si on reprend l'exemple de la *Figure 5.6*, ci-dessus, la boîte de regroupement 'Est composé de' inclut un TStringGrid à quatre colonnes. On considère que les cardinalités d'un TStringGrid, puisque représentant un tableau en deux dimensions, sont du type [0-N]. Dans ce cas particulier la cardinalité minimum 0 s'impose de soi, puisqu'il faut bien que certains objets soient atomiques (tôt ou tard sinon on est en présence d'un problème de type « l'œuf ou la poule ? »). Cette cardinalité [0-N]

doit être de mise pour les futurs attributs qui sont liés avec le TStringGrid, c'est-à-dire 'Numéro', 'Version', 'Support', 'Date'. Ce qui nous donne comme type d'entité celui visible à la *Figure 5.8* située ci-dessous.

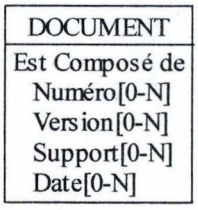


Figure 5.8 – Cardinalités [0-N] non reportées.

Généralement ce genre de cardinalité doit être reporté au niveau supérieur comme il l'est sur la *Figure 5.9*, cela dépend de la sémantique du schéma. Dans notre exemple, l'utilisation d'une entité comme celle de la *Figure 5.8* conduira à avoir des documents composés d'autres documents dont on ne connaît aucun quadruplet (numéro, version, support, date). Par exemple, on disposera de 3 numéros identifiants, de 2 numéros de version, d'aucun support mais de 5 dates de parution. Même si ces attributs étaient en nombre égaux, il serait impossible de recréer le quadruplet auquel ils appartiennent.

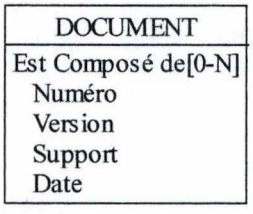


Figure 5.9 – Cardinalité [0-N] mise à bon niveau.

Cette modification est réalisée par la fonction Cardinalities. Le principe de celle-ci est de regarder si tous les attributs les plus bas dans la hiérarchie sont liés à un même composant de type TStringGrid. Si oui, les cardinalités de ces attributs sont mises à [1-1] et la cardinalité [0-N] est reportée au niveau du père de ces attributs. Par les attributs les plus bas dans la hiérarchie, on entend ceux correspondant aux feuilles d'un arbre représentant l'entité.

e) List2ET(liste)

La dernière fonction du programme est en fait une procédure. Son but est de transformer l'ensemble des composants du prototype d'écran susceptible de contenir de l'information en un type d'entité muni de ses attributs.

Le premier composant stocké dans le tableau, s'il existe, correspond à la fenêtre principale qui va être représentée dans le sous-schéma conceptuel sous forme d'une entité. Les autres composants vont être analysés et ne seront insérés comme attribut de l'entité que s'ils possèdent un champ 'Caption' c'est-à-dire qu'ils sont du type TLabel, TGroupBox, TCheckBox, TRadioButton et s'ils sont liés à un autre élément. Le gros de l'algorithme est la gestion de la hiérarchie et des cardinalités des éléments du futur type d'entité. Par exemple pour toute insertion d'un nouvel attribut, il faut savoir s'il possède des fils qui ne sont pas encore insérés car DB-MAIN fait la distinction entre des attributs simples et des attributs composés. Insérer un attribut sous un attribut simple existant ne

transforme pas ce dernier en attribut composé mais provoque une erreur. Si le type du composant à insérer est TCheckBox ou TRadioButton alors l'attribut qui le représente est de type booléen. Pour les champs de type TMemo, les cardinalités choisies sont [0-1] car un champ mémo n'est généralement pas obligatoire. Évidemment toutes ces propriétés données aux objets de manière arbitraire peuvent être facilement modifiées par l'utilisateur lorsque ce dernier va à partir des sous-schémas conceptuels créer un schéma conceptuel global.

5.2.3 Code

Le code du programme DFM2ET est trop volumineux pour être inséré ici, il a été reporté en Annexe 1.

5.3 RÈGLES DE CONCEPTION D'UN PROTOTYPE D'ÉCRAN

Les règles de conception de prototypes d'écran sous Delphi sont assez simples puisqu'elles sont au nombre de trois mais encore faut-il les appliquer. Si elles ne sont pas respectées, il y a de grandes chances pour que le programme DFM2ET ne fournisse pas le résultat escompté.

Les trois règles sont :

A. Ensemble limité de types d'objets

B. Cohérence de la hiérarchie

C. Superposition

5.3.1 Ensemble limité de types d'objets

Tous les composants fournis par Delphi pour créer un prototype de fenêtre ne sont pas pris en compte par DFM2ET et seuls les composants les plus utilisés sont acceptés. Les autres devront être soit ajoutés au programme DFM2ET ou alors programmés à la main dans l'application que l'utilisateur désire créer. La liste des composants acceptés est la suivante :

Nom des composants	Représentation graphique	Nom des composants	Représentation graphique
TLabel	Label	TEdit	Edit
TButton	Button	TBitButton	BitButton
TCheckBox	CheckBox	TComboBox	ComboBox
TMemo	Memo	TGroupBox	GroupBox
TStringGrid			

Figure 5.10 – Liste des composants acceptés par DFM2ET.

5.3.2 Cohérence de la hiérarchie

Pour introduire la notion de groupement et de hiérarchie entre les éléments d'une fenêtre et pour qu'elle se retrouve dans la future entité, il faut utiliser des boîtes de regroupement (TGroupBox). Chaque objet d'une fenêtre se situe à un niveau dans la hiérarchie. La fenêtre principale est au niveau 1. Tous les objets qui lui sont attachés directement sont au niveau 2 (y compris les boîtes de regroupement). Tous les éléments appartenant à ces boîtes de regroupement sont au niveau 3 et ainsi de suite puisqu'un TGroupBox peut en contenir plusieurs.

Il faut être particulièrement attentif lors de la conception car il est fréquent qu'un design qui nous paraît correct comme sur la *Figure 5.11*, ci-dessous soit fatal au programme DFM2ET. Dans cet exemple, ce dernier va introduire dans l'attribut composé 'Inscrit' les attributs 'Numéro', 'Nom' puis de nouveau un attribut 'Numéro' alors qu'il en existe déjà un ce qui provoque une erreur et la terminaison de l'exécution du programme.

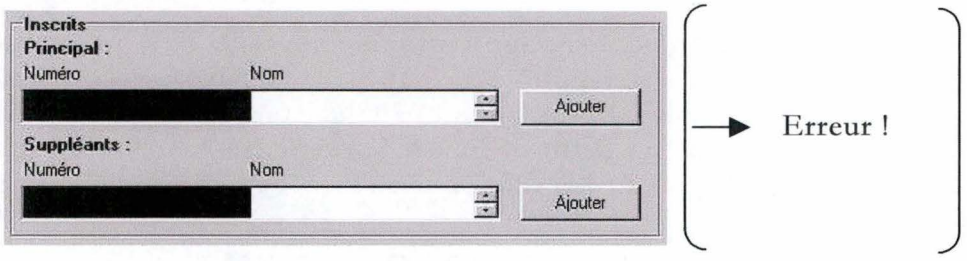


Figure 5.11 – Mauvais respect des règles de hiérarchie.

Pourtant, l'emploi correct de la hiérarchie (qui ne pose pas de problème) est tout aussi facile et esthétique comme le montre la *Figure 5.12*.

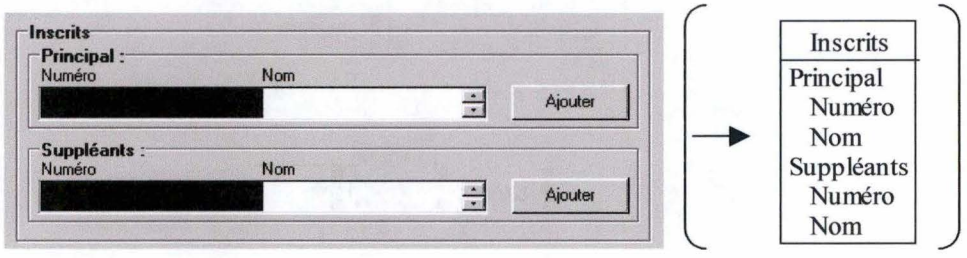


Figure 5.12 – Règles de hiérarchie respectées.

Lorsqu'une boîte de regroupement ne contient qu'un TStringGrid, le programme prend par défaut la légende de cette boîte comme nom de l'attribut qui représentera ce TStringGrid dans l'entité.

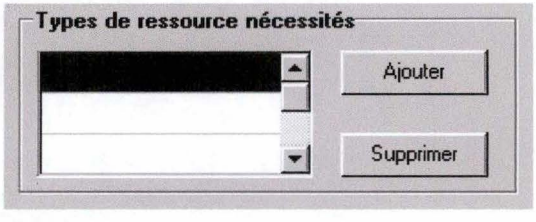


Figure 5.13 – Boîte de regroupement composée uniquement d'un TStringGrid.

5.4 SUPERPOSITION

Le problème de la superposition a déjà été expliqué en détail au point 5.2.2 - CrossTab1(liste) de ce chapitre. Nous ne reviendrons pas dessus et nous nous limiterons uniquement sur la solution.

Lorsqu'un composant de type TLabel est placé sur un prototype d'écran, Delphi le dimensionne automatiquement pour que la surface délimitée par le cadre qui l'entoure soit minimisée par rapport à la surface qu'il utilise réellement.

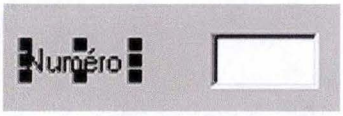


Figure 5.14 – Minimisation automatique de la surface du composant TLabel.

Or, il est essentiel pour le programme DFM2ET qu'un des coins de ce cadre soit superposé par rapport au composant éditable qu'on désire lui assigner ou alors qu'un des coins du composant éditable soit superposé au cadre de la zone de texte. Pour ce faire, il suffit de mettre la propriété AutoSize du TLabel à False (*Figure 5.15*) et d'agrandir le cadre pour qu'il recouvre le composant à lier (*Figure 5.16*).

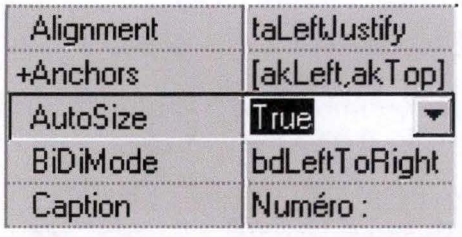


Figure 5.15 – AutoSize à False.

Figure 5.16 – Superposition de cadres.

Chapitre 6

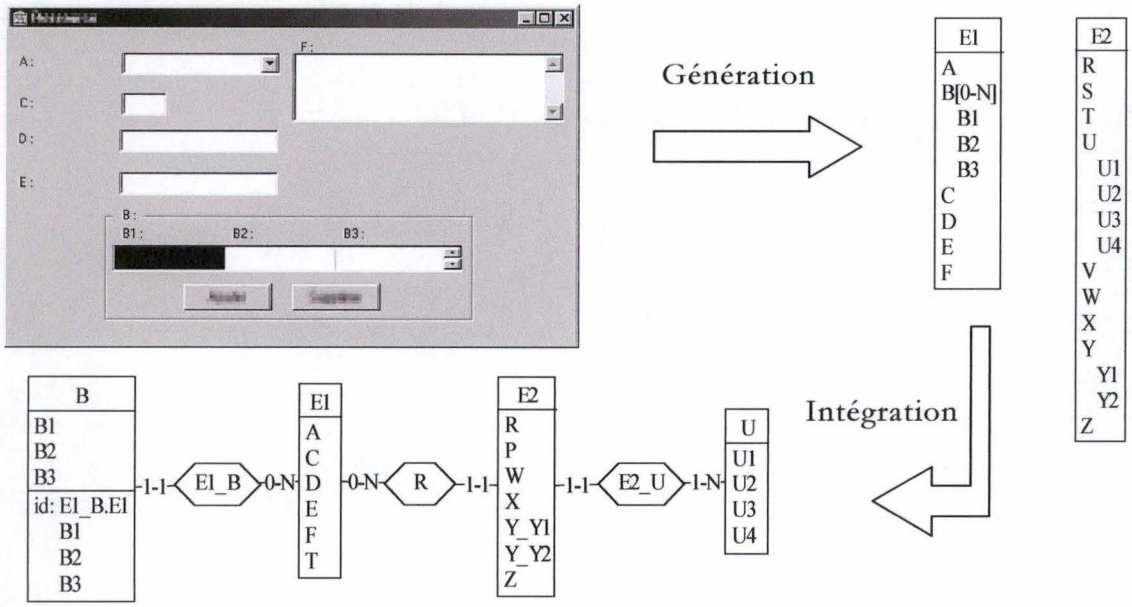
Deuxième support outil - Un écran, une vue

Objectif :

L'intégration des sous-schémas, étape suivant leur génération, modifie considérablement ces sous-schémas à tel point qu'il est impossible de savoir comment a évolué un objet du schéma. Ce chapitre explique comment un programme permet de résoudre ce problème.

6.1 LA PROBLÉMATIQUE

L'étape qui suit la génération de sous-schémas conceptuels, dans le cheminement de la conception d'une architecture business object comme la nôtre (point 3.2), est l'intégration de ces sous-schémas en un schéma conceptuel global. Cette étape d'intégration est interne à DB-MAIN qui permet de la réaliser avec beaucoup de facilité. Mais comme le montre la *Figure 6.1*, s'il est facile de retrouver où est stockée l'information contenue dans les prototypes de fenêtre après la génération des sous-schémas, il est beaucoup moins aisé, voire même impossible de réaliser cette correspondance lorsque ces sous-schémas ont été intégrés dans un schéma conceptuel global.



- L'attribut multivalué B a été transformé en un nouveau type d'entité.
- L'attribut composé Y a été désagrégé.
- L'attribut composé U a été transformé en un nouveau type d'entité, par exemple pour mieux correspondre à la réalité.
- L'attribut T a été transféré de l'entité E2 à l'entité E1.
- Les attributs A et S qui représentaient le même objet de la réalité ont été fusionnés et seul l'attribut A persiste (correspondant aux deux) dans l'entité E1.

Figure 6.1 – Problèmes d'intégration de sous-schémas conceptuels.

6.2 UNE SOLUTION

Le programme LogAnalyser a été conçu pour combler le problème de suivi des attributs pendant la phase d'intégration. Ce programme travaille en deux phases. Tout d'abord il va lire le fichier journal des évènements de DB-MAIN (pour cela, il faut absolument que ce journal ait été activé depuis le début de la génération des sous-schémas jusqu'à la fin de l'intégration en un schéma global) et générer un tableau de même format que celui à la *Figure 6.2* (ci-dessous), dans lequel vont être enregistrés les changements apportés aux objets. Le programme va ensuite lire le fichier journal complémentaire car il se peut qu'une modification apportée à un objet ne se trouve pas dans le journal principal.

	1	2	3
OID New	35	39	41
Nom New	IDENTIFICATION	Login	Password
OWN New	26	35	35
OID Old	35	39	41
Nom Old	Identification	Login	Password
OWN Old	26	35	35
Status	ON	ON	ON

Où :
OID = Identifiant de l'objet dans le repository
OWN = Identifiant du père de l'objet dans le repository
Status = ON ou OFF suivant que l'objet est encore utilisé ou pas

Figure 6.2 – Tableau de suivi des objets de l'exemple de la *Figure 5.2.*

6.2.1 Le tableau

Le tableau de la *Figure 6.2* est destiné à enregistrer l'information de tous les objets du repository[15], qui nous est essentielle c'est-à-dire l'identifiant de l'objet (OID), son nom ainsi que l'identifiant de son père (OWN). Un autre champ est utilisé pour enregistrer le statut d'un objet qui indique si celui-ci est utilisé (ON) ou non (OFF). Chaque objet est stocké dans une colonne du tableau (l'implémentation de ce tableau repose sur des listes chaînées). On désire savoir pour chaque objet comment il a évolué au cours des différents changements apportés au schéma. Pour ce faire, on enregistre dans les champs du tableau (OID Old, Nom Old et OWN Old) l'information se rapportant au moment de sa création et dans OID New, Nom New et OWN New sont enregistrées les dernières modifications de ces propriétés. Grâce à l'ensemble de cette information, il nous est possible de savoir exactement ce qu'est devenu tout objet du schéma.

[15] Sans trop entrer dans les détails, le repository est la base de données de DB-MAIN où sont stockés tous les composants d'un schéma (appelés objets).

6.2.2 Lecture du fichier journal principal

La première étape du programme LogAnalyser est la lecture du fichier journal de DB-MAIN. Son format est simple comme on peut le voir à la *Figure 6.3*. Les informations utilisées par LogAnalyser sont en gras.

```
*POT "##1##new_ent"
*CRE ENT
%BEG
      %OID 37
      %NAM "ENTITY_1"
      %POX 111653
      %POY 52651
      %OWN 1 "SCHEMA"/"1" 26
%END
*POT "##1##getatt"
*MOD ENT
%BEG
      *OLD ENT
      %BEG
            %OID 35
            %NAM "ENTITY"
            %OWN 1 "SCHEMA"/"1" 26
      %END
      %OID 35
      %NAM "E1"
      %OWN 1 "SCHEMA"/"1" 26
%END
```

Figure 6.3 – Extrait du journal d'enregistrement de DB-MAIN.

Le fichier journal est construit de telle façon que sur chaque ligne se trouve une et une seule information. LogAnalyser va en première phase lire chaque ligne et en analyser son premier mot, qui identifie le type d'information contenue dans celle-ci. Les blocs d'information du ficher journal sont délimités par BEG et END. La ligne précédant un bloc d'information indique quel est le type de celui-ci. Les trois types de bloc qui nous intéressent sont la création (CRE), la modification (MOD) et la suppression (DEL) d'objets. En cas de création (voir *Figure 6.3*, bloc ***CRE ENT**) on insert l'information OID, NAM et OWN dans le tableau aux lignes du même nom suffixé par Old puisque c'est là que doit se trouver l'information relative à la création des objets. On insert cette même information dans les lignes suffixées par New puisque celles-ci enregistrent les dernières modifications effectuées sur l'objet. Le statut est quant à lui mis à ON. Lors d'une modification, l'information pertinente de l'objet (l'OID, le NAM et le OWN) issue du journal est insérée dans les cellules situées à l'intersection des lignes ('OID New', 'NAM New' et 'OWN New') et de la colonne de l'objet. Si l'instruction est une suppression alors seul le 'statut' est modifié et est mis à OFF.

6.2.3 Lecture du fichier journal complémentaire

Le fichier journal accessoire trouve sa justification dans la non-complétude actuelle du fichier journal de DB-MAIN (DB-MAIN n'enregistre pas dans son « log » toutes les modifications apportées au schéma). Par exemple, l'information écrite dans le log par le module d'intégration se résume à donner la situation avant l'intégration et la situation après. Nul mot n'est dit à propos **de la**

manière dont ont évolué les objets du repository. Ce type de renseignement est pourtant indispensable pour notre application. Heureusement l'équipe de DB-MAIN a rapidement proposé une solution intermédiaire, celle-ci étant de modifier la fonction d'intégration pour qu'elle sauvegarde toute l'information désirée dans un fichier annexe.

```
####################
20:30
Type : 2
Master : E1
MasterId : 35
k j i h g f e d c b [ a r 323 ]
59 57 55 53 51 49 47 45 43 41 [ 39 61 323 ]
Slave : E2
SlaveId : 37
z y x w v u t s
77 75 73 71 69 67 65 63
```

Figure 6.4 – Extrait du journal d'enregistrement complémentaire de DB-MAIN.

Sur la ***Figure 6.4*** se trouve un bloc issu du journal complémentaire de DB-MAIN correspondant à une intégration. Un tel bloc a toujours la structure suivante (ligne par ligne) :

1) une ligne de séparation de blocs (####################)
2) l'heure à laquelle le bloc est écrit dans le fichier (cela facilite la localisation temporelle lors de la lecture par un humain)
3) le type d'intégration[16] :
   - 1 = copy slave into master : transfert les attributs, les unités de traitement, les rôles et les relations is-a dans un nouveau master (et crée ce master).
   - 2 = merge slave into master : fusionne les objets désirés du slave (transfert les attributs, les unités de traitement, les rôles et les relations is-a) avec le master. Si le slave est vide, il est effacé.
   - 3 = create a 1-1 link : crée une relation entre l'entité master et l'entité slave. Ses rôles sont (1-1/1-1) ou (0-1/1-1).
   - 4 = slave is-a master : crée un relation is-a entre l'entité master et l'entité slave. Le master est le super type.
   - 5 = master is-a slave : crée un relation is-a entre l'entité master et l'entité slave. Le slave est le super type.
   - 6 = create common supertype : crée un super type commun à l'entité master et l'entité slave.
4) le nom du master
5) l'identifiant du master

[16] Le module d'intégration d'objet intègre deux objets (types d'entité, types de relation ou des attributs composés) appartenant au même schéma ou à deux schémas différents. Avant de lancer le module d'intégration il faut sélectionner un objet du schéma. Cet objet est appelé « master ». S'il n'y a pas d'objet sélectionné dans le schéma, seul le type d'intégration 1 est disponible. La sélection de l'objet « slave » se fait à l'intérieur du module d'intégration.

6) l'ensemble des objets du master. Une liste du type [x,y,s] signifie que l'objet x du master et l'objet y du slave ont subi une intégration dont les paramètres sont s (s est de type entier codant une chaîne de bits). Les différents paramètres d'intégration sont :

Bit	Signification	Bit	Signification
1	not use	6	1 = type of non selected object 0 = type of selected object
2	1 = master selected 0 = slave selected	7	1 = sem. of master selected 0 = sem. of master non selected
3	1 = name the non selected object 0 = name the selected object	8	1 = sem. of slave selected 0 = sem. of slave non selected
4	1 = shortname the non selected object 0 = shortname the selected object	9	1 = tech. of master selected 0 = tech. of master non selected
5	1 = cardinality the non selected object 0 = cardinality the selected object	10	1 = tech. of slave selected 0 = tech. of slave non selected

7) les identifiants de chacun des objets de la liste située à la ligne 6. Les paramètres de la ligne 6 sont également recopiés.
8) le nom du slave
9) l'identifiant du slave
10) idem ligne 6 mais appliqué au slave
11) idem ligne 7 mais appliqué au slave

En parcourant le journal complémentaire, le programme LogAnalyser va mettre à jour le tableau de suivi des objets (exemple à la *Figure 6.2*). Notons que cette solution est temporaire puisque le journal complémentaire devrait à terme être intégré au journal principal. Quand tel sera le cas et si cette intégration se fait en suivant les règles déjà établies pour le journal existant, la procédure qui lit le journal existant ne devrait pas être modifiée car elle a été pensée sur base d'un seul journal complet. Seule la fonction de lecture du journal complémentaire devrait être supprimée du programme LogAnalyser.

6.2.4 Le code

Le code du programme LogAnalyser a été reporté en Annexe 2.

6.2.5 Exécution de LogAnalyser

L'exécution du programme LogAnalyser est sans surprise si ce n'est que le journal complémentaire est ajouté à la fin du fichier 'tdest.dbm'[17] qui se situe à la racine du disque C. Ce qui oblige d'effacer le fichier avant d'intégrer les sous-schémas conceptuels (une seule fois seulement). Remarquons que seule la fonction d'intégration va écrire dans le journal complémentaire.

Il suffit de lancer l'exécutable Voyager LogAnalyser après avoir réalisé toutes les étapes d'intégration. Le programme va lire les fichiers de log (le principal et le complémentaire), créer un tableau de suivi des objets puis demander à l'utilisateur de donner un nom ainsi qu'un emplacement au fichier qui va accueillir le tableau sous forme de texte.

[17] Si le fichier tdest.dbm n'existe pas à la racine du disque C, le programme le crée lui-même.

Chapitre 7

Troisième support outil - Le composeur et le décomposeur de business objects

Objectif :

Le but de ce chapitre est de construire un prototype d'applications capable de générer des composeurs/ décomposeurs (dernière étape de notre démarche). La principale difficulté de cet exercice est la rédaction d'une fonction de génération automatique de requêtes SQL.

7.1 L'IDÉE DU COMPOSEUR/DÉCOMPOSEUR

La dernière étape du développement assisté d'applications business objects se compose de l'instanciation et de la gestion du contenu des business objects. En d'autres termes, cette phase comprend la création de la structure d'arbre et la gestion de l'information qu'elle contient. La gestion de l'information se fait dans deux sens. Dans un sens, il s'agit de remplir le business object en utilisant les données trouvées dans la base de données et dans l'autre de le décharger et d'insérer ses informations dans cette dernière. Le composeur/ décomposeur est un programme chargé de gérer cette dernière étape. Le générateur de composeur/décomposeur va créer la structure d'arbre, générer des requêtes SQL sur base du schéma de la base de données et de l'arborescence du business object et intégrer le tout dans le programme qui héberge les services côté serveur.

7.2 CRÉATION DE LA STRUCTURE D'UN BUSINESS OBJECT

La structure d'un business object est celle d'un arbre. La création de ce type d'arborescence ne pose aucun problème puisque lors de la construction du prototype de fenêtre, l'utilisateur introduit lui-même une structure qui, même sans être apparente, est une structure d'arbre. L'introduction involontaire de cet agencement est induite par le respect des règles de construction d'un maquette d'écran. Cette correspondance est montrée distinctement à la *Figure 7.1* (la maquette d'écran) et à la *Figure 7.2* (la structure correspondante).

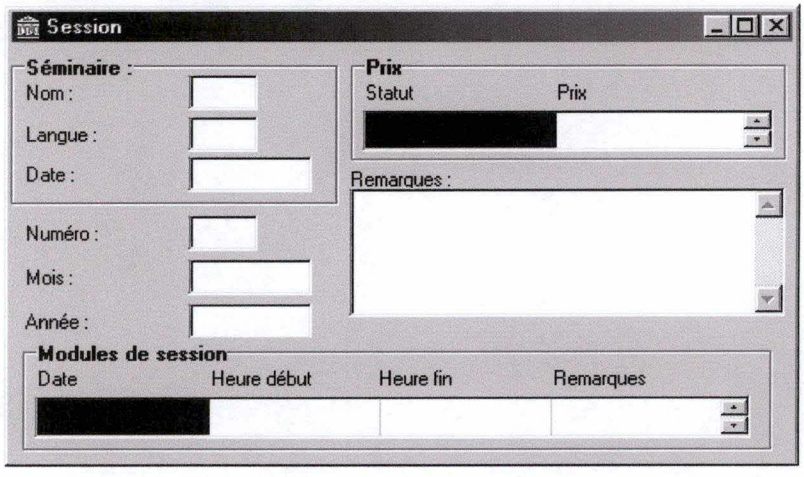


Figure 7.1 – Une maquette d'écran.

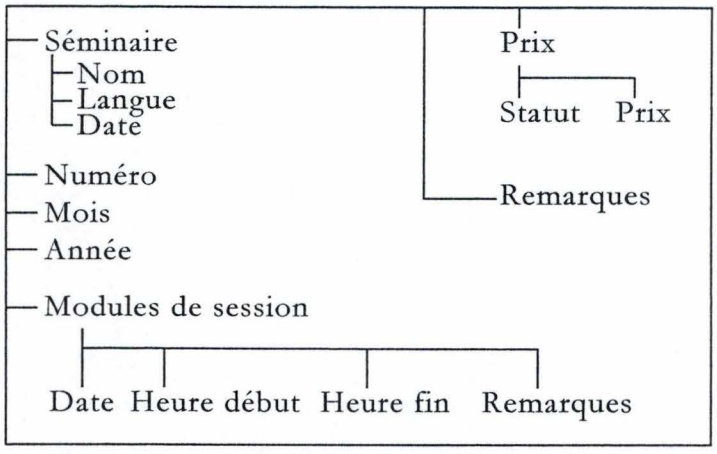


Figure 7.2 – La structure (d'arbre) correspondant à la maquette d'écran de la *Figure 7.1.*

La génération du code de l'arborescence est quelque chose d'assez élémentaire, c'est pour cela qu'elle ne sera pas décrite ici. La méthode qui la réalise peut être complètement automatique, mais l'utilisateur doit pouvoir interagir avec elle. C'est à lui de décider si le business object sera uniquement présent sur le serveur ou s'il pourra transiter du côté client. Dans ce dernier cas, l'objet se doit d'être « sérialisable ».

7.3 CRÉATION AUTOMATIQUE DES REQUÊTES

7.3.1 Méthode générale de génération des requêtes *[BRO]*

Il est intéressant de disposer d'une méthode automatique ou semi-automatique capable de dériver directement des requêtes SQL en se basant sur les maquettes d'écran. Ce sont ces requêtes qui permettent de garnir les écrans de l'application à partir des enregistrements de la base de données. La première étape de cette méthode consiste à découper une maquette d'écran en différentes zones. La première correspond à l'ensemble des composants qui sont directement attachés au prototype d'écran. Chaque autre zone est délimitée par une boîte de regroupement. Une zone peut être composée d'autres zones. Pour chaque zone, il faut produire la liste des attributs qui correspondent aux composants qui la constituent. Ces composants peuvent appartenir à plusieurs tables, cela n'a pas d'importance. A partir de ce moment, la rédaction de la requête SQL est automatique. S'il y a m attributs qui correspondent aux composants à remplir de la fenêtre, que ces attributs sont répartis dans n tables, alors la requête SQL s'écrit :

```
select attribut_1, attribut_2, ..., attribut_m
from table1, table2, ..., table n
where jointure_1
and jointure_2
and ...
and jointure_n-1
and identifiant_de_l'objet
```

7.3.2 Implémentation de la méthode de génération des requêtes

La partie la plus difficile de l'application qui génère un composeur/ décomposeur est sans conteste la génération automatique des requêtes. Même si la méthode est automatique, elle n'est pas évidente à implémenter. Comme il a été expliqué en détail à la *Figure 6.1* (page 52), un des problèmes vient de la phase d'intégration. Cette étape modifie considérablement les sous-schémas ce qui casse la correspondance entre une maquette d'écran et un sous-schéma. Cette correspondance se trouve dans le fichier d'évolution des objets créés par la programme LogAnalyser. Un prototype de programme de génération de requêtes a été créé, nous allons en expliquer le fonctionnement.

a) Les entrées

Les informations qui permettent de créer des requêtes de manière automatique ont trois origines. Il y a bien entendu le fichier de sortie de LogAnalyser qui reprend l'évolution des objets durant l'intégration et le fichier texte qui représente la maquette d'écran. Ce dernier permet de savoir dans l'ensemble des sous-schémas conceptuels créés par DFM2ET quel est celui correspondant à la maquette d'écran en simulant la génération du sous-schéma. Le troisième fichier à donner en entrée est le script de création de la base de données (le fichier SQL). Celui-ci contient un type d'élément essentiel que les autres n'ont pas : les clés étrangères. Elles sont utilisées pour réaliser les jointures entre les différentes tables où ont été répartis les attributs représentant les boîtes d'édition de la maquette d'écran.

b) Fonctionnement

Après l'introduction par l'utilisateur des chemins des différents fichiers d'entrée, le programme va lire ces fichiers et stocker l'information qui lui est utile dans des tableaux :

- Le premier tableau construit à partir du fichier d'évolution des objets du schéma conceptuel global est identique à celui utilisé par le programme LogAnalyser (voir *Figure 6.2* page 53).
- Le second tableau est construit à partir du fichier texte de description de la maquette d'écran. Il se compose d'autant de colonnes que la fenêtre contient d'objets possédant un champ 'Caption'. Sur la première ligne se trouve la valeur contenue dans le champ 'Caption' et sur la deuxième le niveau de l'objet. Les objets de ce tableau ne seront pris en compte dans le reste de l'application que s'ils existent dans le premier tableau (créé au point ci-dessus)[18]. De cette manière, les objets de la maquette de fenêtre qui possèdent un 'Caption' mais qui n'ont pas été repris lors de la génération de l'entité (celle qui correspond à la fenêtre) sont exclus.
- Le troisième et dernier tableau est celui qui reprend les informations relatives aux clés étrangères. Il est construit à partir du script SQL de génération de la base de données et est un peu plus compliqué que les deux

[18] En d'autres termes, les objets du tableau 2 qui seront utilisés pour la suite du programme correspondent à l'intersection de ce tableau avec le tableau 1.

premiers (voir *Figure 7.3*). Chacune de ses colonnes représentent une table de la future base de données.

1)	Nom de l'entité		
2)	Clé primaire[19]		
3)	Nombre de clés étrangères		
4)	Clé étrangère n° 1		
5)	Clé étrangère n° 2		
6)	Clé étrangère n° 3		
...)	Clé étrangère n°...		

Figure 7.3 – Tableau 3 : les clés étrangères.

Une fois que les tableaux sont créés, le programme est prêt à générer la première requête SQL. Il y a autant de requêtes à créer qu'il y a de groupes de composants dans la maquette d'écran. Par exemple il y en aurait quatre pour le prototype d'écran de la *Figure 7.1* (voir page 58): la première s'occupe des composants se trouvant directement sur la fenêtre (numéro, mois, année, remarques), les trois autres s'occupent de chacune des boîtes de regroupement.

La génération d'une requête est composée de deux étapes. La première s'occupe du « select ... from ... », la seconde du « where ...».

Le « select ... from ... »

Pour générer la partie « select ... from ... » de la requête, le programme va pour l'ensemble des composants qui se trouvent dans la même boîte de regroupement (à l'exclusion de tous ceux qui sont eux-mêmes composés) rechercher dans le premier tableau comment les attributs qui les représentent ont évolué. Ce même tableau permet de connaître le père de chacun des attributs ce qui récursivement permet de savoir à quelle table chaque attribut appartient. La première ligne de la requête est alors composée d'un « select » suivi de tous les attributs les uns derrières les autres (séparés par des virgules). La ligne « from ... » est composée des noms des tables, dont sont issus ces attributs, mises les unes derrière les autres. Ces tables sont les ancêtres les plus haut dans la hiérarchie des attributs. Il est à noter qu'une table qui existe déjà dans le « from ... » ne doit pas y être insérée une deuxième fois. Il arrive que quelques tables 'techniques' doivent parfois être ajoutées dans cette ligne, cela est dû au problème posé par les clés étrangères expliqué en détail ci-dessous.

Le « where ... »

La partie « where ... » de la requête SQL à générer est de loin la plus difficile. Sélectionner des attributs situés dans différentes tables qui sont adjacentes en terme de clés étrangères (i.e. liées directement) ne pose pas de

[19] Si plusieurs attributs composent la clé primaire, ils seront notés les uns dernière les autres, la barre verticale (|) servant de séparateur.

problème comme le montre la requête ci-dessous construite à partir de la *Figure 7.4*.

```
Select a, b, c, d, e, x, y, z
From E_1, E_2
Where E_1.a = E_2.a
```

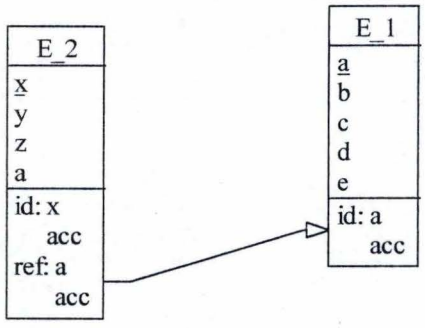


Figure 7.4 – Deux tables liées directement.

Le problème se complique fortement lorsque l'on désire extraire le contenu de lignes situées dans des tables liées par une chaîne de plusieurs clés étrangères à parcourir l'une après l'autre. Par exemple, à partir du schéma d'une base de données de la *Figure 7.5*, on désire sélectionner les lignes a, b, c, m, n et o. La requête s'écrit comme suit :

```
Select a, b, c, m, n, o
From E_1, E_3
Where E_1.a = E_2.a
And E_2.x = E_3.x
```

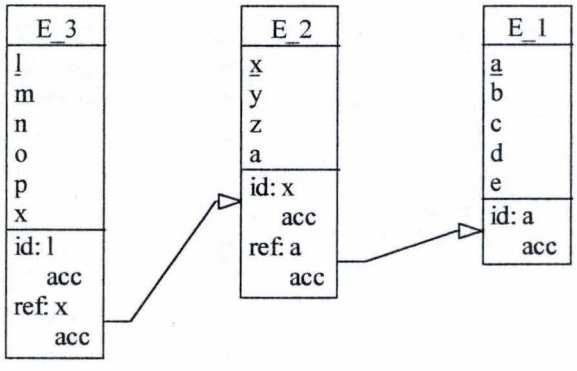


Figure 7.5 – La table E_1 et la table E_3 sont liées indirectement.

La solution pour résoudre le problème de chaînes de clés étrangères est de créer un graphe de relations entre tables et de rechercher le plus court chemin qui permet de relier l'ensemble des tables désirées. Ce problème est plus compliqué que la recherche du plus court chemin entre deux points dans un graphe puisque celui-ci est la recherche du plus court chemin entre n points d'un graphe (n étant le nombre de tables dans lesquelles se situent les lignes que l'on veut extraire). Comme on peut le voir sur le graphe de la *Figure 7.6*, il y a plusieurs chemins reliant l'ensemble des tables. Il ne faut pas penser que le plus court chemin est celui qui contient le moins d'arcs puisque la configuration technique de la base de données joue un rôle important en terme de performance. Par exemple, le fait qu'il y ait ou pas un index sur un identifiant d'une table

référencée par une clé étrangère modifie complètement le notion de plus court chemin. En réalité, ce n'est pas le plus court chemin qu'il faut rechercher mais le chemin le plus performant.

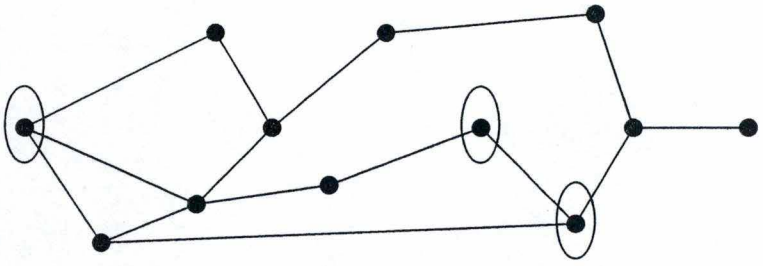


Figure 7.6 – Un exemple de graphe représentant des tables liées par des clés étrangères. Les tables où se trouve l'information désirée sont encerclées.

Chapitre 8

Étude de cas

Objectif :

Ce chapitre a pour but de montrer concrètement le déroulement de la méthode, plus ou moins automatisée, de création d'applications basées sur les bases de données relationnelles et les business objects. Les outils utilisés sont ceux présentés aux trois chapitres précédents.

8.1 MISE EN PROPOS

Le chapitre 3 a présenté une méthode de conception d'applications basée sur les business objects. Les chapitres 5, 6 et 7 ont décrit le fonctionnement de trois outils qui automatisent certaines étapes de la méthode. Le présent chapitre applique la théorie sur un exemple. Ce dernier n'est bien sûr pas complet car le temps et la place nous sont comptés mais il permet au lecteur d'avoir un aperçu global, sur un exemple simple, de cette nouvelle technique de développement assisté.

L'application que nous allons développer gère de manière quelque peu simpliste une base de données d'ouvrages littéraires. Elle se décompose en trois fenêtres, la première servant de menu général aiguillant vers les deux autres. De ces deux dernières, l'une permet d'introduire un ouvrage et les auteurs qui l'ont écrit tandis que la seconde permet de rechercher un ouvrage sur base de son code ISBN (code unique, utilisé partout dans le monde).

8.2 ÉTUDE DE CAS

8.2.1 Description des maquettes d'écran

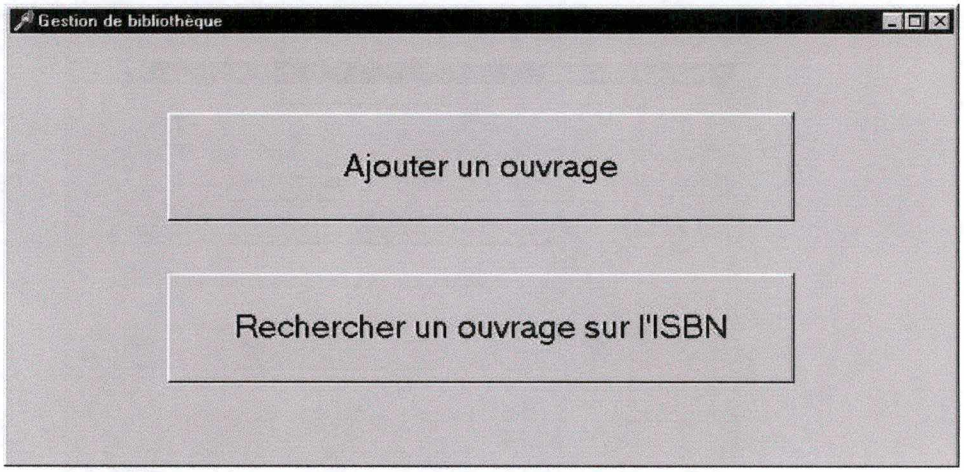


Figure 8.1 – Menu principal.

Le menu principal ne correspond à aucun processus d'affaires. Il aiguille vers les deux fonctions de l'application, toutes deux correspondant à des fonctions d'affaires. L'événement « OnClick » déclenché sur l'un des boutons rend visible la fenêtre correspondante.

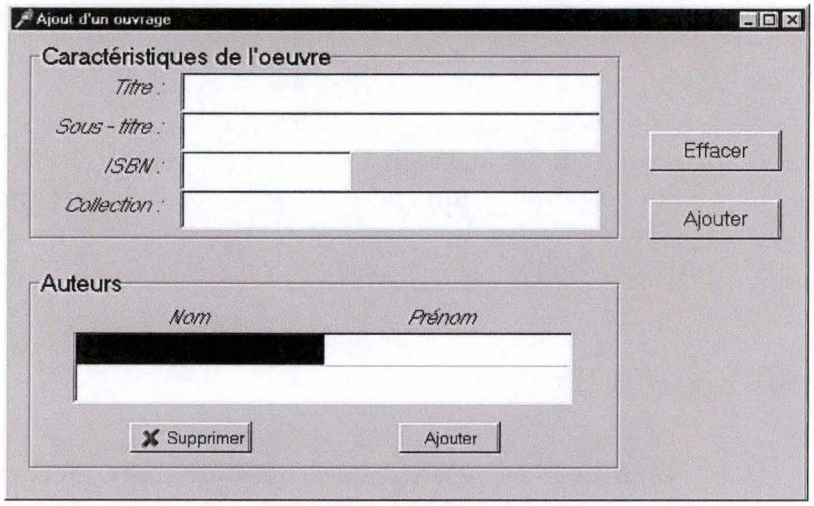


Figure 8.2 – Fenêtre d'ajout d'un ouvrage.

La fenêtre d'ajout d'un ouvrage (ci-dessus) permet à l'utilisateur d'ajouter une nouvelle œuvre dans la base de données. Cette œuvre comporte quatre attributs : le titre qui est la partie la plus significative du nom de l'œuvre ; le sous-titre, la partie la moins significative ; l'ISBN, le numéro d'identification unique de chaque livre ; la collection dont est issue le livre. De plus un livre possède un certain nombre d'auteurs qui chacun possède un nom et un prénom. On remarque directement les groupes qui composent la fenêtre : les propriétés de l'œuvre d'une part, les auteurs d'autre part. L'idée de fonctionnement de ce prototype d'écran est la suivante : l'utilisateur introduit de manière classique les caractéristiques de l'œuvre. Pour les auteurs, c'est différent. Seul le premier auteur peut être introduit (sur la première ligne). Pour ajouter une ligne et par conséquent permettre l'introduction d'un nouvel auteur, l'utilisateur doit appuyer sur le bouton 'Ajouter'. Pour supprimer la ligne de l'auteur activé, il faut appuyer sur 'Supprimer'. Le bouton principal 'Ajouter' (celui tout à droite) va insérer l'œuvre avec ses propriétés dans la base de données et renvoyer l'utilisateur à l'écran principal. Le bouton 'Effacer' va vider tous les champs de la fenêtre et ainsi rendre à l'utilisateur une fenêtre « propre ».

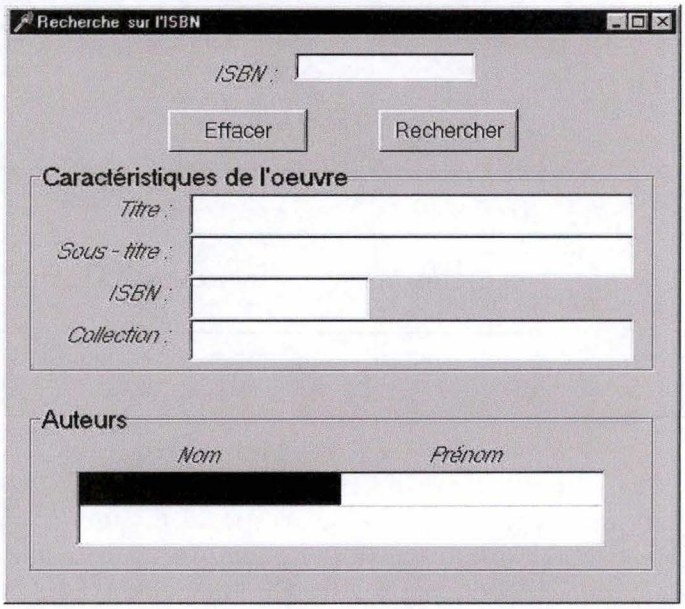


Figure 8.3 – Fenêtre permettant la recherche d'ouvrage à partir de l'ISBN.

Le haut de la fenêtre de recherche d'ouvrages sur l'ISBN propose un champ (nommé tout naturellement ISBN) que l'utilisateur est invité à remplir. Une fois cela fait, l'appui sur le bouton 'Rechercher' va exécuter la recherche. Dans le cas où l'ouvrage serait trouvé par le logiciel, ce dernier remplira tous les champs, disponibles dans la fenêtre, avec les valeurs trouvées. Dans le cas contraire, le comportement du logiciel est à programmer. Soit il n'affiche rien, soit il affiche un message de recherche infructueuse... Le bouton 'Effacer' sert à effacer le contenu des champs.

8.2.2 Identification des business objects

La taille des business objects pose toujours quelques problèmes et divise les chercheurs qui travaillent dans le domaine. Pour le cas présent et sans faire de théorie ici[20], nous pouvons considérer qu'il existe ici un business object que nous nommerons « Ouvrage », qui possède les propriétés classiques d'un ouvrage à savoir un titre, un sous-titre, un ISBN, une collection ainsi que différents auteurs. De plus notre business object possède deux méthodes qui lui sont attachées, une pour enregistrer l'information qu'il contient dans une base de données et l'autre pour rechercher l'information correspondante à un ouvrage dont seul l'ISBN est connu.

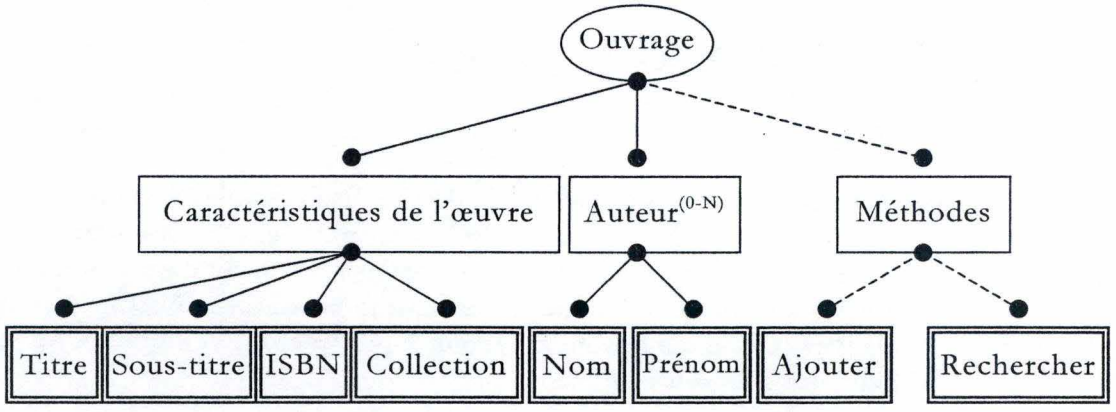


Figure 8.4 – Structure du BO Ouvrage.

8.2.3 Extraction de sous-schémas conceptuels

Nous allons extraire de façon 100% automatique deux sous-schémas conceptuels de nos deux fenêtres (voir *Figure* 8.2 et *Figure* 8.3). Un sous-schéma est extrait d'une fenêtre. Cette étape peut être réalisée de manière 100% automatique grâce au programme DFM2ET et DB-MAIN. Pour rappel, DFM2ET est un programme écrit en Voyager 2[21] décrit en détail au chapitre 5.

---

[20] Pour plus d'informations, voir chapitre 4 de ce même document.

[21] Voyager 2 est un langage qui permet d'écrire des programmes spécifiques pour DB-MAIN. Son but est de combler l'écart entre les fonctions fournies avec DB-MAIN et celles attendues par les utilisateurs.

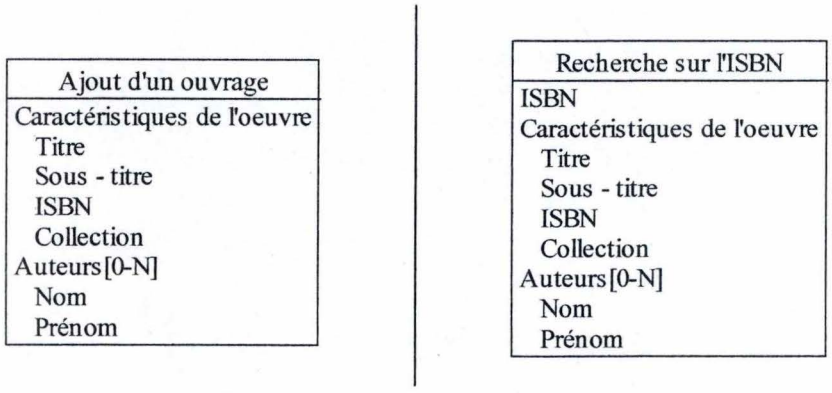


Figure 8.5 – Sous-schémas conceptuels des prototypes "Ajout d'un ouvrage" à gauche et "Recherche sur l'ISBN" à droite générés par DFM2ET.

8.2.4 Intégration des sous-schémas en un schéma conceptuel global

Avant d'intégrer ces sous-schémas, il nous faut les retravailler car ils sont encore « bruts ». C'est la conceptualisation des vues issues des maquettes d'écran.

a) Conceptualisation de la vue « ouvrage »

Étapes de la conceptualisation

- Extraction de l'attribut multivalué décomposable « Auteurs » sous forme d'un type d'entité (« Auteur »), par représentation des instances. La représentation par valeur aurait été plus judicieuse mais elle a le désavantage de créer une relation N-N. Celle-ci allait être traduite dans le schéma logique par une nouvelle table, table de liens qui allait alourdir l'exemple.
- Désagrégation de l'attribut décomposable « Caractéristiques de l'œuvre ».
- L'attribut « Ou_ISBN » est déclaré identifiant.

Schéma conceptuel

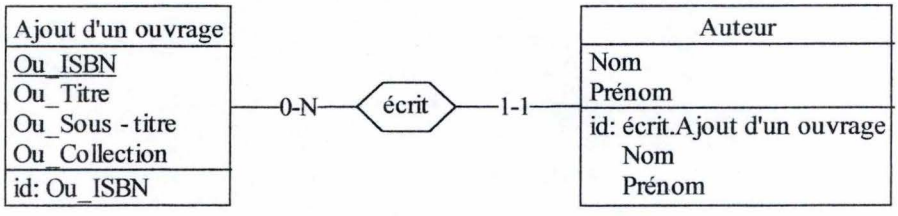


Figure 8.6 – Le sous-schéma « ouvrage » sous sa forme conceptuelle.

b) Conceptualisation de la vue « recherche »

Étapes de la conceptualisation

- Extraction de l'attribut multivalué décomposable « Auteurs » sous forme d'un type d'entité (« Auteur »), par représentation des instances. La remarque de l'étape de conceptualisation de la vue « ouvrage » peut également être appliquée à ce choix.
- L'attribut ISBN de l'entité « Recherche » et l'attribut ISBN de l'attribut « Caractéristiques de l'œuvre » de la même entité sont les mêmes. On les fusionne de manière à ce que le premier disparaisse.
- Désagrégation de l'attribut décomposable « Caractéristiques de l'œuvre ».
- L'attribut « Ou_ISBN » est déclaré identifiant.

Schéma conceptuel

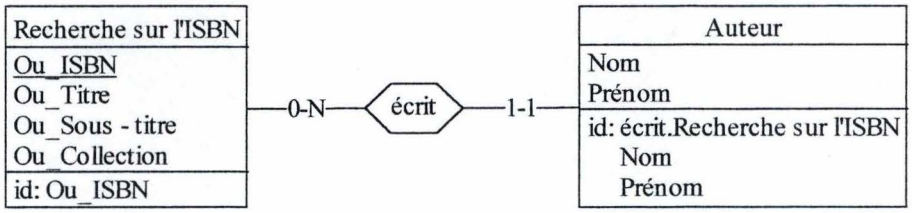


Figure 8.7 – Le sous-schéma « Recherche » sous sa forme conceptuelle

c) Intégration des sous-schémas conceptuels en un schéma conceptuel global

L'intégration des sous-schémas conceptuels permet d'obtenir le schéma conceptuel global qui décrit l'ensemble du domaine d'applications.

Les deux sous-schémas ci-dessus sont tellement proches que l'on remarque rapidement qu'ils sont identiques mis à par le nom d'une de leurs entités. L'intégration de deux sous-schémas aussi proches est immédiate puisque tous les attributs du sous-schéma « ouvrage » ont une correspondance dans le sous-schéma « recherche ». Au passage, tous les attributs préfixés par « Ou_ » ont perdu ce préfixe.

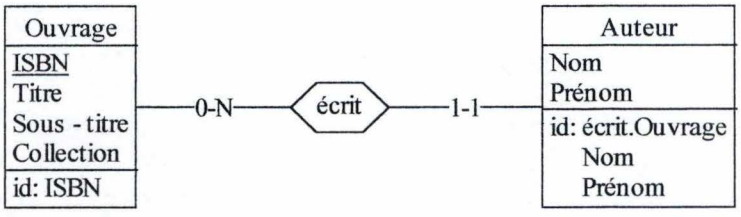


Figure 8.8 – Le schéma conceptuel global de l'application.

8.2.5 Développement d'une base de données relationnelle

A partir du schéma conceptuel global, nous allons créer une base de données relationnelle, c'est-à-dire passer par un schéma de conception logique et physique. Les étapes de transformation sont :

- Transformation des types d'associations 'un à un' ou 'un à plusieurs' en clés étrangères.
- Finalisation : conditionnement des noms, domaines de valeurs des colonnes, etc.
- Création de clés d'accès.

Suivant la complexité du schéma, d'autres étapes devront être ajoutées mais dans notre cas, nous sommes en présence d'un schéma simple et déjà fort proche d'un schéma physique.

Le traitement des noms est un traitement classique : suppression des accents, mise en majuscules, remplacement des caractères non autorisés...

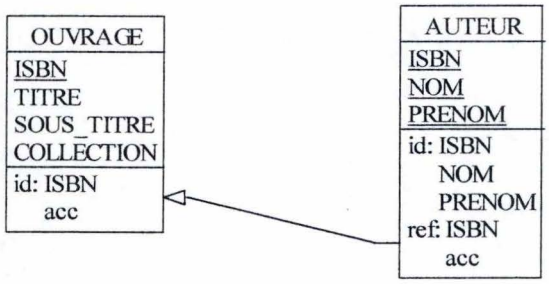


Figure 8.9 – Schéma physique.

```
create table AUTEUR (
     ISBN char(1) not null,
     NOM char(1) not null,
     PRENOM char(1) not null,
     primary key (ISBN, NOM, PRENOM));

create table OUVRAGE (
     ISBN char(1) not null,
     TITRE char(1) not null,
     SOUS_TITRE char(1) not null,
     COLLECTION char(1) not null,
     primary key (ISBN));
```

Figure 8.10 – Le script de création des tables

8.2.6 Un business objet, une vue

L'outil décrit au chapitre 7 crée automatiquement des requêtes qui permettent de garnir les écrans de notre application à partir de la base de données. Ces requêtes fonctionnent mais ne sont généralement pas les plus performantes. Le programme qui génère ces requêtes prend en entrée le fichier texte représentant la fenêtre, le fichier de suivi des objets créé par LogAnalyser[22] (il faut donc l'exécuter avant de pouvoir générer des requêtes) et le script SQL de création de la

[22] LogAnalyser est décrit en détail au chapitre 6.

base de données. Le programme génère une requête pour chaque boîte de regroupement de la maquette d'écran.

Pour la recherche sur l'ISBN le programme va générer deux requêtes. La première pour rechercher l'information concernant directement l'ouvrage (ISBN, titre, sous-titre et collection). La seconde s'occupera de l'information relative aux auteurs de cet ouvrage (nom et prénom des auteurs).

```
Select ISBN, TITRE, SOUS_TITRE, COLLECTION,
From OUVRAGE
Where ISBN = :SQL_ISBN

Select NOM, PRENOM
From AUTEUR
Where ISBN = :SQL_ISBN
```

La dernière ligne de chacune de ces requêtes doit être ajoutée par l'utilisateur, elle permet de sélectionner un ouvrage particulier (choisi ici sur base de l'ISBN). SQL_ISBN est un paramètre donné avant l'ouverture de la requête. Pour la première requête le résultat sera composé de maximum une ligne puisque ISBN est identifiant dans la table (zéro ligne si l'ouvrage recherché n'existe pas). La deuxième requête va renvoyer un ensemble de lignes comme résultat (autant que l'ouvrage a d'auteurs) et l'application qui l'a exécutée devra le parcourir itérativement.

Pour l'insertion d'un ouvrage et de ses auteurs, les requêtes ne sont pas plus difficiles mais elles sont au nombre de deux et doivent être exécutées dans un ordre stricte pour ne pas engendrer de violation de contraintes référentielles. Il faut d'abord insérer l'ouvrage dans la table 'OUVRAGE' et puis seulement ses auteurs dans la table 'AUTEUR'. Pour l'instant aucun programme ne permet de générer de telles requêtes.

```
Insert Into OUVRAGE(ISBN, TITRE, SOUS_TITRE, COLLECTION)
values (
          :SQL_ISBN,
          :SQL_TITRE,
          :SQL_SOUS_TITRE,
          :SQL_COLLECTION
       )
Insert Into AUTEUR(ISBN, NOM, PRENOM)
values (
          :SQL_ISBN,
          :SQL_NOM,
          :SQL_PRENOM
       )
```

La deuxième requête doit être exécutée autant de fois que l'ouvrage a d'auteurs. Pour faciliter le traitement des erreurs, il est nécessaire d'encapsuler ces deux insertions dans une transaction qui en cas de problème peut être annulée (rollback).

8.2.7 Aspect technique et implémentation des business objects

Un schéma de la structure du business object de notre exemple a déjà été donné à la *Figure 8.4*, page 67. Nous allons ici nous intéresser à sa transformation en code, Java pour l'occasion. Par rapport au schéma présenté ci-dessus, rien n'a changé si ce n'est que certains noms ont été raccourcis. La structure est identique. Les méthodes implémentant les règles d'affaires attachées aux business objects doivent encore être ajoutées. Le nombre d'auteurs maximum pour un ouvrage a été limité à 10. Toutes les classes dérivent de l'interface « Serializable » qui leur permet d'être transférées à travers le réseau. Toute l'information contenue dans la structure est accessible au travers de méthodes GetNomVariable et GetNomVariable.

```
import java.io.*;
class Ouvrage implements Serializable
{
 Oeuvre oeuvre;
 Auteur[] auteur;
 Methode methode;
 Ouvrage()
 {
  oeuvre = new Oeuvre();
  auteur = new Auteur[10];
  for (int i = 1 ; i<=10 ; i++)
  {
   auteur[i-1] = new Auteur();
  }
  methode = new Methode();
 }
 class Oeuvre implements Serializable
 {
  Titre titre;
  Sous_titre sous_titre;
  Isbn isbn;
  Collection collection;
  Oeuvre()
  {
   titre = new Titre();
   sous_titre = new Sous_titre();
   isbn = new Isbn();
   collection = new Collection();
  }
  class Titre implements Serializable
  {
   private String titre;
   Titre() {}
   void SetTitre(String a){titre=a;}
   String GetTitre(){return titre;}
  }
  class Sous_titre implements Serializable
  {
   private String sous_titre;
   Sous_titre() {}
   void SetSous_titre(String a){sous_titre=a;}
   String GetSous_titre(){return sous_titre;}
  }
  class Isbn implements Serializable
  {
   private String isbn;
   Isbn() {}
   void SetIsbn(String a){isbn=a;}
   String GetIsbn(){return isbn;}
  }
  class Collection implements Serializable
  {
   private String collection;
   Collection() {}
   void SetCollection(String a){collection=a;}
   String GetCollection(){return collection;}
  }
}
```

```
class Auteur implements Serializable
 {
  Nom nom;
  Prenom prenom;
  {
   nom = new Nom();
   prenom = new Prenom();
  }
  class Nom implements Serializable
  {
   private String nom;
   Nom() {}
   void SetNom(String a){nom=a;}
   String GetNom(){return nom;}
  }
  class Prenom implements Serializable
  {
   private String prenom;
   Prenom() {}
   void SetPrenom(String a){prenom=a;}
   String GetPrenom(){return prenom;}
  }
 }
}
```

8.2.8 Méthodes attachées aux business objects

Comme dit précédemment, les méthodes attachées aux business objects doivent être écrites à la main. Si nous mettons de côté les méthodes qui permettent d'accéder et de modifier les valeurs des variables (GetNomVariable et SetNomVariable), il ne reste plus à ajouter à notre business object qu'une méthode pour lancer la recherche sur l'ISBN et une autre pour insérer les valeurs qu'il contient dans la base de données (qui se base en grande partie sur les requêtes établies au point 8.2.6). D'autres méthodes issues des règles d'affaires pourraient voir le jour. Par exemple qu'une collection a toujours le même préfixe d'ISBN. Ces méthodes doivent pour l'instant encore être écrites de manière classique.

8.2.9 Construction des écrans de dialogue

La reconstruction des écrans de dialogue va se baser entièrement sur les prototypes d'écran puisque les différentes étapes de la méthode n'ont rien changé du point de vue de l'interface homme machine. Pour rappel, les captures d'écran sont visibles à la page 66.

8.2.10 Intégration dans une architecture distribuée

L'intégration de notre application dans une architecture distribuée est plus facile que cela en a l'air. Notre business object en tout cas est déjà prêt pour ce passage puisque toutes ses classes implémentent l'interface « Serializable » qui permet à Java d'aplatir une structure (d'arbre dans notre cas) de manière à pouvoir la transporter sur un réseau ou la stocker sur un disque. Le reste correspond à la mise en place d'une architecture distribuée. Le chapitre 4 y est principalement consacré.

8.2.11 Tests, évaluation et documentation

La dernière étape, celle qui réalise les différents tests qui évaluent les résultats et créent la documentation du logiciel n'a rien de spécifique et toutes les méthodes de tests, d'évaluation et de documentation utilisées lors du développement classique de logiciel sont également valables dans notre cas.

Chapitre 9

Conclusion & perspectives

9.1 CONCLUSION

Ce travail propose une nouvelle méthode qui, se basant sur DB-MAIN, permet de créer des applications de façon assistée, l'architecture de ces dernières reposant sur des business objects et des bases de données relationnelles. Cette méthode n'a pas la prétention de construire les applications les plus performantes, tel n'est d'ailleurs pas son but ; mais elle a pour objectif de développer des applications évolutives en un minimum de temps. La méthode est exposée de façon théorique au chapitre 3 et un cas concret et complet d'utilisation de cette méthode ainsi que des applications développées durant ce travail est décrit au chapitre 8.

Nous avons essayé de donner, dans le chapitre 2, une définition des business objects le plus en accord possible avec l'ensemble de la discipline. Ce n'était pas chose facile au vu de l'ensemble des spécifications existantes qui, même si elles ne sont pas incompatibles entre elles, indiquent qu'il règne une certaine confusion dans le milieu des objets lorsqu'il est question d'aborder ces derniers sous leur forme « business ». En restant en total accord avec cette définition, qui semble être la plus utilisée, nous avons pris quelques libertés qui confèrent à ce travail un côté novateur. D'une part le modèle business object et celui des bases de données relationnelles n'ont jamais été dissociés. En effet, ce sont ces dernières qui fournissent aux instances de business objects un moyen de stockage de l'information qu'ils contiennent. D'autre part, alors qu'il existe un courant qui fournit une notation de spécification des business objects (celle-ci ressemble plus ou moins à UML) nous avons opté pour une toute autre voie. Notre solution permet de créer des business objects de manière complètement automatique à partir de maquettes d'écran qui sont créées lors de tout développement d'applications de gestion. Le chapitre 5 décrit les quelques règles peu contraignantes à respecter lors de la création de prototypes d'écran.

Pour être plus complet, le chapitre 4 aurait dû aborder l'idée d'un programme d'aide à la conception d'applications RMI. Par exemple, il est inutile que le programmeur encode l'interface alors qu'elle peut être facilement générée automatiquement. Il en est de même pour l'application du côté serveur qui héberge les services définis par cette interface. Malgré tout, ce chapitre explique rapidement l'architecture client/serveur deux et trois tiers ainsi que la technologie RMI. Il décrit également toutes les étapes nécessaires à la création d'une application client/serveur qui utilise cette technologie. Ce chapitre donne, à quiconque connaissant la philosophie Java, assez de notions pour pouvoir réaliser une application basée sur RMI.

Dès le début, ce mémoire était prévu pour être assez technique et il l'a été. Au final, trois programmes correspondants à trois étapes de la méthode de développement ont vu le jour. Le chapitre 5 décrit en détail le premier d'entre eux. Il s'agit de celui qui transforme automatiquement une maquette d'écran en une entité correspondante à un des sous-schémas conceptuels de l'application. Le chapitre 6 est lui aussi consacré à l'explication d'une application. Celle-ci se situe dans l'ensemble de la démarche juste après l'application décrite au chapitre 5. Elle s'occupe de retracer l'évolution des différents objets composant les sous-schémas conceptuels qui a eu lieu lors de l'intégration. Ces deux programmes fournissent des résultats satisfaisants même si leur intégration complète à DB-MAIN (c'est-à-

dire sous une forme autre que celle d'un programme Voyager externe) pourrait en accroître la performance et surtout en diminuer l'aspect « quick fix » qu'ils donnent d'eux. Nous pensons particulièrement ici au fait que le module d'intégration est obligé de transmettre la liste de ses modifications au second programme via un fichier texte.

Le chapitre 7 a montré et solutionné les problèmes relatifs à la génération automatique de requêtes SQL. Cette solution utilise naturellement les résultats issus des étapes situées en amont de la démarche de conception automatisée d'applications. Le générateur de composeur/décomposeur de business objects n'est pas réalisé mais toutes les pièces qui le composent le sont. Il faut maintenant les assembler.

Au final, les apports fondamentaux de cette étude sont au nombre de trois : tout d'abord, ce travail constitue une première approche d'une nouvelle méthode de développement assisté et il assure que la démarche suivie est la bonne. Ensuite, il apporte une première version d'applications qui automatise la démarche de conception. Pour terminer, et c'est le plus important, il confirme que des outils d'aide à la conception sont implémentables pour toutes les étapes constituant la démarche.

9.2 PERSPECTIVES

Comme nous l'avons dit au début, ce travail n'a jamais prétendu être exhaustif. Le chemin est long mais le fait de l'avoir trouvé est déjà une victoire en soi. De nombreux points doivent encore être approfondis. L'ordre choisi pour citer ceux-ci est celui de l'enchaînement des étapes de conception.

Lors de la création de prototypes d'écran, il sera intéressant de posséder un outil propre qui permettrait de se passer de Delphi (dont la nécessité d'utilisation peut être considérée comme une contrainte) mais surtout qui rendrait possible la définition précise des champs, des types de valeurs, de la sémantique et des règles de cohérence de la maquette. Cet outil pourrait en outre permettre d'identifier les business objects de base, une maquette d'écran pouvant en regrouper plusieurs. Pour être complète, l'identification devrait être accompagnée d'une théorie décrivant la taille optimale d'un tel objet. Pour rappel, actuellement nous réduisons le problème au fait qu'un écran ne représente qu'un business object.

Le dernier point à ajouter au processus pour qu'il soit complet est la définition des règles et des méthodes attachées aux business objects qui en définissent le comportement, l'intégrité, ... Même si ce point n'est pas essentiel, il serait dommage de laisser l'utilisateur les rédiger seul alors que jusque là l'ensemble du développement est assisté.

Bibliographie

[AAR96] Amund Aarsten, Davide Brugali, Giuseppe Menga, "Patterns for Three-Tier Client/Server Applications", *Proceedings of the Third Conference on the Pattern Languages of Programs (PLoP'96)*, 3-6 September 1996, Illinois, USA.

[DBM95] DB-MAIN project, *Tutorial Volume 1 Introduction to Database Design*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1995.

[DBM98] DB-MAIN project, *The DB-MAIN Database Engineering CASE tool, Version 4, Functions Overview*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, November 3, 1998.

[BRO99] Anne-France Brognaux, *Gestion et organisation de séminaires*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium 1999.

[BUR95] Carol Burt, "OMG BOMSIG survey with published definition of a business object",Object Managment Group, document 95-02-04. Www.omg.org

[CAS] Cory Casanave, "Business-Object Architectures and Standards", *Data Access Corparation*, Miami, USA, 1996. cory_casanave@omg.org

[CER97] Stefano Ceri and Piero Fraternali, *Designing Database Applications with Objects and Rules : The Idea Methodology*, Addison-Wesley, Paris, France, 1997.

[DAT99] Data Access Technologies, "Business Object Component Architecture, Specification, Version 1.3", *Data Access Technologies*, May 1999. www.dataaccess.com/dat

[DEV] Pr. Deville, *Cours de Calculabilité et complexité*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium.

[DIG99] Tom Digre, "Business Object Component Architecture KomponentTorget99", *Data Access Technologies*, 1999.

[ECK00] Bruce Eckel, *Thinking in Java, Second Edition*, Prentice Hall, New Jersey, United States of America, 2000. ISBN 0-13-027363-5.

[ENG98] Vincent Englebert, *Voyager 2 Reference Manual Version 4 Release 0*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, November 1998.

[HAI94] Jean-Luc Hainaut, *Bases de données et modèles de calcul, Outils et méthodes pour l'utilisateur*, InterEditions, Paris, France, 1994. ISBN : 2-7296-0516-9

[HAI99a] Jean-Luc Hainaut, *Computer-Aided Database Engineering, Volume 1 : Database Models, DB-MAIN Tutorial*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1999.

[HAI99b] Jean-Luc Hainaut, *Computer-Aided Database Engineering, Volume 2 : Database Engineering Techniques, DB-MAIN Tutorial*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1999.

[ENC98] "Programmation, langage de", *Encyclopédie® Microsoft® Encarta 98*. 1998.

[HEI99] Peter M. Heinckiens, *Bulding Scalable Database Applications, Object-Oriented Design, Architectures, And Implementations*, Addison-Wesley, Paris, France, 1999.

[HEN98] Jean Henrard, *Description du journal de DB-MAIN*, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, August 1998.

[HOR] W. Hordijk, S. Molterer, B. Paech, Ch. Salzmann, *Working with Business Objects: A Case Study*, Institut fur Informatik, Technische Universität München.

[JGU] jGuru, "Fundamentals of RMI, Short Course", *Java Developer Connection*, 25-Feb-2000, http://www.java.sun.com

[KEL] Wolfgang Keller "Object/Relational Access Layers - A Roadmap, Missing Links and More Patterns", Wien, Austria. http://ourworld.compuserve.com/homepages/WofgangWKeller/

[KIL] Haim Kilov, Bernhard Rumpe, "Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications)", Merrill Lynch, Operations, Services and Technology, World Financial Center, South Tower.

[MAH98a] Qusay H. Mahmoud, "Writing distributed applications in Java, part 1", *Microsoft Developer Network, Visual J++ Developer's Journal*, September 1998.

[MAH98b] Qusay H. Mahmoud, "Writing distributed applications in Java, part 1", *Microsoft Developer Network, Visual J++ Developer's Journal*, October1998.

[ORE] Jack Orenstein, *Supporting Retrievals and Updates in an Object/Relational Mapping System*, Novera Software, Inc., jack@novera.com.

[OUS99] Chabane Oussalah, *Génie Object : analyse et conception de l'évolution*, HERMES Science Publications, Paris, France, 1999.

[PAI] Richard F. Paige and Jonathan S. Ostroff "A Comparison of the Business Object Notation and the Unified Modeling Language" Department of Computer Science, York University, Toronto, Ontario M3J 1P3, Canada.

[RAM95] Sita Ramakrishnan "Object Frameworks - An Empirical Study of Software Architecture Issues", Department of Software Development, Monash University, Australia, June 1, 1995.

[SAL99] Chris Salzmann, *Managing Shared Business-Objects*, Institute for Informatics Munich University of Technology, Germany, April 29, 1999, salzmann@in.tum.de

[SUT] Dr. Jeff Sutherland, "Business Objects and the Evolution of the Internet", *IDX Systems Corporation*. http://www.jeffsutherland.org/papers/crcweb.html

[SUT95] Dr. Jeff Sutherland "Business Objects in Corporate Information Systems" *ACM Computmg Surveys*, Vol 27, No 2, June 1995.

[SUT97] Dr. Jeff Sutherland, D. PATAL, C. Casanave, J.Miller, G. Hollowell, *The Object Technology Architecture : Business Objects for Corporate Information Systems, Business Object Design and Implementation*, Springer, 1997.

[VAU97] Bill Vaughn, "Building Successful Client/Server Applications", *Microsoft Developer Network*, March 19, 1997.

[SUT] Jeff Sutherland, "Business Object Design and Implementation Workshop", *VMARK Software*, http://www.tiac.net/users/jsuth/

Annexes

Annexe 1

Code du programme Dfm2Et

```
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */
/*                                     */
/* Déclaration des variables globales */
/*                                     */
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */

  list: liste;

/* ^^^^^^^^^^^^^^^^^^^^^^^^^^ */
/*                            */
/* Déclaration des procédures */
/*                            */
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^ */

/* ------------------------------------------------------------------------------------------ */
/* Sélectionne un fichier à mettre dans le stream d'entrée à l'aide d'une boîte à dialogue.  */
/* ------------------------------------------------------------------------------------------ */
procedure OuvreTxtStream()
    file: fin;
  string: namefile;
{
    namefile:=BrowseRead("Open a dfm->txt file","Files TXT (*.txt)|*.TXT|All (*.*)|*.*","TXT");
    fin:=OpenFile(namefile,_R);
    SetParser(fin);
}

/* ------------------------------------------------ */
/* Utilisé pour passer les blancs en début de ligne. */
/* ------------------------------------------------ */
procedure Skip1stBlanc()
{
    SkipWhile(" ");
}

/* ------------------------------------------------------------------------------------------ */
/* Le string de sortie = le string d'entrée où il ne reste plus que les caractères suivants: */
/* de a à Z de 0 à 9 et - _ espace ainsi que les accents.                                     */
/* ------------------------------------------------------------------------------------------ */
function string CleanStr(string: motin)
  integer: blanc;
```

```
  integer: longueur;
   string: motout;
  integer: rail;
   string: temp;
{
  rail:=StrFindSubStr(motin,0,"'#39'");
  if rail<>-1 then {
                     temp:=StrGetSubStr(motin,0,rail)
                     +" "
                     +StrGetSubStr(motin,rail+4,StrLength(motin));
                     motin:=temp;
                   }
  motout:="";
  SetParser(motin);
  while nseof() do
    {
     SkipUntil("-a-zA-Z0-9_éèàùâêôëï ");
     motout:=motout+CharToStr(GetChar());
    }
  /* A partir d'ici on supprime les espaces en fin de ligne. */
  longueur:=StrLength(motout);
  blanc:=1;
  while blanc=1 do
  {
     if StrGetSubStr(motout,longueur-1,1)=" " then {longueur:=longueur-1;}
                                              else {
                                                      blanc:=0;
                                                   }
  }
  return StrGetSubStr(motout,0,longueur);
}

/* ------------------------------------------------------------------- */
/* La liste de sortie est la liste d'entrée où l'élément s a été inséré */
/* à l'emplacement i,j du tableau correspondant à la liste. Cela        */
/* permet d'utiliser l'avantage d'un tableau (index) avec ce qu'offre   */
/* Voyager: les listes.                                                 */
/* i=indice de la ligne ; j=indice de la colonne                        */
/* ------------------------------------------------------------------- */
function list SetTab2(list: tabin, integer: i, integer: j, string: s)
```

```
  integer: a;
   cursor: c;
   cursor: d;
  integer: longueur;
     list: temp;
{

   /* Si le tableau est vide, on l'initialise. */
   if Length(tabin)=0
      then {
                  tabin:=[0];
           }
   longueur:=Length(tabin)-1;
   /* Si la ligne dans laquelle l'élément doit être inséré n'existe pas, on doit la  */
   /* créer (ainsi que toutes celles entre la dernière ligne du tableau actuel et la */
   /* ligne (i) de l'élément à insérer.                                               */
   if i>longueur
      then {
              a:=1;
              attach c to tabin;
              while a<=longueur do {
                                     c>>;
                                 a:=a+1;
                                }
              while a<i do {
                                 c+>[0];
                                 c>>;
                                 a:=a+1;
                                }
             c+>[0];

           }
   /* Maintenant que le ligne i existe il faut se positionner sur cette ligne i. */
   attach c to tabin;
   c>>;
   a:=1;
   while a<i do {
                 a:=a+1;
                 c>>;
               }
```

```
   /* On est sur la bonne ligne, on positionne le curseur sur le 1er élément de */
   /* la ligne.                                                                  */
   attach d to get(c);
   longueur:=Length(get(c));
   a:=0;
   temp:=[];
   while ((a<j) and (a<longueur))
   do {
         temp:=temp++[get(d)];
         d>>;
         a:=a+1;
      }
   if a<j then {temp:=temp++[a..j-1];}
   temp:=temp++[s];
   /* On a copié dans temp le contenu de la ligne i de 0 à j. Il faut encore   */
   /* la fin de la ligne (j+1 -> la fin) si elle existe.                        */
   if a<longueur
   then {
           d>>;
           a:=a+1;
           while a<longueur do {
                                  temp:=temp++[get(d)];
                                  d>>;
                                  a:=a+1;
                               }
         }
   /* On insère la ligne temp à la place de la ligne i. */
   kill(c);
   c<<;
   c+>temp;
   return(tabin);
}

/* ---------------------------------------- */
/* Retourne l'élément i,j du tableau tabin. */
/* ---------------------------------------- */
function string GetTab2(list: tabin, integer: i, integer: j)
  integer: a;
   cursor: c;
{
```

```
   /* On se positionne sur la ligne i. */
   attach c to tabin;
   c>>;
   a:=1;
   while a<i do {
                  a:=a+1;
                  c>>;
                }
   /* Sur la ligne i on se positionne sur le j ème élément. */
   attach c to get(c);
   c>>;
   a:=1;
   while a<j do {
                  a:=a+1;
                  c>>;
                }
   /* On le renvoie comme résultat. */
   return get(c);
}

/* ------------------------------------------------------------------------- */
/* La liste de sortie est la liste d'entrée où 'dat' a été inséré en j ème position. */
/* Ceci permet d'indexer une liste.                                          */
/* ------------------------------------------------------------------------- */
function list SetTabDO(list: listin, integer: j, data_object: dat)
   cursor: c;
  integer: i;
     list: listout;
  integer: max;
{
      listout:=[];
      max:=Length(listin);

      /* Si la longueur de la liste (listin) est plus courte que l'indice j de l'élément à */
      /* insérer, on la complète d'entiers compris entre max+1 et j.                       */
      if j>max then {listin:=listin++[max+1..j];}
      attach c to listin;
      i:=1;
      while i<j do /* On copie dans la listout les j-1 1er éléments de listin. */
      {
```

```
        listout:=listout++[get(c)];
        c>>;
        i:=i+1;
      }
      listout:=listout++[dat]; /* On insère le dat en j ème position. */
      c>>;
      i:=i+1;
      /* On copie dans la listout les éléments de listin situés entre j+1 et la fin. */
      while i<=max do
      {
        listout:=listout++[get(c)];
        c>>;
        i:=i+1;
      }
      return listout;
}

/* ------------------------------------ */
/* Renvoie le j ème élément de la liste l. */
/* ------------------------------------ */
function data_object GetTabDO (list: l, integer: j)
   cursor: c;
  integer: i;
  integer: max;
{
      attach c to l;
      max:=Length(l);
      i:=1;
      if j>max then {print("Indice hors tableau");}
               else {
                      while i<j do
                      {
                        c>>;
                        i:=i+1;
                      }
                      return get(c);
                    }
}

/* ------------------------------------------------------------------------------ */
```

```
/* Transforme un tab en l'entité correspondante qui sera stockée dans le repository. */
/* ------------------------------------------------------------------------------- */
procedure List2ET(list: liste)
       integer: alength;
          char: atype;
        cursor: c;
         char : car;
  co_attribute: coa;
        cursor: e;
   entity_type: et;
   data_object: frere;
       integer: i;
       integer: j;
          list: listeParent;
       integer: max;
       integer: maxcard;
       integer: mincard;
       integer: niveau;
        string: nomet;
   data_object: pere;
  si_attribute: sia;
       integer: suiti;
        string: temp;
       integer: trouve;
       integer: trouve2;

{
    listeParent:=[];    /* On crée une liste pour savoir quel est le parent de tel attribut. */
    max:=Length(liste)-1;
    if max>=1 then      /* Si la liste est vide, il n'y a pas d'entité à créer, sinon GO.    */
    {
      /* Si la liste n'est pas vide le 1er élément est l'entité.                            */
      /* Transforme les caractères du nom de l'entité sur lesquels StrToUpper n'a pas       */
      /* de prise.                                                                          */
      SetParser(StrToUpper(CleanStr(GetTab2(liste,1,1))));
      nomet:="";
      while nseof() do
      {
         car:=GetChar();
         temp:=CharToStr(car);
```

```
   switch(temp)
   {
      case "é": car:='E';
      case "è": car:="E";
      case "ê": car:="E";
      case "ë": car:="E";
      case "ê": car:="E";
      case "à": car:="A";
      case "â": car:="A";
      case "ä": car:="A";
      case "î": car:="I";
      case "ï": car:="I";
      case "ô": car:="O";
      case "ö": car:="O";
      case "ù": car:="U";
      case "û": car:="U";
      case "ü": car:="U";
   }
   nomet:=nomet+CharToStr(car);
}
et:=create(ENTITY_TYPE,
    name:nomet,
    short_name:StrGetSubStr(nomet,0,3),
    @SCH_DATA:GetCurrentSchema());

listeParent:=SetTabDO(listeParent,1,et); /* Le parent de 1ère génération est l'entité. */
i:=2;
trouve:=0;
/* Les attributs (simples ou composés) de l'entité prennent leurs noms des TLabel, */
/* TGroupBox, TCheckBox et TRadioButton (on n'utilise que ceux-là).                 */
while trouve=0 and i<=max do
{
   temp:=GetTab2(liste,i,7);
   if (temp="TLabel" and GetTab2(liste,i,8)<>"-")
     or (temp="TGroupBox")
     or (temp="TCheckBox")
     or (temp="TRadioButton")
   then {trouve:=1;}
   else {i:=i+1;}
}
```

```
while i<=max do   /* Tant qu'on n'a pas ajouté tous les éléments de la liste d'entrée. */
{
  if Length(listeParent) > StrStoi(GetTab2(liste,i,2))
  /* Si l'élément courant est de niveau x et que la listeParent est de longueur */
  /* x+1, ce niveau x+1 représente un neveu de l'élément courant et n'est donc  */
  /* plus pertinent. On efface ce niveau x+1 (le dernier de la liste).          */
  then {
          attach e to listeParent;
          j:=1;
          while j <= Length(listeParent)-1 do
          {
             j:=j+1;
             e>>;
          }
          kill(e);
       }
  suiti:=i+1;
  trouve2:=0;
  /* On veut connaître l'élément suivant i de type TLabel, TGroupBox, TCheckBox, */
  /* TRadioButton pour ensuite tester le lien de parenté qu'il y a entre eux.    */
  while trouve2=0 and suiti<=max do
  {
     temp:=GetTab2(liste,suiti,7);
     if (temp="TLabel" and GetTab2(liste,suiti,8)<>"-")
       or (temp="TGroupBox")
       or (temp="TCheckBox")
       or (temp="TRadioButton")
     then {trouve2:=1;}
     else {suiti:=suiti+1;}
  }
  if suiti<=max  /* Si l'élément qui suit l'élément i existe (=est dans la liste)  */
                 /* alors l'élément i n'est pas le dernier de la liste.           */
  then {
          /* On initialise les variables contenant les cardinalités, le type et la */
          /* longueur du futur attribut correspondant à l'élément i.               */
 /* La cardinalité minimum */
      mincard:=StrStoi(GetTab2(liste,i,11));
          /* La cardinalité maximum */
      temp:=GetTab2(liste,i,12);
```

```
if temp="N" then {maxcard:=N_CARD;}
            else {maxcard:=StrStoi(temp);}
/* Le type et la longueur */
atype:=CHAR_ATT;
alength:=20;
temp:=GetTab2(liste,i,7);
if temp="TCheckBox" or temp="TRadioButton"
then {
        atype:=BOOL_ATT;
        alength:=1;
     }
    /* Les labels attachés à un TMemo deviennent des attributs facultatifs. */
temp:=GetTab2(liste,i,8);
if temp<>"-" and GetTab2(liste,StrStoi(temp),7)="TMemo"
then {
        mincard:=0;
        maxcard:=1;
     }
    /* Teste le rapport familiale entre l'élément i et suiti. */
    if GetTab2(liste,i,2)=GetTab2(liste,suiti,2)
    then {
            /* Si l'élément i et suiti sont frères. */
            niveau:=StrStoi(GetTab2(liste,i,2));
            pere:=GetTabDO(listeParent,niveau-1);
            /* Y-a-t'il déjà un attribut qui est de même niveau que le courant ? */
            if niveau<=Length(listeParent)
            then {  /* Oui -> on place le courant après son frère. */
                    frere:=GetTabDO(listeParent,niveau);
                    sia:=create(SI_ATTRIBUTE,
                                name:CleanStr(GetTab2(liste,i,1)),
                                min_rep:mincard,
                                max_rep:maxcard,
                                type:atype,
                                length:alength,
                                where:frere,
                                @OWNER_ATT:pere);
                    listeParent:=SetTabDO(listeParent,niveau,sia);
                 }
            else {  /* Non -> on le place en premier sous son père. */
                    sia:=create(SI_ATTRIBUTE,
```

```
                                name:CleanStr(GetTab2(liste,i,1)),
                                min_rep:mincard,
                                max_rep:maxcard,
                                type:atype,
                                length:alength,
                                @OWNER_ATT:pere);
                        listeParent:=SetTabDO(listeParent,niveau,sia);
                    }
            }
        else {
            if StrStoi(GetTab2(liste,i,2))<StrStoi(GetTab2(liste,suiti,2))
            then {
                    /* Si l'élément suivant est le fils du courant */
                    /* alors l'élément courant est composé.        */
                    niveau:=StrStoi(GetTab2(liste,i,2));
                    pere:=GetTabDO(listeParent,niveau-1);
                    /* Y-a-t'il déjà un attribut qui est de même niveau */
                    /* que le courant ?                                 */
                    if niveau<=Length(listeParent)
                    then {  /* Oui -> on place le courant après son frère. */
                            frere:=GetTabDO(listeParent,niveau);
                            coa:=create(CO_ATTRIBUTE,
                                        name:CleanStr(GetTab2(liste,i,1)),
                                        min_rep:mincard,
                                        max_rep:maxcard,
                                        where:frere,
                                        @OWNER_ATT:pere);
                            listeParent:=SetTabDO(listeParent,niveau,coa);
                         }
                    else {  /* Non -> on le place en premier sous son père. */
                            coa:=create(CO_ATTRIBUTE,
                                        name:CleanStr(GetTab2(liste,i,1)),
                                        min_rep:mincard,
                                        max_rep:maxcard,
                                        @OWNER_ATT:pere);
                            listeParent:=SetTabDO(listeParent,niveau,coa);
                         }
                }
            else {
                    /* L'élément qui suit est l'oncle du courant. */
```

```
                        niveau:=StrStoi(GetTab2(liste,i,2));
                        pere:=GetTabDO(listeParent,niveau-1);
                        /* Y-a-t'il déjà un attribut qui est de même niveau */
                        /* que le courant ?                                 */
                        if niveau<=Length(listeParent)
                        then {  /* Oui -> on place le courant après son frère. */
                                frere:=GetTabDO(listeParent,niveau);
                                sia:=create(SI_ATTRIBUTE,
                                            name:CleanStr(GetTab2(liste,i,1)),
                                            min_rep:mincard,
                                            max_rep:maxcard,
                                            type:atype,
                                            length:alength,
                                            where:frere,
                                            @OWNER_ATT:pere);
                                listeParent:=SetTabDO(listeParent,niveau,sia);
                             }
                        else {  /* Non -> on le place en premier sous son père. */
                                sia:=create(SI_ATTRIBUTE,
                                            name:CleanStr(GetTab2(liste,i,1)),
                                            min_rep:mincard,
                                            max_rep:maxcard,
                                            type:atype,
                                            length:alength,
                                            @OWNER_ATT:pere);
                                listeParent:=SetTabDO(listeParent,niveau,sia);
                             }
                    }


            }
    }
else { /* On doit encore traiter le dernier élément de la liste. */
       if i<=max
       then
       {
       /* On initialise les variables contenant les cardinalités, le type et la */
       /* longueur du futur attribut correspondant à l'élément i.                */
   /* La cardinalité minimum */
       mincard:=StrStoi(GetTab2(liste,i,11));
```

```
/* La cardinalité maximum */
temp:=GetTab2(liste,i,12);
if temp="N" then {maxcard:=N_CARD;}
           else {maxcard:=StrStoi(temp);}
/* Le type et la longueur */
atype:=CHAR_ATT;
alength:=20;
temp:=GetTab2(liste,i,7);
if temp="TCheckBox" or temp="TRadioButton"
then {
        atype:=BOOL_ATT;
        alength:=1;
     }

    niveau:=StrStoi(GetTab2(liste,i,2));
    pere:=GetTabDO(listeParent,niveau-1);
    c>>;
    /* Y-a-t'il déjà un attribut qui est de même niveau que le courant ? */
    if niveau<=Length(listeParent)
            then {  /* Oui -> on place le courant après son frère. */
                    frere:=GetTabDO(listeParent,niveau);
                    sia:=create(SI_ATTRIBUTE,
                                name:CleanStr(GetTab2(liste,i,1)),
                                min_rep:mincard,
                                max_rep:maxcard,
                                type:atype,
                                length:alength,
                                where:frere,
                                @OWNER_ATT:pere);
                    listeParent:=SetTabDO(listeParent,niveau,sia);
                 }
            else {  /* Non -> on le place en premier sous son père. */
                    sia:=create(SI_ATTRIBUTE,
                                name:CleanStr(GetTab2(liste,i,1)),
                                min_rep:mincard,
                                max_rep:maxcard,
                                type:atype,
                                length:alength,
                                @OWNER_ATT:pere);
                    listeParent:=SetTabDO(listeParent,niveau,sia);
```

```
                        }
                    }
                }
            /* L'élément qui doit être inséré, en attribut de l'entité, après l'élément i */
            /* est l'élément suiti.                                                        */

            i:=suiti;
        }
    }
}

/* ------------------------------------------------------------------------------- */
/* Lit le stream d'entrée jusqu'au premier blanc et renvoie la liste des caractères */
/* lus sous forme de string.                                                       */
/* ------------------------------------------------------------------------------- */
function string LireMot()
{
    return GetTokenUntil(" ");
}

/* ------------------------------------------------------------------------------- */
/* Retourne 1 si le point situé à pg pixels du bord gauche et à ph du haut est situé à */
/*              l'intérieur du rectangle rg,rd,rh,rb                               */
/*          0 sinon                                                                */
/* où rg pour coordonnée du côté gauche du rectangle                               */
/*    rd pour coordonnée du côté droit du rectangle                                */
/*    rh pour coordonnée du haut du rectangle                                      */
/*    rb pour coordonnée du bas du rectangle                                       */
/* ------------------------------------------------------------------------------- */
function integer IsIn(integer: pg
                     ,integer: ph
                     ,integer: rg
                     ,integer: rd
                     ,integer: rh
                     ,integer: rb)
{
      if (pg>=rg) and (pg<=rd) and (ph>=rh) and (ph<=rb)
      then {return 1;}
      else {return 0;}
}
```

```
/* ------------------------------------------------------------------------- */
/* Liste de la forme [left,top,width,height] où left, top, width, height sont des  */
/* coordonnées en pixels, relatives à la fenêtre, formant un rectangle.          */
/* Chaque liste correspond à un rectangle. La fonction retourne 1 si les listes se */
/* superposent au moins en un point et 0 sinon.                                  */
/* ------------------------------------------------------------------------- */
function integer Superpose(list: lista, list: listb)
  integer: al;
  integer: at;
  integer: aw;
  integer: ah;
  integer: bl;
  integer: bt;
  integer: bw;
  integer: bh;
   cursor: c;
{
      attach c to lista;
      al:=get(c); c>>;
      at:=get(c); c>>;
      aw:=get(c); c>>;
      ah:=get(c);
      attach c to listb;
      bl:=get(c); c>>;
      bt:=get(c); c>>;
      bw:=get(c); c>>;
      bh:=get(c);
      if (IsIn(al,   at   ,bl,bl+bw,bt,bt+bh)=1) or
         (IsIn(al   ,at+ah,bl,bl+bw,bt,bt+bh)=1) or
         (IsIn(al+aw,at   ,bl,bl+bw,bt,bt+bh)=1) or
         (IsIn(al+aw,at+ah,bl,bl+bw,bt,bt+bh)=1) or
         (IsIn(bl   ,bt   ,al,al+aw,at,at+ah)=1) or
         (IsIn(bl   ,bt+bh,al,al+aw,at,at+ah)=1) or
         (IsIn(bl+bw,bt   ,al,al+aw,at,at+ah)=1) or
         (IsIn(bl+bw,bt+bh,al,al+aw,at,at+ah)=1)
      then {return 1;}
      else {return 0;}
}
```

```
/* ------------------------------------------------------------------------------ */
/* Transforme un form enregistrer en format texte en une liste de listes (un tableau)  */
/* [0,l1,l2,l3, ..... ] où li 1<=i<= ... est de la forme                          */
/* [0,label,niveau,left,top,width,height,type,proche,colcount,nbliens,min_rep,max_rep] */
/* label=le caption de l'objet ou "-" s'il n'en a pas                             */
/* niveau=niveau hiérarchique de l'objet (1=la fenêtre principale)                */
/* left,top,whidth,height=les coordonnées de l'objet                              */
/* type=le type de l'objet                                                        */
/* proche=pour les labels uniquement, c'est l'indice de l'élément à qui il est attaché */
/* colcount=nombre de colonnes de l'élément                                       */
/* nbliens=nombre d'objets qui sont attachés à l'objet courant                    */
/* min_rep,max_rep=les cardinalités de l'élément                                  */
/* ------------------------------------------------------------------------------ */
function list Dfm2List()
  integer: i;
     list: liste;
   string: mot;
  integer: niveau;
{
  niveau:=0;
  i:=0;
  liste:=[];
  OuvreTxtStream(); /* On ouvre le fichier et on le définit comme stream d'entrée. */
  while nseof() do
  {
     SkiplstBlanc(); /* On passe les premiers blancs. */
     mot:=LireMot(); /* On lit le premier mot du stream. */
     switch(mot)
     {
        case "object":
                        niveau:=niveau+1; /* Si c'est un objet on augmente le niveau.            */
                        if i=0 then {     /* Si i=0 c'est qu'on commence la lecture et           */
                                          /* c'est donc la « form » générale qu'on lit maintenant */
                                    i:=i+1;
                                    liste:=SetTab2(liste,i,2,StrItos(niveau));
                                    liste:=SetTab2(liste,i,7,"TForm");
                                    liste:=SetTab2(liste,i,8,"-");
                                    liste:=SetTab2(liste,i,9,"1");
                                    liste:=SetTab2(liste,i,10,"0");
                                    liste:=SetTab2(liste,i,11,"1");
```

```
                        liste:=SetTab2(liste,i,12,"1");
                    }
                else
                {  /* On traite les autres objets. */
                   Skip1stBlanc();
                   mot:=LireMot();
                   liste:=SetTab2(liste,i+1,1,"-");
                   liste:=SetTab2(liste,i+1,2,StrItos(niveau));
                   liste:=SetTab2(liste,i+1,8,"-");
                   liste:=SetTab2(liste,i+1,9,"1");
                   liste:=SetTab2(liste,i+1,10,"0");
                   liste:=SetTab2(liste,i+1,11,"1");
                   liste:=SetTab2(liste,i+1,12,"1");
                }

/* Suivant le type de l'objet, on insère le type + parfois on modifie ses */
/* propriétés.                                                            */
case "TButton\n": i:=i+1;
                  liste:=SetTab2(liste,i,7,"TButton");

case "TBitBtn\n": i:=i+1;
                  liste:=SetTab2(liste,i,7,"TButton");

case "TLabel\n": i:=i+1;
                 liste:=SetTab2(liste,i,7,"TLabel");

case "TEdit\n": i:=i+1;
                liste:=SetTab2(liste,i,7,"TEdit");

case "TComboBox\n": i:=i+1;
                    liste:=SetTab2(liste,i,7,"TComboBox");

case "TCheckBox\n": i:=i+1;
                    liste:=SetTab2(liste,i,7,"TCheckBox");

case "TGroupBox\n": i:=i+1;
                    liste:=SetTab2(liste,i,7,"TGroupBox");

case "TMemo\n": i:=i+1;
                liste:=SetTab2(liste,i,7,"TMemo");
```

```
        case "TStringGrid\n": i:=i+1;
                              liste:=SetTab2(liste,i,7,"TStringGrid");
                              liste:=SetTab2(liste,i,9,"5");

        case "end\n":   niveau:=niveau-1; /* Si c'est end (\n car un end est toujours */
                                          /* en fin de ligne) alors c'est la fin      */
                                          /* d'un objet -> on décrémente le niveau.   */

        case "Caption": SkipUntil("'");
                        liste:=SetTab2(liste,i,1,GetTokenUntil("\n"));
        /* Si on a le mot Caption -> on lit jusque '. Là commence le mot qui servira */
        /* de nom d'entité ou d'attribut. On le stocke avec son niveau hiérarchique  */
        /* dans la liste qui sera renvoyée.                                          */

        case "Left": SkipUntil("0-9");
                mot:=LireMot();
                liste:=SetTab2(liste,i,3,StrGetSubStr(mot,0,(StrLength(mot)-1)));

        case "Top":  SkipUntil("0-9");
                mot:=LireMot();
                liste:=SetTab2(liste,i,4,StrGetSubStr(mot,0,(StrLength(mot)-1)));

        case "Width": SkipUntil("0-9");
                 mot:=LireMot();
                 liste:=SetTab2(liste,i,5,StrGetSubStr(mot,0,(StrLength(mot)-1)));

        case "Height": SkipUntil("0-9");
                  mot:=LireMot();
                  liste:=SetTab2(liste,i,6,StrGetSubStr(mot,0,(StrLength(mot)-1)));

        case "ColCount": SkipUntil("0-9");
                    mot:=LireMot();
                    liste:=SetTab2(liste,i,9,StrGetSubStr(mot,0,(StrLength(mot)-1)));
    }
  }
  return liste;
}

/* ------------------------------------------------------------------------------ */
```

```
/* Le but de cette fonction est de remplir la cellule 8 (proche) de chaque label avec */
/* l'indice de l'élément avec lequel le label est attaché.                            */
/* ---------------------------------------------------------------------------------- */
function list CrossTab1(list: tab)
  integer: h;
  integer: i;
  integer: j;
  integer: l;
  integer: longueur;
  integer: nbliens;
  integer: t;
  integer: trouve;
   string: typ;
  integer: w;

{
      longueur:=Length(tab)-1;
      i:=2;
      while i<=longueur do
        {
          if GetTab2(tab,i,7)="TLabel"
          /* On ne modifie que les cellules des labels. */
          then {
                  l:=StrStoi(GetTab2(tab,i,3));
                  t:=StrStoi(GetTab2(tab,i,4));
                  w:=StrStoi(GetTab2(tab,i,5));
                  h:=StrStoi(GetTab2(tab,i,6));
                  /* On prend ses coordonnées. */
                  j:=2;
                  trouve:=0;
                  while j<=longueur and trouve=0 do
                    {
                       nbliens:=StrStoi(GetTab2(tab,j,10));
                       typ:=GetTab2(tab,j,7);
                         /* Si on a un élément j de type Edit,ComboBox,Memo ou StringGrid */
                         /* de même niveau que l'élément i et qui est lié moins de fois   */
                         /* que son nombre de colonne ne le permet.                       */
                       if (   typ="TEdit"
                           or typ="TComboBox"
                           or typ="TMemo"
```

```
                        or typ="TStringGrid")
                            and GetTab2(tab,i,2)=GetTab2(tab,j,2)
                            and nbliens < StrStoi(GetTab2(tab,j,9))
                        then {  /* Alors c'est un candidat sérieux et on teste si il se */
                                /* superpose en au moins un point.                      */
                                if 1=Superpose([l,t,w,h],[StrStoi(GetTab2(tab,j,3))
                                                         ,StrStoi(GetTab2(tab,j,4))
                                                         ,StrStoi(GetTab2(tab,j,5))
                                                         ,StrStoi(GetTab2(tab,j,6))])
                                then {
                                      tab:=SetTab2(tab,i,8,StrItos(j));
                                      tab:=SetTab2(tab,j,10,StrItos(nbliens+1));
                                      trouve:=1;
                                     }
                                j:=j+1;
                             }
                        else {
                                j:=j+1;
                             }
                    }
                }
            i:=i+1;
        }
    return tab;
}

/* ------------------------------------------------------------------------------ */
/* Le but de la fonction est de modifier les coordonnées des éléments car celles-ci sont  */
/* relatives par rapport au père de chaque élément et on voudrait qu'elles le soient par  */
/* rapport à la fenêtre principale.                                                       */
/* ------------------------------------------------------------------------------ */
function list Align (list: tab)
  integer: i;
  integer: j;
  integer: l;
  integer: longueur;
  integer: n;
  integer: ok;
  integer: t;
{
```

```
    longueur:=Length(tab)-1;
    i:=3;
      while i<=longueur do /* Pour tous les éléments à partir du 3ième car le 1er */
                           /* est de niveau 1 et le 2ième de niveau 2.           */
      {
        n:=StrStoi(GetTab2(tab,i,2));
        l:=StrStoi(GetTab2(tab,i,3));
        t:=StrStoi(GetTab2(tab,i,4));
        if n>=3 /* Si le niveau de l'élément est supérieur ou égal à 3. */
        then {
               j:=i-1;
               ok:=0;
               /* Alors on cherche dans les éléments précédents son père pour */
               /* corriger les coordonnées.                                   */
               while j>=2 and ok=0 do
               {
                 if StrStoi(GetTab2(tab,j,2))=n-1
                 then {
                        tab:=SetTab2(tab,i,3,StrItos(StrStoi(GetTab2(tab,j,3))+l));
                        tab:=SetTab2(tab,i,4,StrItos(StrStoi(GetTab2(tab,j,4))+t));
                        ok:=1;
                      }
                 j:=j-1;
               }
             }
        i:=i+1;
      }
      return tab;
}

/* ------------------------------------------------------------------------------------------ */
/* Cette fonction teste le niveau de l'élément i du tableau liste par rapport aux éléments i+1 -> fin. */
/* Cette fonction renvoie 1 si tous les éléments qui se situent entre l'élément i et le premier       */
/* élément suivant i et de même niveau que l'élément i sont tous égaux au (niveau de i) + 1 .         */
/* ------------------------------------------------------------------------------------------ */
function integer AvantFeuille(list: liste,integer: i)
  integer: j;
  integer: fini;
  integer: max;
  integer: nivi;
```

```
  integer: nivj;
  integer: ok;
  integer: out;
{
   j:=i+1;
   max:=Length(liste)-1;
   nivi:=StrStoi(GetTab2(liste,i,2));
   fini:=0;
   ok:=0;
   if j<=max then {out:=0;}
             else {out:=1;}
   while out=0 and fini=0 do
   {
      nivj:=StrStoi(GetTab2(liste,j,2));
      if nivj=nivi then {
                              fini:=1;
                        }
                  else {
                          if nivj=nivi+1 then {ok:=1;}
                          if nivj>nivi+1 then {ok:=0;fini:=1;}
                        }
      j:=j+1;
      if j<=max then {out:=0;}
                else {out:=1;}
   }
   return ok;
}

/* ------------------------------------------------------------------------------ */
/* On règle le problème du père ayant des fils attachés à un même élément et ayant les */
/* mêmes cardinalités.                                                            */
/* ------------------------------------------------------------------------------ */
function list Cardinalities(list:tab)
  integer: change;
  integer: havelabel;
  integer: havesg;
  integer: i;
  integer: j;
  integer: max;
   string: maxcard;
```

```
 string: mincard;
integer: niveau;
integer: nivi;
integer: out;
 string: proche;
 string: temp;
integer: tempi;
 string: voisin;
{
    i:=2;
    max:=Length(tab)-1;
    /* Tous les éléments qui sont attachés à un StringGrid prennent [0-N] comme cardinalité. */
    while i<=max do
    {
            proche:=GetTab2(liste,i,8);
            if proche<>"-" then {
                        voisin:=(GetTab2(liste,StrStoi(proche),7));
                        if voisin="TStringGrid"
                        then {
                                tab:=SetTab2(tab,i,11,"0");
                                tab:=SetTab2(tab,i,12,"N");
                             }
                      }
            i:=i+1;
    }
    i:=2;
    /* Pour tous les éléments */
    while i<=max do
    {
       if AvantFeuille(tab,i)=1
       then { /* Si leur niveau est égal au niveau max - 1. */
           nivi:=StrStoi(GetTab2(tab,i,2));
            havesg:=0;
            havelabel:=0;
            change:=0;
            j:=i+1;
            /* Pour tous les fils (labels) de l'élément i, on regarde s'ils sont */
            /* attachés au même élément et si leurs cardinalités sont les mêmes. */
            if j<=max then {out:=0;}
                      else {out:=1;}
```

```
while out=0 do
{
   proche:=GetTab2(tab,j,8);
   if StrStoi(GetTab2(tab,j,2))=nivi+1
      and GetTab2(tab,j,7)="TLabel"
      and proche<>"-"
   then {
           mincard:=GetTab2(tab,j,11);
           maxcard:=GetTab2(tab,j,12);
           change:=1;
           out:=1;
        }
   else {
           if GetTab2(tab,j,7)="TStringGrid" then {havesg:=1;}
           if j+1>max or StrStoi(GetTab2(tab,j+1,2))<>nivi+1
           then {out:=1;}
           else {out:=0;j:=j+1;}
        }
}
/* Si change=1 c'est qu'on a au moins un fils de type Label attaché à */
/* un objet.                                                          */
if change=1 then {out:=0;}
/* On prend l'élément suivant. */
j:=j+1;
if j<=max then {
                  if StrStoi(GetTab2(tab,j,2))=nivi+1
                  then {out:=0;}
                  else {out:=1;}
               }
            else {out:=1;}
while out=0 do
{
   temp:=GetTab2(tab,j,7);
   if temp="TLabel" and GetTab2(tab,j,8)<>"-"
                    then {
                           if proche<>GetTab2(tab,j,8)
                              and mincard<>GetTab2(tab,j,11)
                              and maxcard<>GetTab2(tab,j,12)
                           then {change:=0;}
                           havelabel:=1;
```

```
                        }
   if temp="TStringGrid" then {
                                havesg:=1;
                            }
   j:=j+1;
   if j<=max then {
                    if StrStoi(GetTab2(tab,j,2))=nivi+1
                    then {out:=0;}
                    else {out:=1;}
                  }
            else {out:=1;}
}
/* Change était à 1 avant d'entrer dans le boucle du dessus. Change passe */
/* à 0 si un label, fils de i, attaché à un objet n'est pas attaché au     */
/* même objet ou n'a pas les mêmes cardinalités que le premier fils        */
/* (de type label et attaché à un objet) de i.                             */
if change=1 then {
                  tab:=SetTab2(tab,i,11,mincard);
                  tab:=SetTab2(tab,i,12,maxcard);
                  j:=i+1;
                  out:=0;
                  while out=0 do
                  {
                     if GetTab2(tab,j,7)="TLabel"
                     then {
                             tab:=SetTab2(tab,j,11,"1");
                             tab:=SetTab2(tab,j,12,"1");
                          }
                     j:=j+1;
                     if j<=max
                     then {
                             if StrStoi(GetTab2(tab,j,2))=nivi+1
                             then {
                                     out:=0;
                                  }
                             else {
                                     out:=1;
                                  }
                          }
                     else {
```

```
                                        out:=1;
                                    }
                                }
                            }
                /* Havesg=bool pour savoir si un TStringGrid est fils de i. */
                /* Havelabel=bool pour savoir s'il y a un label attaché à   */
                /* un objet dans les fils de i.                             */
                if havesg=1 and havelabel=0
                then {
                        tab:=SetTab2(tab,i,11,"0");
                        tab:=SetTab2(tab,i,12,"N");
                    }
            }
        i:=i+1;
    }
    return tab;
}

/* ^^^^^^^^^^^^^^^^^^^ */
/*                     */
/* Programme principal */
/*                     */
/* ^^^^^^^^^^^^^^^^^^^ */
begin
if IsVoid(GetCurrentSchema())
then {MessageBox ("ERROR","No schema available");}
else {
        liste:=Dfm2List();
        liste:=Align(liste);
        liste:=CrossTab1(liste);
        liste:=Cardinalities(liste);
        List2ET(liste);
    }
end
```

Annexe 2

Code du programme LogAnalyser

```
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */
/*                                     */
/* Déclaration des variables globales */
/*                                     */
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */

   file: fout;
integer: i;
integer: long;
 string: namefile;
   list: tab;

/* ^^^^^^^^^^^^^^^^^^^^^^^^^^ */
/*                            */
/* Déclaration des procédures */
/*                            */
/* ^^^^^^^^^^^^^^^^^^^^^^^^^^ */

/* ------------------------------------------------- */
/* Utilisé pour passer les blancs en début de ligne. */
/* ------------------------------------------------- */
procedure Skip1stBlanc()
{
    SkipWhile(" ");
}

/* ------------------------------------------------------------------------------- */
/* Sélectionne le fichier log à mettre dans le stream d'entrée, à l'aide d'une boîte */
/* à dialogue.                                                                     */
/* ------------------------------------------------------------------------------- */
function file OuvreLogFile()
  string: namefile;
{
    namefile:=BrowseRead("Open the LOG file","Files log (*.log)|*.LOG|All (*.*)|*.*","TXT");
    return OpenFile(namefile,_R);
}

/* ----------------------------------------------------------------- */
/* La liste de sortie est la liste d'entrée où l'élément s a été inséré  */
/* à l'emplacement i,j du tableau correspondant à la liste. Cela        */
```

```
/* permet d'utiliser l'avantage d'un tableau (index) avec ce qu'offre    */
/* voyager: les listes.                                                  */
/* i=indice de la ligne ; j=indice de la colonne                         */
/* --------------------------------------------------------------------- */
function list SetTab2(list: tabin, integer: i, integer: j, string: s)
  integer: a;
   cursor: c;
   cursor: d;
  integer: longueur;
     list: temp;
{

   /* Si le tableau est vide, on l'initialise. */
   if Length(tabin)=0
      then {
                  tabin:=[0];
           }
   longueur:=Length(tabin)-1;
   /* Si la ligne dans laquelle l'élément doit être inséré n'existe pas, on doit la   */
   /* créer (ainsi que toutes celles entre la dernière ligne du tableau actuel et la  */
   /* ligne (i) de l'élément à insérer.                                               */
   if i>longueur
      then {
              a:=1;
              attach c to tabin;
              while a<=longueur do {
                                     c>>;
                                  a:=a+1;
                                 }
              while a<i do {
                                    c+>[0];
                                    c>>;
                                    a:=a+1;
                                  }
             c+>[0];

           }
   /* Maintenant que le ligne i existe il faut se positionner sur cette ligne i. */
   attach c to tabin;
   c>>;
```

```
   a:=1;
   while a<i do {
                a:=a+1;
                c>>;
              }
   /* On est sur la bonne ligne, on positionne le curseur sur le 1er élément de */
   /* la ligne.                                                                  */
   attach d to get(c);
   longueur:=Length(get(c));
   a:=0;
   temp:=[];
   while ((a<j) and (a<longueur))
   do {
         temp:=temp++[get(d)];
         d>>;
         a:=a+1;
      }
   if a<j then {temp:=temp++[a..j-1];}
   temp:=temp++[s];
   /* On a copié dans temp le contenu de la ligne i de 0 à j. Il faut encore    */
   /* la fin de la ligne (j+1 -> la fin) si elle existe.                         */
   if a<longueur
   then {
           d>>;
           a:=a+1;
           while a<longueur do {
                                  temp:=temp++[get(d)];
                                  d>>;
                                  a:=a+1;
                               }
         }
   /* On insère la ligne temp à la place de la ligne i. */
   kill(c);
   c<<;
   c+>temp;
   return(tabin);
}

/* ---------------------------------------- */
/* Retourne l'élément i,j du tableau tabin. */
```

```
/* ---------------------------------------- */
function string GetTab2(list: tabin, integer: i, integer: j)
  integer: a;
   cursor: c;
{
   /* On se positionne sur la ligne i. */
   attach c to tabin;
   c>>;
   a:=1;
   while a<i do {
                  a:=a+1;
                  c>>;
                }
   /* Sur la ligne i on se positionne sur le j-ème élément. */
   attach c to get(c);
   c>>;
   a:=1;
   while a<j do {
                  a:=a+1;
                  c>>;
                }
   /* On le renvoie comme résultat. */
   return get(c);
}

/* ------------------------------------------------------------------------------ */
/* Retrouve l'indice de la première colonne i, du tableau 'liste' donné en paramètre */
/* dont la cellule de la ligne j est égale à la clé k (paramètres).               */
/* ------------------------------------------------------------------------------ */
function integer KeySearch(list: liste, integer: j, string: k)
 cursor: c;
 cursor: d;
integer: icol;
integer: iligne;
integer: ncol;
integer: trouve;
{
      attach c to liste;
      c>>;
      ncol:=Length(liste)-1;
```

```
      icol:=1;
      trouve:=0;
      while icol<=ncol and trouve=0 do
      {
         attach d to get(c);
         iligne:=1;
         d>>;
         while iligne<j do
         {
            d>>;
            iligne:=iligne+1;
         }
         if get(d)=k then {trouve:=1;}
                     else {c>>;icol:=icol+1;}
      }
      if trouve=1 then {return icol;}
                  else {return 0;}
}

/* ---------------------------------------- */
/* Lit une ligne d'un log généré par DB-MAIN. */
/* ---------------------------------------- -*/
procedure Read1Ligne(string: log)
string: ligne;
{
   ligne:=readf(log,_string);
   SetParser(ligne);
   SkipUntil("*%&"); /* Passe les blancs pour se positionner sur le contenu de la ligne */
                     /* (une ligne de log commence toujours par *, % ou & .            */
}

/* ------------------------------------------------------------------------------------ */
/* Analyse toutes les lignes du log de DB-MAIN pour essayer de retrouver des changements */
/* qui portent sur des objets du repository.                                            */
/* ------------------------------------------------------------------------------------ */
function list ScanLog(list: tab)
integer: col;
 string: command;
   char: devnull;
   file: log;
```

```
integer: longueur;
 string: namnew;
 string: namold;
integer: niveau;
 string: oidnew;
 string: oidold;
integer: old;
 string: ownnew;
 string: ownold;
integer: start;
 string: temp;
 string: temp2;
integer: trf;

{
   log:=OuvreLogFile();
   trf:=0;
   niveau:=0;
   old:=0;
   oidnew:="";
   oidold:="";
   namnew:="";
   namold:="";
   ownnew:="";
   ownold:="";
   while neof(log) do
   {
      Read1Ligne(log);
      temp:=(GetTokenUntil(" "));
      switch(temp) /* Le comportement varie suivant le premier mot lu sur une ligne du log. */
         {
            case "*TRF": trf:=1;
            case "*OLD": old:=2;
            case "*DEL": command:="*DEL";
            case "&DEL": command:="*DEL";
            case "*CRE": command:="*CRE";
            case "&CRE": command:="*CRE";
            case "*MOD": command:="*MOD";
            case "&MOD": command:="*MOD";
            case "%BEG": niveau:=niveau+1;
```

```
      case "%END": niveau:=niveau-1;
                   if old=2 then {old:=1;};
                   if niveau=0 and trf=1 then {trf:=0;temp:="";}
      case "%OID": Skip1stBlanc();if old=2 then {oidold:=GetTokenUntil("\n");}
                                           else {oidnew:=GetTokenUntil("\n");}
      case "%NAM": Skip1stBlanc();devnull:=GetChar();
                   if old=2 then {namold:=GetTokenUntil("\"\n");}
                            else {namnew:=GetTokenUntil("\"\n");}
      case "%OWN": Skip1stBlanc();temp2:=GetTokenUntil("\n");
                   if old=2
                   then {
                          start:=StrFindSubStr(temp2,1,"\" ")+3;
                          ownold:=StrGetSubStr(temp2,start,StrLength(temp2)-start);
                        }
                   else {
                          start:=StrFindSubStr(temp2,1,"\" ")+3;
                          ownnew:=StrGetSubStr(temp2,start,StrLength(temp2)-start);
                        }
   }
if (niveau=0 and temp="%END") or (niveau=1 and trf=1 and temp="%END")
then {
      if command="*DEL" then {
                                col:=KeySearch(tab, 4, oidnew);
                                tab:=SetTab2(tab,col,7,"OFF");
                                oidnew:="";
                                namnew:="";
                                ownnew:="";
                              }
      if command="*CRE" then {
                                longueur:=Length(tab)-1;
                                if longueur=-1 then {longueur:=0;}
                                tab:=SetTab2(tab,longueur+1,1,oidnew);
                                tab:=SetTab2(tab,longueur+1,2,namnew);
                                tab:=SetTab2(tab,longueur+1,3,ownnew);
                                tab:=SetTab2(tab,longueur+1,4,oidnew);
                                tab:=SetTab2(tab,longueur+1,5,namnew);
                                tab:=SetTab2(tab,longueur+1,6,ownnew);
                                tab:=SetTab2(tab,longueur+1,7,"ON");
                                oidnew:="";
                                namnew:="";
```

```
                                    ownnew:="";
                                }
            if command="*MOD" then {
                                    col:=KeySearch(tab, 4, oidold);
                                    if oidold<>oidnew and oidnew<>""
                                       then {tab:=SetTab2(tab,col,1,oidnew);}
                                    if namold<>namnew and namnew<>""
                                       then {tab:=SetTab2(tab,col,2,namnew);}
                                    if ownold<>ownnew and ownnew<>""
                                       then {tab:=SetTab2(tab,col,3,ownnew);}
                                    old:=0;
                                    oidnew:="";
                                    oidold:="";
                                    namnew:="";
                                    namold:="";
                                    ownnew:="";
                                    ownold:="";
                                }
            }
   }
   CloseFile(log);
   return tab;
}

/* ************************************************************ */
/* Analyse le fichier d'intégration créé par la fonction Transfo */
/* ************************************************************ */
function list ScanLog2(list: tab)
integer: col;
integer: crochet;
   char: devnull;
   file: fdest;
integer: i;
 string: inflow;
integer: longueur;
 string: master;
   list: masteratt
 string: masterid;
 string: slave;
   list: slaveatt;
```

```
 string: slaveid;
   list: superatt;
 string: supertype;
 string: supertypeid;
 string: t1;
 string: t2;
 string: t3;
 string: t4;
 string: temp;
integer: tt2;
integer: typ;

/* ------------------------------------------------------------------------ */
/* Format de masteratt, slaveatt et superatt (liste de liste = tableau). */
/*                                                                          */
/* 1 : nom1                                                                 */
/* 2 : id1                                                                  */
/* 3 : nom2                                                                 */
/* 4 : id2                                                                  */
/* 5 : bit                                                                  */
/* ------------------------------------------------------------------------ */
{
   /* Ouverture du fichier en lecture */
   fdest:=OpenFile("c:\\tdest.dbm",_R);
   while neof(fdest) do
   {
      masteratt:=[];
      slaveatt:=[];
      superatt:=[];
      inflow:=readf(fdest,_string);
      inflow:=readf(fdest,_string);
      /* Type : 2 */
      inflow:=readf(fdest,_string);
      SetParser(inflow);
      SkipUntil(":");
      devnull:=GetChar();
      typ:=StrStoi(GetTokenUntil("\n"));
      /* Master : ENTITY 1 */
      inflow:=readf(fdest,_string);
      SetParser(inflow);
```

```
SkipUntil(":");
devnull:=GetChar();
devnull:=GetChar();
master:=GetTokenUntil("\n");
/* MasterId : 35 */
inflow:=readf(fdest,_string);
SetParser(inflow);
SkipUntil(":");
devnull:=GetChar();
devnull:=GetChar();
masterid:=GetTokenUntil("\n");
/* b [ a x 323 ] ou x b a  */
i:=1;
crochet:=0;
inflow:=readf(fdest,_string);
SetParser(inflow);

while nseof() do
{
   temp:=GetTokenUntil(" ");
   devnull:=GetChar();
   if temp="[" then {
                       crochet:=1;
                       temp:=GetTokenUntil(" ");
                    }
   if crochet=0 then {
                        masteratt:=SetTab2(masteratt,i,1,temp);
                        masteratt:=SetTab2(masteratt,i,3,"-");
                        masteratt:=SetTab2(masteratt,i,5,"-");
                        i:=i+1;
                      }
                else {
                        masteratt:=SetTab2(masteratt,i,1,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        masteratt:=SetTab2(masteratt,i,3,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        masteratt:=SetTab2(masteratt,i,5,temp);
                        devnull:=GetChar();
```

```
                        devnull:=GetChar();
                        devnull:=GetChar();
                        crochet:=0;
                        i:=i+1;
                    }
}
/* 41 [ 39 43 323 ] ou 41 39  */
i:=1;

crochet:=0;
inflow:=readf(fdest,_string);
SetParser(inflow);
while nseof() do
{
   temp:=GetTokenUntil(" ");
   devnull:=GetChar();
   if temp="[" then {
                        crochet:=1;
                        temp:=GetTokenUntil(" ");
                    }
   if crochet=0 then {
                        masteratt:=SetTab2(masteratt,i,2,temp);
                        masteratt:=SetTab2(masteratt,i,4,"-");
                        i:=i+1;
                    }
               else {
                        masteratt:=SetTab2(masteratt,i,2,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        masteratt:=SetTab2(masteratt,i,4,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        devnull:=GetChar();
                        devnull:=GetChar();
                        devnull:=GetChar();
                        crochet:=0;
                        i:=i+1;
                    }
}
/* Slave : ENTITY 2 */
```

```
inflow:=readf(fdest,_string);
SetParser(inflow);
SkipUntil(":");
devnull:=GetChar();
devnull:=GetChar();
slave:=GetTokenUntil("\n");
/* SlaveId : 37 */
inflow:=readf(fdest,_string);
SetParser(inflow);
SkipUntil(":");
devnull:=GetChar();
devnull:=GetChar();
slaveid:=GetTokenUntil("\n");
/* y */
i:=1;
crochet:=0;
inflow:=readf(fdest,_string);
SetParser(inflow);
while nseof() do
{
   temp:=GetTokenUntil(" ");
   devnull:=GetChar();
   if temp="[" then {
                       crochet:=1;
                       temp:=GetTokenUntil(" ");
                    }
   if crochet=0 then {
                        slaveatt:=SetTab2(slaveatt,i,1,temp);
                        slaveatt:=SetTab2(slaveatt,i,3,"-");
                        slaveatt:=SetTab2(slaveatt,i,5,"-");
                        i:=i+1;
                     }
                else {
                        slaveatt:=SetTab2(slaveatt,i,1,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        slaveatt:=SetTab2(slaveatt,i,3,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
```

```
                        slaveatt:=SetTab2(slaveatt,i,5,temp);
                        devnull:=GetChar();
                        devnull:=GetChar();
                        devnull:=GetChar();
                        crochet:=0;
                        i:=i+1;
                    }
}
/* 45  */

i:=1;
crochet:=0;
inflow:=readf(fdest,_string);
SetParser(inflow);
while nseof() do
{
   temp:=GetTokenUntil(" ");
   devnull:=GetChar();
   if temp="[" then {
                       crochet:=1;
                       temp:=GetTokenUntil(" ");
                    }
   if crochet=0 then {
                        slaveatt:=SetTab2(slaveatt,i,2,temp);
                        slaveatt:=SetTab2(slaveatt,i,4,"-");
                        i:=i+1;
                    }
                else {
                        slaveatt:=SetTab2(slaveatt,i,2,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        slaveatt:=SetTab2(slaveatt,i,4,temp);
                        devnull:=GetChar();
                        temp:=GetTokenUntil(" ");
                        devnull:=GetChar();
                        devnull:=GetChar();
                        devnull:=GetChar();
                        crochet:=0;
                        i:=i+1;
                    }
```

```
}

/* Supertype : E1_E2
   [ x g 641 ] [ 73 51 641 ] [ d w 323 ] [ 45 71 323 ]   */
if typ=6 then
                {
                    inflow:=readf(fdest,_string);
                    SetParser(inflow);
                    SkipUntil(":");
                    devnull:=GetChar();
                    devnull:=GetChar();
                    supertype:=GetTokenUntil("\n");
                    inflow:=readf(fdest,_string);
                    SetParser(inflow);
                    i:=1;
                    crochet:=0;
                    while nseof() do
                    {
                       temp:=GetTokenUntil(" ");
                       devnull:=GetChar();
                       if temp="[" then {
                                          crochet:=crochet+1;
                                          temp:=GetTokenUntil(" ");
                                        }
                       if crochet=1 then {
                                           superatt:=SetTab2(superatt,i,1,temp);
                                           devnull:=GetChar();
                                           temp:=GetTokenUntil(" ");
                                           superatt:=SetTab2(superatt,i,3,temp);
                                           temp:=GetTokenUntil("]");
                                           devnull:=GetChar();
                                           devnull:=GetChar();
                                         }
                       if crochet=2 then {
                                           superatt:=SetTab2(superatt,i,2,temp);
                                           devnull:=GetChar();
                                           temp:=GetTokenUntil(" ");
                                           superatt:=SetTab2(superatt,i,4,temp);
                                           temp:=GetTokenUntil("]");
                                           devnull:=GetChar();
```

```
                                        crochet:=0;
                                        i:=i+1;
                                    }
                }
            }
/* Récupération de typ,master,masterid,masteratt,slave,slaveid,slaveatt pour les intégrer dans */
/* tab.                                                                                        */

if typ=2 then
        {
          /* On traite la partie master. */
          i:=1;
          longueur:=Length(masteratt)-1;
          while i<=longueur do
          {
           /*####################
             19:47
             Type : 2
             Master : E1
             MasterId : 35
             k j i h g f e d c [ s b 641 ] a
             59 57 55 53 51 49 47 45 43 [ 63 41 641 ] 39
             Slave : E2
             SlaveId : 37
             z y x w v u t
             77 75 73 71 69 67 65 */

             t1:=GetTab2(masteratt,i,1);
             t2:=GetTab2(masteratt,i,2);
             t3:=GetTab2(masteratt,i,3);
             t4:=GetTab2(masteratt,i,4);
             col:=KeySearch(tab,4,t2);
             tab:=SetTab2(tab,col,3,masterid);
             if t3 <>"-"
             then
             {
                col:=KeySearch(tab,4,t4);
                tab:=SetTab2(tab,col,3,masterid);
                tab:=SetTab2(tab,col,2,t1);
                tab:=SetTab2(tab,col,1,t2);
```

```
                }
                i:=i+1;
            }
            /* On traite la partie slave. */
            i:=1;
            longueur:=Length(slaveatt)-1;
            while i<=longueur do
            {
                t1:=GetTab2(slaveatt,i,1);
                t2:=GetTab2(slaveatt,i,2);
                t3:=GetTab2(slaveatt,i,3);
                t4:=GetTab2(slaveatt,i,4);
                col:=KeySearch(tab,4,t2);
                tab:=SetTab2(tab,col,3,slaveid);
                if t3 <>"-"
                then
                {
                    col:=KeySearch(tab,4,t4);
                    tab:=SetTab2(tab,col,3,slaveid);
                    tab:=SetTab2(tab,col,2,t1);
                    tab:=SetTab2(tab,col,1,t2);
                }
                i:=i+1;
            }
        }

if typ=6 then /* [ t  ] [ a  ]
                 [ 57 ] [ 39 ]
                 [ b  ] [ r  ]
                 [ 41 ] [ 53 ]
                 */
        {
          /* Supertype : A1A2
             [ d  45 w 71 323 ]  */

            i:=1;
            longueur:=Length(superatt)-1;
            col:=KeySearch(tab,5,supertype);
            supertypeid:=GetTab2(tab,col,4);
            while i<=longueur do
```

```
                {
                    t1:=GetTab2(superatt,i,1);
                    t2:=GetTab2(superatt,i,2);
                    t3:=GetTab2(superatt,i,3);
                    t4:=GetTab2(superatt,i,4);

                    col:=KeySearch(tab,4,t2);
                    tab:=SetTab2(tab,col,3,supertypeid);
                    col:=KeySearch(tab,4,t4);
                    tab:=SetTab2(tab,col,3,supertypeid);
                    tab:=SetTab2(tab,col,2,t1);
                    tab:=SetTab2(tab,col,1,t2);

                    i:=i+1;
                }
            }
    }
    CloseFile(fdest);
    return tab;
}

/* ^^^^^^^^^^^^^^^^^^^ */
/*                     */
/* Programme principal */
/*                     */
/* ^^^^^^^^^^^^^^^^^^^ */

begin
        tab:=[];
        tab:=ScanLog(tab);
        tab:=ScanLog2(tab);
        namefile:=BrowsePrint("Save the tab file","Files TAB (*.txt)|*.TXT|All (*.*)|*.*","TXT");
        print("\nParsing fini, début de l'écriture du tab dans un fichier");
        fout:=OpenFile(namefile,_W);
        long:=Length(tab)-1;
        i:=1;

        while i<=long do
        {
            printf(fout,GetTab2(tab,i,1));
```

```
            printf(fout,"|");
            printf(fout,GetTab2(tab,i,2));
            printf(fout,"|");
            printf(fout,GetTab2(tab,i,3));
            printf(fout,"\n");
            printf(fout,GetTab2(tab,i,4));
            printf(fout,"|");
            printf(fout,GetTab2(tab,i,5));
            printf(fout,"|");
            printf(fout,GetTab2(tab,i,6));
            printf(fout,"|");
            printf(fout,GetTab2(tab,i,7));
            printf(fout,"\n\n");
            i:=i+1;
        }
        CloseFile(fout);
        print("\nFini");
end
```

Annexe 3

Modifications apportées au programme Transfo.v2

```
/*****************************************************************************/
/* Integration of entity types, rel-types or compound attributes             */
/*****************************************************************************/

export function integer PRE_integrate_dto(list : l)
/*
@NAME:   PRE_integrate_dto
@PARAM:  list *l : list that contains the following elements
                   1. Integer : type of the integration. The value must be :
                              1 = copy slave into master
                              2 = merge slave into master
                              3 = create a 1-1 link
                              4 = slave is-a master
                              5 = master is-a slave
                              6 = create common supertype
                   2. Data-object : master
                   3. List : list of master elements
                             If same is selected, the element has the following
                             structure : [generic-object selected,generic-object
                             rejected, status] else the element is a
                             generic-object (attributs, processing-units, roles
                             and sub-types).
                   4. Data-object : slave
                   5. List : list of slave generic-objects (attributs,
                             processing-units, roles and sub-types)
                   6. string : name of the common super-type
                   7. List : list of supertype elements
                             Each element is a list with the structure
                             [generic-object selected,generic-object rejected,
                             status]
         where status has the following meaning :
          bit 1 : not use
          bit 2 : 1 = master selected, 0 = slave selected
          bit 3 : 1 = name the non selected object, 0 = name the selected object
          bit 4 : 1 = shortname the non selected object, 0 = shortname the selected object
          bit 5 : 1 = cardinality the non selected object, 0 = cardinality the selected object
          bit 6 : 1 = type of non selected object, 0 = type of no selected object
          bit 7 : 1 = sem. of master selected, 0 = sem. of master non selected
          bit 8 : 1 = sem. of slave selected, 0 = sem. of slave non selected
          bit 9 : 1 = tech. of master selected, 0 = tech. of master non selected
```

```
        bit 10: 1 = tech. of slave selected, 0 = tech. of slave non selected
@RESULT: integer : return FALSE if there are problems
                   return TRUE otherwise
@DESCR:  This function is called before the integration of data-objects (entity
         type, rel-type or compound attribute).
*/
integer:     t,t1,i;
data_object: master,slave;
string:      super,sortie;
list:        lmast,lsla,lsup;
cursor:      c,c1,c2;
data_object: dto;
sub_type:    sub;
role:        rol;
proc_unit:   proc;
file:        fdest;
{
  /*
    Création d'un fichier de destinnation.
  */
  fdest:=OpenFile("c:\\tdest.dbm",_A);

  printf(fdest,"####################\n");
  printf(fdest,[GetHour(),":",GetMin(),"\n"]);
  attach c to l;
  t := get(c);printf(fdest,["Type : ",t,"\n"]);
  c>>;
  master := get(c);printf(fdest,["Master : ",master.name,"\n"]);
                   printf(fdest,["MasterId : ",GetOID(master),"\n"]);
  c>>;
  sortie:="";
  if (GetType(get(c)) = _list) then {
    lmast := get(c);
    attach c1 to lmast;
    while IsNoVoid(c1) do {
      if (GetType(get(c1)) = _list) then {
        l := get(c1);
        printf(fdest,"[ ");
        sortie:=sortie+"[ ";
        attach c2 to l;
```

```
      while IsNoVoid(c2) do {
        t1 := GetType(get(c2));
        switch(t1) {
          case CO_ATTRIBUTE: dto := get(c2); printf(fdest,[dto.name," "]);
                                             sortie:=sortie+StrItos(GetOID(dto))+" ";
          case SI_ATTRIBUTE: dto := get(c2); printf(fdest,[dto.name," "]);
                                             sortie:=sortie+StrItos(GetOID(dto))+" ";
          case SUB_TYPE:     sub := get(c2); printf(fdest,[sub.name," "]);
                                             sortie:=sortie+StrItos(GetOID(sub))+" ";
          case ROLE:         rol := get(c2); printf(fdest,[rol.name," "]);
                                             sortie:=sortie+StrItos(GetOID(rol))+" ";
          case PROC_UNIT:    proc := get(c2);printf(fdest,[proc.name," "]);
                                             sortie:=sortie+StrItos(GetOID(proc))+" ";
          otherwise:         printf(fdest,[get(c2)," "]);
                             sortie:=sortie+StrItos(get(c2))+" ";
        }
        c2>>;
      }
      printf(fdest,"] "); sortie:=sortie+"] ";
    }
    else {
      t1 := GetType(get(c1));
      switch(t1) {
        case CO_ATTRIBUTE: dto := get(c1); printf(fdest,[dto.name," "]) ;sortie:=sortie+StrItos(GetOID(dto))+" ";
        case SI_ATTRIBUTE: dto := get(c1); printf(fdest,[dto.name," "]) ;sortie:=sortie+StrItos(GetOID(dto))+" ";
        case SUB_TYPE:     sub := get(c1); printf(fdest,[sub.name," "]) ;sortie:=sortie+StrItos(GetOID(sub))+" ";
        case ROLE:         rol := get(c1); printf(fdest,[rol.name," "]) ;sortie:=sortie+StrItos(GetOID(rol))+" ";
        case PROC_UNIT:    proc := get(c1);printf(fdest,[proc.name," "]);sortie:=sortie+StrItos(GetOID(proc))+" ";
      }
     }
    c1>>;
  }
  printf(fdest,"\n"); printf(fdest,[sortie,"\n"]);sortie:="";
  c>>;
}
slave := get(c);printf(fdest,["Slave : ",slave.name,"\n"]);
                printf(fdest,["SlaveId : ",GetOID(slave),"\n"]);
c>>;
if (IsNoVoid(c) and GetType(get(c)) = _list) then {
  lsla := get(c);
```

```
  attach c1 to lsla;
  while IsNoVoid(c1) do {
    t1 := GetType(get(c1));
    switch(t1) {
      case CO_ATTRIBUTE: dto := get(c1); printf(fdest,[dto.name," "]); sortie:=sortie+StrItos(GetOID(dto))+" ";
      case SI_ATTRIBUTE: dto := get(c1); printf(fdest,[dto.name," "]); sortie:=sortie+StrItos(GetOID(dto))+" ";
      case SUB_TYPE:     sub := get(c1); printf(fdest,[sub.name," "]); sortie:=sortie+StrItos(GetOID(sub))+" ";
      case ROLE:         rol := get(c1); printf(fdest,[rol.name," "]); sortie:=sortie+StrItos(GetOID(rol))+" ";
      case PROC_UNIT:    proc := get(c1);printf(fdest,[proc.name," "]);sortie:=sortie+StrItos(GetOID(proc))+" ";
    }
    c1>>;
  }
  printf(fdest,"\n"); printf(fdest,[sortie,"\n"]);sortie:="";
  c>>;
}
if IsNoVoid(c) then
{
  if GetType(get(c)) = _string then
  {
     super := get(c);printf(fdest,["Supertype : ",super,"\n"]);
     c>>;
  }
}
if IsNoVoid(c) then
{
   if GetType(get(c)) = _string then
   {
      super := get(c);printf(fdest,["Supertype : ",super,"\n"]);
     c>>;
   }
}
if IsNoVoid(c) then
{
   if GetType(get(c)) = _list then
   {
      lsup := get(c);
      attach c1 to lsup;
      while IsNoVoid(c1) do {
        if (GetType(get(c1)) = _list) then {
          printf(fdest,"[ ");sortie:=sortie+"[ ";
```

```
            l := get(c1);
            attach c2 to l;
            while IsNoVoid(c2) do {
              t1 := GetType(get(c2));
              switch(t1) {
                case CO_ATTRIBUTE: dto := get(c2);
                                   printf(fdest,[dto.name," "]);
                                   sortie:=sortie+StrItos(GetOID(dto))+" ";
                case SI_ATTRIBUTE: dto := get(c2);
                                   printf(fdest,[dto.name," "]);
                                   sortie:=sortie+StrItos(GetOID(dto))+" ";
                case SUB_TYPE:     sub := get(c2);
                                   printf(fdest,[sub.name," "]);
                                   sortie:=sortie+StrItos(GetOID(sub))+" ";
                case ROLE:         rol := get(c2);
                                   printf(fdest,[rol.name," "]);
                                   sortie:=sortie+StrItos(GetOID(rol))+" ";
                case PROC_UNIT:    proc := get(c2);
                                   printf(fdest,[proc.name," "]);
                                   sortie:=sortie+StrItos(GetOID(proc))+" ";
                otherwise:         printf(fdest,[get(c2)," "]);
                                   sortie:=sortie+StrItos(get(c2))+" ";
              }
              c2>>;
            }
            printf(fdest,"] ");sortie:=sortie+"] ";printf(fdest,sortie);sortie:="";
          }
          c1>>;
        }
        printf(fdest,"\n");
      }
    }
    CloseFile(fdest);
    return TRUE;
}

export function integer POST_integrate_dto(list : l)
/*
@NAME:   POST_integrate_dto
@PARAM:  list *l : list that contains a data-object.
```

```
@RESULT: integer : return FALSE if there are problems
                   return TRUE otherwise
@DESCR:  This function is called after the integration of data-objects (entity
         type, rel-type or compound attribute). The master, slave (if not
         deleted) and super-type (if create a common supertype strategy is
         choosen) data-objects are in l.
*/
data_object: dto;
cursor:    c;
{
  attach c to l;
  while IsNoVoid(c) do {
    dto := get(c);

    c>>;
  }
  return TRUE;
}
```