MASTER EN SCIENCES INFORMATIQUES

Les outils de quatrième génération
Concepts et enquête

Henneaux, Thierry; Smets, Pierre

Award date:
1987

Awarding institution:
Universite de Namur

LES OUTILS
DE QUATRIEME GENERATION

CONCEPTS ET ENQUETE

Mémoire présenté en vue de l'obtention
du diplôme de licencié et maître en informatique
par **Thierry Henneaux et Pierre Smets**

Année académique 1986 - 1987

REMERCIEMENTS

C'est un grand plaisir pour nous de remercier Monsieur Jean-Luc Hainaut, professeur à l'Institut d'Informatique, qui a véritablement été notre guide tout au long de l'élaboration de cette étude, et qui nous a prodigué de nombreux conseils.

Toute notre gratitude s'adresse également à Madame Jumet et à Messieurs Clajot, Vermeire, Van Houtte, Bustyn, Trieststst, Danhiez, Saint-Amand et Moentack qui ont répondu à notre questionnaire avec beaucoup de compétence.

TABLE DES MATIERES

PARTIE 1 : CONCEPTS

INTRODUCTION

L'objectif de toute organisation est de répondre à certains besoins de la société en respectant des contraintes de performance. Dès lors, les organisations ont sans cesse essayé d'améliorer la productivité à tous les niveaux. L'informatique a contribué très certainement à améliorer cette productivité, mais très vite, elle a elle-même subi des évolutions successives destinées à améliorer son propre rendement ... C'est dans cet esprit que, voici quelques années sont apparus les outils de quatrième génération...

La première partie de ce travail a pour but de déterminer l'environnement dans lequel les outils de quatrième génération évoluent. Cette approche est essentiellement théorique et se découpe en trois chapitres.

Le premier chapitre décrit le contexte actuel en matière de développement d'applications. Les étapes du cycle de vie d'une application et les principaux problèmes rencontrés dans le contexte actuel sont exposés. Une section sur l'informatique décisionnelle clôture ce chapitre.

Le deuxième chapitre décrit la philosophie des outils de quatrième génération. Ceux-ci sont le confluent de plusieurs domaines que nous décrirons succintement :

- les bases de données;
- la micro-informatique;
- les réseaux;
- la bureautique;
- les ateliers logiciels;
- les infocentres.

Ensuite, les outils de quatrième génération sont étudiés d'une part dans un cadre opérationnel, et d'autre part dans un cadre décisionnel.

Les problèmes suscités par une "approche quatrième génération" sont examinés au chapitre 3. Les réticences psychologiques, la formation, l'intégration dans un environnement existant, les performances, la maintenance, la conception des applications sont les principales contraintes imposées par un outil de quatrième génération.

Enfin, la dernière section de ce chapitre a pour objet de guider le choix d'un L4G (Langage de 4ème Génération).

CHAPITRE PREMIER

CONTEXTE ET PROBLEMES

Section 1.1 : Evolution du matériel et du logiciel

A l'aube de l'informatique, le prix du matériel s'avérait tellement élevé que les logiciels qui accompagnaient ce matériel étaient livrés gratuitement.

Depuis lors, le prix du matériel n'a cessé de chuter. L'on enregistre une diminution de 30 à 40 % du prix du matériel tous les deux ans. Et cette chute de prix est prévue jusqu'en 1990...

D'autre part, les performances du matériel informatique se sont accrue d'année en année notamment par l'augmentation du nombre d'instructions traitées par seconde et par l'accroissement de la capacité de stockage des informations. Remarquons que le marché n'enregistre pas toujours les baisses de prix annoncées plus haut. En effet, bien souvent les constructeurs fournissent un matériel plus puissant, à un prix équivalent à un matériel moins puissant déjà disponible sur le marché.

Hélas, le prix du logiciel subit le phénomène inverse que l'on peut expliquer en deux points :

- le coût du logiciel est très lié aux salaires;
- certains logiciels sont complexes et nécessitent une main-d'oeuvre spécialisée.

A titre d'exemple [MART 83], aux Etats Unis, le coût global moyen d'une instruction de programmation testée est d'environ dix dollars. Une étude du Ministère américain de la Défense a montré que les frais de développement pour les applications avioniques de l'Air Force étaient d'environ 75 dollars par instruction; de plus, le coût de la maintenance des applications se chiffre à 4000 dollars par instruction.

Le figure 1.1 de la page 6 montre la variation des coûts du temps d'ordinateur et la variation des coûts de la main-d'oeuvre.

Une autre tendance dans l'évolution du matériel et du logiciel est l'importance démesurée accordée au matériel par rapport au logiciel. Ce facteur s'explique par des raisons à la fois historique et humaine. Il y a dix ans, il s'agissait d'abord d'acheter du bon matériel et puis ensuite de développer des applications basées avant tout sur l'aspect quantitatif : traitement de comptabilité, état des stocks, paiement des salaires. De plus, les responsables du service informatique, issus des rangs techniques, influençaient dans une certaine mesure le déséquilibre matériel-logiciel qui pouvait se chiffrer à deux tiers d'investissement consacrés au matériel et un tiers au logiciel. A l'heure actuelle, au fur et à mesure

que l'informatique gagne tous les aspects qualitatifs de l'entreprise (gestion, stratégies commerciale et financière), la balance précédente matériel-logiciel devrait être complètement inversée. Les dirigeants optent encore pour la solution bien concrète qui consiste à acheter sans cesse du matériel plus performant plutôt que du logiciel "invisible" de plus en plus coûteux; mais alors ils ne maîtrisent pas toujours les décisions à prendre...

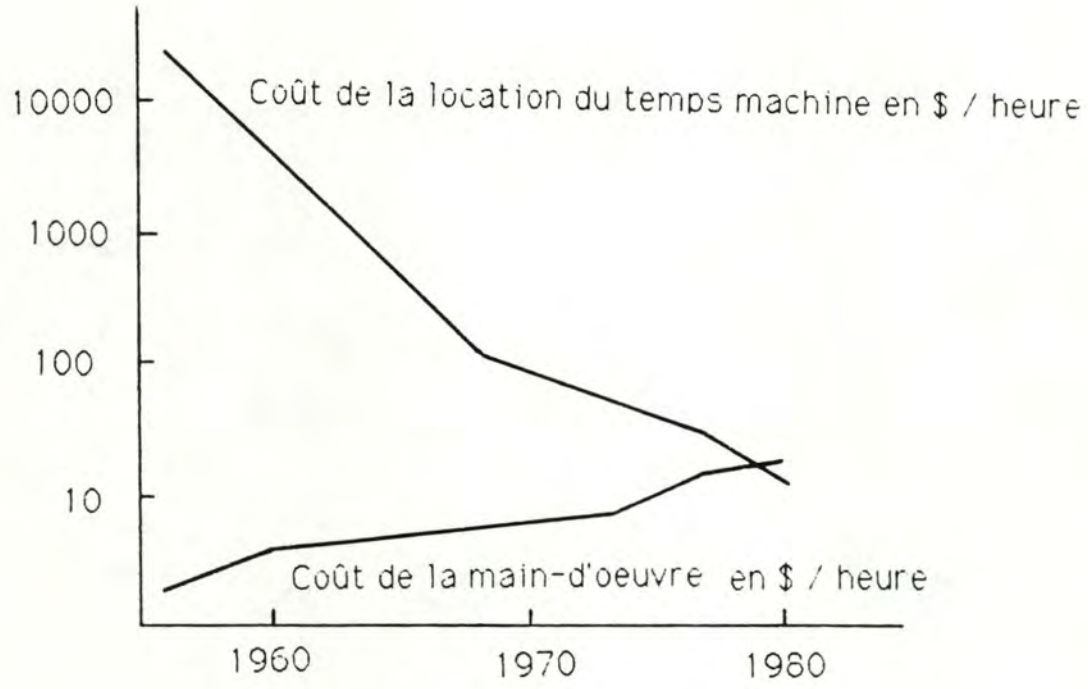


Figure I.1 : variation des coûts du temps d'ordinateur et des coûts de la main-d'oeuvre [MART 83].

Section 1.2 : L'informatique opérationnelle

1.2.1 Types d'applications concernées

Les applications informatiques envisagées dans le cadre de ce mémoire entrent dans le domaine de l'informatique de gestion et peuvent se décomposer en trois parties :

1) les applications dites **lourdes** : il s'agit d'applications stables qui nécessitent une planification au point de vue des ressources (humaine et technique).
Parmi ces applications, l'on peut citer : la comptabilité générale ou analytique, le paiement des salaires,...

2) les applications dites **légères** : il s'agit d'applications ponctuelles, non planifiées, urgentes qui émanent souvent de l'utilisateur final.
Exemples : consultation des clients douteux, récapitulatif des clients par région,...

3) les applications dites **moyennes** : se trouvent à mi-chemin entre les applications "lourdes" et "légères".

Les frontières entre ces trois définitions sont quelque peu floues et seront établies différemment d'une personne à l'autre. Mais il est évident que l'identification d'une application à une de ces trois parties se base sur la complexité du développement, le temps de développement de l'application, les ressources utilisées,....

Des applications comme l'écriture d'un compilateur ou l'automatisation d'un processus industriel n'interviennent pas dans ce contexte.

1.2.2 Le cycle de vie d'une application informatique

Le cycle de vie d'une application "lourde" ou "moyenne" se décompose en **sept** parties :

- l'analyse des besoins ou analyse d'opportunité
- la spécification fonctionnelle
- la conception
- le codage
- les tests
- l'exploitation
- la maintenance

Alors que cette découpe devient une norme pour mener à bien une application informatique, il existe par contre différentes méthodes pour réaliser certaines parties de cette découpe (notamment l'analyse des besoins et surtout la spécification fonctionnelle). Etant donné le caractère trop spécifique d'une étude comparative de ces différentes méthodes, nous nous sommes limités à exposer brièvement les méthodes enseignées à l'Institut d'Informatique de Namur.

L'analyse des besoins ou analyse d'opportunité

L'organisation peut être définie comme "un ensemble de personnes" qui font quelque chose ensemble. D'une manière très générale, toute organisation a des besoins de natures différentes. Pour chaque besoin émis, l'organisation définit le cadre, l'environnement nécessité pour le satisfaire.

Dans le cadre plus restreint d'un projet informatique, **l'analyse des besoins ou analyse d'opportunité définit un cadre général pour une solution automatisée**.

Cette analyse se base sur une situation existante et sur le réel perçu afin d'établir d'une part **une expression**

précise des besoins et d'autre part une ébauche de solution retenue. L'expression précise des besoins donne une définition générale des objectifs à atteindre ainsi que la liste des différentes contraintes à respecter. Les objectifs et les contraintes sont fixés par l'organisation. Une ébauche de la solution donne un aperçu très général de l'application à développer afin de satisfaire les objectifs préfixés. Cette première étape détermine notamment les fonctions à automatiser et les fonctions à ne pas automatiser. A cette ébauche de solution vient s'ajouter les propriétés des données et des fonctions mises en évidence, la description des performances requises, des contraintes en ressource, des contraintes organisationnelles,...

L'analyse d'opportunité nécessite la rencontre entre l'utilisateur final (la personne qui éprouve le besoin) et l'analyste. De cette rencontre naissent, autant que faire se peut, les objectifs et les contraintes du projet. Interviews, questionnaires; langages communs, ... sont les moyens mis en oeuvre pour réaliser cette tâche. Ensuite, l'élaboration d'une ébauche de solution requiert une gestion de projet qui a pour objectif d'identifier des sous-problèmes, de calculer le coût d'investissement et la rentabilité, de planifier, ... et qui dispose d'outils tels que l'analyse financière, la théorie des graphes, ... On peut retenir de cette brève description de l'analyse d'opportunité l'aspect important de **la communication entre les utilisateurs et les informaticiens**. Il est intéressant aussi de souligner que le temps passé à réaliser l'analyse des besoins est faible par rapport au temps de l'ensemble des étapes d'un projet informatique (voir la figure I.2 de la page 15).

La spécification fonctionnelle

Après l'analyse des besoins, la spécification fonctionnelle donne une définition de ce que la solution automatisée devra réaliser. Il importe que cette définition qui lie le client au producteur du système soit

- précise, simple et standard afin d'établir un "bon" dialogue entre les parties concernées;
- complète et cohérente.

Partant de l'analyse des besoins, la spécification fonctionnelle produit la définition de l'ensemble des opérations à automatiser, de l'ensemble des objets manipulés par le système. L'analyse fonctionnelle détermine aussi l'interface entre le système automatisé et son environnement, les contraintes de performance et elle

établit une brève description de l'environnement technique nécessaire à l'exploitation du système. Les informations de gestion de projet et une maquette recevant l'approbation du client composent, dans le meilleur des cas, le dossier de l'analyse fonctionnelle.

Afin de systématiser une démarche, il existe des modèles qui font l'objet d'une description informelle et intuitive dans le livre de François Bodart et d'Yves Pigneur [BOPI 83] au chapitre 2 :
le modèle de structuration des informations sert à définir la sémantique des données appartenant à la mémoire ou base des informations du système d'information;
le modèle de structuration des traitements permet de décomposer un projet en traitements de plus en plus élémentaires;
le modèle de la dynamique des traitements permet de décrire l'enchaînement des traitements;
le modèle de la statique des traitements permet de déterminer les messages en entrée et en sortie, la mémoire du système d'information ainsi que la procédure de traitement nécessaire à la réalisation d'un traitement donné;
le modèle des ressources permet de caractériser les ressources utilisées pour exécuter les procédures de traitement.

Malgré l'importance de cette étape, on note (cfr. figure I.2 page 15) que seulement 3 % du temps total du cycle de vie d'une application est consacré à la spécification fonctionnelle.

La conception

La conception élabore une solution automatisée "abstraite" c'est-à-dire une solution indépendante de l'environnement final (machine et langage de programmation concrets) qui réalise les spécifications fonctionnelles.
"Cette solution se compose :

1) d'algorithmes
2) de structures de données
3) de relations entre algorithmes, entre structures de données, entre algorithmes et structures de données".[1]

[1] extrait de [VLAM 85]

Ces trois points sont réalisés dans un pseudo-code dans le but de rendre la solution multi-cibles.

La conception se divise en général en deux étapes successives : la **conception globale** et la **conception détaillée**.

Le but de la conception globale est de déterminer une architecture logique du système, "c'est-à-dire une structuration de la solution automatisée en composants et en relations bien définies entre ces composants". [VLAM 85]. La première démarche pour réaliser une architecture logique consiste à structurer d'une façon hiérarchique le système ou à découper le système en niveaux différents ordonnés. Les relations possibles entre les différents niveaux sont exporte/importe, active, utilise, ... La deuxième démarche de conception d'une architecture logique s'appelle la structuration modulaire du système. Cette démarche se décompose en deux points :

1) découpe en modules
2) pour chaque niveau défini lors de la première démarche, identifier les modules appartenant à ce niveau.

Par ce principe de construction descendante et/ou ascendante en niveaux avec une même relation entre niveaux, on obtient une architecture logique qui est représentable sous forme d'un graphe où le noeud correspond à un module et où l'arc indique la relation ainsi que sa nature. Après cela, chaque module reçoit une spécification c'est-à-dire "ce qu'il doit faire". Il existe deux types de modules :

- les modules de traitement pour lesquels le développeur détermine les propriétés des entrées et des sorties;

- les modules de données pour lesquels il est nécessaire de spécifier les propriétés des opérations associées aux données.

Il existe trois manières d'établir les spécifications d'un module :

- la spécification par règles c'est-à-dire que le développeur établit un ensemble non structuré de phrases décrivant le problème;

- la spécification par assertion (notion de pré/post conditions);

- la spécification par type abstrait consiste à établir une stucture de données associée à un concept de base du problème.

Les modules ainsi développés doivent posséder ,autant que faire se peut, les qualités suivantes :

- forte capacité de cacher de l'information;

- faible degré de couplage c'est-à-dire une représentation simple de l'interface du module;

- forte cohésion en d'autres termes un degré élevé d'interdépendance entre condition/action du module.

Un des points importants dans la conception globale est l'établissement d'un jeu de tests dérivables à partir des spécifications des modules. Ces tests s'appellent "test black box".

"La conception détaillée a pour but d'obtenir des solutions abstraites en termes d'algorithmes et de représentations de structures de données qui réalisent les spécifications des différents modules de l'architecture logique du système". [VLAM 85]. Toujours réalisée dans un pseudo-langage pour garder le caractère abstrait, la conception détaillée contient en plus un jeu de tests "white box" dérivables à partir des algorithmes, les critères de couverture ainsi qu'une procédure de conduite ultérieure des tests. Ces tests "white box" s'avèrent inutiles lorsque les algorithmes sont conçus à partir d'une méthode de démonstration mathématique. Ces méthodes (exécution symbolique, invariant, programme auxiliaire basé sur une extension du principe de la récurrence) s'appliquent généralement à de petits algorithmes.

Le codage

Il s'agit de transformer la solution "abstraite" construite à l'étape précédente en solution "concrète" c'est-à-dire une solution qui est conçue pour une configuration particulière matériel/logiciel donnée.

La première étape du codage est la détermination d'une architecture physique du système. Ce travail consiste

- à déterminer les unités d'exécution, les unités de compilation, les procédures au sein d'un programme, ...;

- à déterminer les relations entre les unités définies;

- à definir une loi d'obtention de chaque unité physique à partir des modules de l'architecture logique;

- à produire un plan de tests d'intégration ainsi qu'un plan de codage.

La deuxième étape consiste à produire dans un langage de programmation les différents modules de l'architecture physique.2

La conduite des tests, l'intégration et la mise au point

Le code produit à l'étape précédente est exécuté sur base des plans de test conçus lors de la conception. Pour chaque test, on effectue une comparaison résultat attendu-résultat effectif. Dans le cas où l'on enregistre une disconcordance entre ces deux résultats, la correction s'effectue par des retours en arrière dans tous les documents produits aux étapes précédentes.

L'exploitation

L'utilisateur dispose du logiciel développé.

La maintenance

La maintenance est le fait d'effectuer les mises à jour d'un logiciel afin de satisfaire les nouvelles exigences de l'utilisateur ou pour adapter le logiciel à des normes définies par la loi ou encore pour adapter le logiciel à un environnement technique différent (autre système d'exploitation, nouveaux terminaux, ...). La

---

2 extrait de [VLAM 85]

maintenance consiste aussi à effectuer des modifications qui proviennent des erreurs découvertes lors de l'exploitation. Cette étape est généralement la plus coûteuse dans le cycle de vie d'une application informatique (cfr. figure I.2 page 15). En effet, une mise à jour d'un logiciel qui apparaît comme un détail peut avoir des répercussions assez importantes. Il s'agit pour certaines mises à jour de reprendre les étapes du cycle de vie de l'application. Dès lors, avant de mettre à jour un logiciel, il est indispensable d'estimer l'impact de la modification sur le logiciel ainsi que d'évaluer les coûts de cette modification.

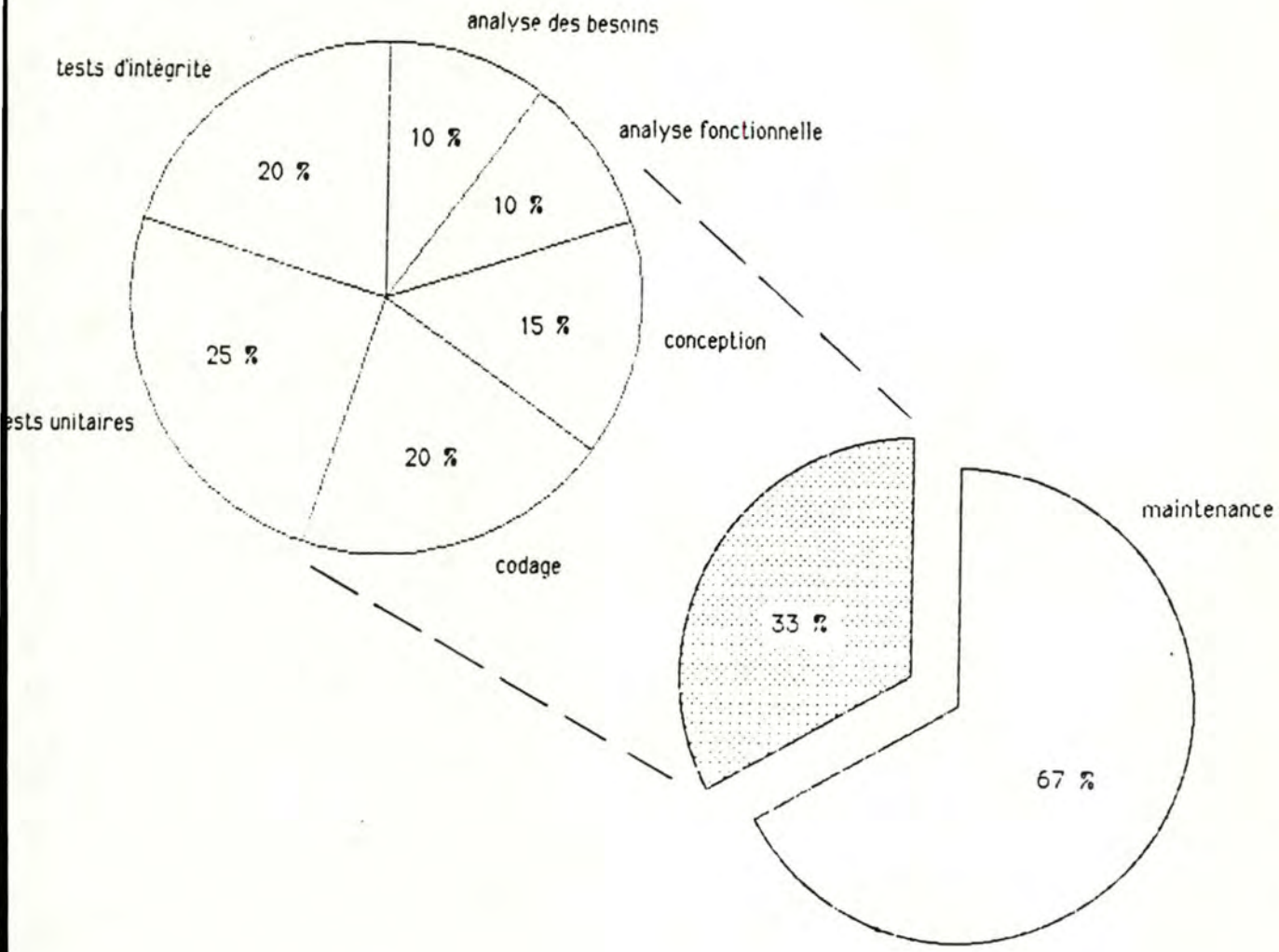


Figure I.2 : répartition du temps alloué aux étapes du cycle de vie d'une application informatique.3

---

[3] extrait du cours de méthodologie du développement de logiciels donné par Axel Van Lamsweerde à l'Institut d'Informatique à Namur.

1.2.3 Les problèmes rencontrés

Les problèmes rencontrés au cours du cycle de vie d'une application dépendent quelquefois de la spécificité du projet, du personnel et des outils disponibles. Cependant, certains problèmes se regroupent autour de quelques grands thèmes : la maintenance, la productivité, la communication, ...
Il s'agit de déterminer, autant que faire se peut, les origines et les conséquences de ces problèmes rencontrés.

L'augmentation des demandes d'applications informatiques

D'une manière générale, l'augmentation de la demande en logiciel provient d'une évolution de **la technologie de l'information**. En effet, l'informatique a été introduite dans les entreprise dans les années 60. Depuis lors, les techniques d'acquisition des informations se sont considérablement développées. Les centres informatiques doivent gérer de plus en plus de gros volumes d'informations. Ceci n'est pas sans conséquences. Par le pouvoir de détenir de l'information, le département informatique d'une entreprise devient le centre nerveu de celle-ci et de ce fait, il subit des pressions. De plus, le fait que l'utilisateur devient plus exigent est certainement une des conséquences de l'amélioration de la technologie de l'information.

La **baisse du coût de traitement, de stockage et de communication** intervient dans une certaine mesure dans l'augmentation de la demande en logiciel. En effet, cette baisse de prix du matériel améliore la rentabilité des projets informatiques. Il se peut très bien que des applications anciennement non rentables se justifient à l'heure actuelle.

On observe dans le marché un phénomène de mondialisation. La concurrence augmente. La **gestion de l'entreprise** doit être menée avec **précision**. La direction qui veut gérer son entreprise d'une manière efficace exige un niveau de contrôle élevé. Ce contrôle - souvent réalisé par des applications informatiques - alourdit le carnet de commandes du département informatique.

Le manque de productivité des services informatiques

La productivité d'un service informatique ne peut se mesurer que difficilement et dépend très fort de la **manière de concevoir un projet** et de **l'environnement humain**.

En ce qui concerne la conception d'un projet, un facteur qui influence la productivité du programmeur est la **dimension du programme**. La productivité sera plus basse avec de grands programmes qu'avec de petits. Dès lors, la méthode de conception doit veiller à créer une architecture modulaire dans laquelle aucun module ne croît de manière excessive. On observe aussi une baisse de productivté avec des **programmes extrèmement complexes**. La méthode de conception doit aussi essayer, autant que possible, de diminuer la complexité des modules confiés aux programmeurs afin d'accroître leur productivité. Malheureusement, la théorie sur la conception d'une architecture d'un logiciel développée brièvement dans la section précédente n'est guère appliquée. Le logiciel est souvent bâti sur une très mauvaise architecture ... de sorte que la productivité n'est pas optimale et la **maintenance** devient très **coûteuse**. A l'heure actuelle, les 2/3 du coût total en moyenne sont consacrés à des travaux de maintenance (voir figure I.2 page 15). Un expert prévoit, si les conditions de productivité des informaticiens ne changent pas, que le pourcentage des ressources d'un service informatique consacré à la maitenance atteindra 100 % . Dès lors, puisque le personnel se trouve sans cesse occupé à des travaux de maintenance peu motivant, on observe un manque de personnes disponibles pour développer de nouvelles applications. Le manque de productivité provient aussi de l'utilisation d'outils et de méthodes démodées face à l'évolution des besoins et des techniques.

En ce qui concerne l'environnement humain, il est évident que les programmeurs diffèrent par leur compétence: les bons programmeurs ont une productivité plus élevée que les médiocres. D'autre part, plutôt qu'effectuer un développement monolithique d'un système, il est recommandé de procéder à des développements successifs de sous-systèmes appelés "utiles". Une telle stratégie améliore la motivation des personnes qui développent un système car elle permet de visualiser l'état d'avancement du projet. De plus, elle rassure le client.

L'explosion de la demande en applications et le manque de productivité = un fossé croissant entre les informaticiens et les utilisateurs

Ce fossé peut être perçu à deux niveaux : utilisateurs et informaticiens.

Au niveau de l'utilisateur, on enregistre un retard dans la livraison des applications dont l'utilisateur a besoin. Ce retard a tendance à s'accroître d'année en année. Une enquête auprès de 25 firmes américaines assez importantes permet de constater la progression d'un retard moyen de programmation qui passe de 19 mois en 1981 à 27 mois en 1984 ... . Dans une société namuroise, un vaste projet de calcul de prix de revient a été reporté d'un an. Les utilisateurs finals se trouvent rapidement frustrés par le long retard et par l'incapacité de l'informatique à répondre à leurs besoins. De plus, l'utilisateur qui subit ce retard aura tendance à ne plus demander au service informatique de développer de nouvelles applications. Dans certains cas, l'ampleur de ce retard s'avère tellement grande que l'utilisateur final ne peut plus attendre le service informatique et crée un petit département informatique au sein de son service. Cette pratique qui décentralise l'information doit être soigneusement étudiée en termes de coût et de faisabilité. En effet, les problèmes de communications, de compatibilité de machines, de compatibilité au niveau de données apparaissent .... ce qui peut engendrer de sérieux problèmes à l'organisation toute entière ... voire même la mettre en péril ...

En ce qui concerne l'informaticien, il voit son carnet de commandes en logiciels augmenter sans cesse. Dès lors, le personnel qui développe n'a pas une grande motivation car il ne sait pas satisfaire l' utilisateur dans les délais désirés. L'informaticien qui a planifié les gros projets informatiques se voit dans l'incapacité de prendre en compte d'une manière satisfaisante les applications légères, urgentes, ponctuelles et non planifiées.

Problèmes plus spécifiques au cycle de vie d'une application informatique

D'après une étude réalisée par une banque canadienne, 56 % des erreurs découvertes dans la phase "tests" du cycle de vie d'une application informatique proviennent d'une spécification erronée, incomplète ou incohérente. L'effort consacré à la correction de ces erreurs par rapport à l'ensemble du travail de correction s'élève à 88 % . En effet, dans le cas de la correction d'une erreur, la conception de la modification nécessaire est évidemment d'autant plus complexe que l'erreur a été commise tôt dans le cycle de vie. [VLAM 85]

Quelles sont les causes de ces erreurs ?

Répondre à cette question d'une manière exhaustive n'est pas réaliste. On doit plutôt essayer de dégager certaines tendances.

Lorsque l'informaticien et l'ensemble des utilisateurs se rencontrent, on remarque qu'ils ont des cultures très différentes. L'informaticien maîtrise la terminologie informatique et les méthodes d'analyse tandis que l'utilisateur final s'exprime dans un langage différent. Or, ces deux acteurs doivent communiquer **sans ambiguïté ni malentendu**. A cette fin, on rédige des spécifications pour essayer de clarifier et de formaliser le traitement. Malheureusement, des **problèmes de communication** entre l'utilisateur et l'informaticien subsistent car il est impossible que chacun puisse comprendre les nuances et les subtilités du mode de pensée de l'autre.

On remarque aussi que l'utilisateur final ne sait pas très bien ce qu'il veut. Et pourtant, des spécifications bien fixes doivent exister et servir de base à la suite du développement. C'est pourquoi, l'utilisateur est invité à parapher le document des spécifications. On effectue par ce procédé le **gel des spécifications**. Ce procédé présente des inconvénients. En effet, d'une part, l'informaticien aura tendance à se retrancher derrière le document signé et sera très réticent à le modifier. D'autre part, l'utilisateur, soucieux de respecter ses engagements ou horrifié par le fait de devoir se contredire ou se compléter ne communiquera peut-être pas les modifications qui lui viennent à l'esprit. Dès lors, le logiciel créé sera essayé et puis éventuellement abandonné par l'utilisateur car il ne correspond pas aux exigences de ce dernier. En plus de ces deux inconvénients, on observe

chez l'utilisateur une certaine hésitation à signer ce document : il doute quelque peu de la complétude des spécifications proposées par l'informaticien. Cette hésitation provoque dans une certaine mesure un léger retard à la livraison du logiciel.

Le principe d'incertitude est certainement à l'origine de certaines erreurs rencontrées dans les spécifications. En physique, le principe d'incertitude établit que l'action d'observer des événements subatomiques modifie ces événements. Ce même principe s'applique dans le traitement des données. En effet, le fait de fournir à un utilisateur ce dont il a besoin modifie la perception de ses besoins. L'informaticien qui livre un logiciel à un client se trouve confronté à une cascade d'événements imprévisibles qui proviennent de l'imagination créative du client face à une solution automatisée.

Si on analyse le rôle de l'utilisateur à travers les étapes du cycle de vie (cfr. supra), on remarque que ce rôle est inactif pendant les phases d'analyse fonctionnelle, de conception et de codage. Dès lors, l'utilisateur imagine mal le résultat de ces trois étapes. Il y a donc peu de chance d'obtenir une adéquation complète entre l'idée qu'il se fait de l'application et l'application réellement développée par l'informaticien.

Section 1.3 : L'informatique décisionnelle

Alors que l'informatique opérationnelle manipule des informations routinières, l'informatique décisionnelle manipule des informations non-routinières. Il s'agit de l'informatique destinée au cadre de l'entreprise. Cette informatique aide l'utilisateur final à exercer des contrôles, à prendre des décisions et à élaborer des prévisions.

"Décider, c'est identifier et résoudre les problèmes que rencontre toute organisation". H.A. Simon.

1.3.1 Cadre de référence pour les systèmes d'aide à la décision 4

1.3.1.1 Le cycle de vie du processus de décision

Il est possible de modéliser les étapes d'un processus décisionnel. Nous allons rapidement passer en revue ces différentes étapes.

1) la perception : la perception d'un problème provient d'une situation d'insatisfaction. Le décideur perçoit un problème à partir de stimuli structurés ou à partir de stimuli non-structurés. Dans le cas de la perception d'un problème à partir de stimuli structurés, l'on perçoit un problème par rapport à des règles. Par contre, les stimuli non-structurés ne sont plus liés à des normes ou des règles. Ils proviennent de la capacité de perception des gens.

2) identification du problème : il s'agit d'établir un diagnostic des causes profondes c'est-à-dire distinguer les causes réelles des conséquences.

3) la définition des alternatives : selon l'approche du décideur, la taille de l'ensemble des solutions possibles diffère.

4) choix d'une solution : parmi l'ensemble des solutions définies au point 3, il s'agit d'effectuer un choix. Pour réaliser ce choix, l'on dispose de critères d'évaluation des solutions possibles.

5) implémentation : il s'agit de mettre en oeuvre la solution choisie en fonction des moyens consacrés à cette solution et en fonction du contexte dans lequel évolue l'organisation.

6) contrôle de la solution : après la mise en oeuvre, il est intéressant d'évaluer si cette solution apporte un remède à la situation initiale d'insatisfaction.

---

[4] extrait du cours de système d'aide à la décision donné par François Bodart à l'Institut d'Informatique à Namur.

1.3.1.2 Typologie des problèmes décisionnels dans une organisation

Les problèmes stucturés : il s'agit de problèmes parfaitement identifiables qui traitent de l'information routinière. Par conséquent, pour un problème donné, on établit une spécification parfaite de sa définition, des alternatives, des règles de calcul d'une solution, des critères de choix d'une solution.

Les problèmes non structurés : ces problèmes traitent de l'information non routinière. Pour ce type de problème, on est incapable d'établir une spécification parfaite. Dans les entreprises, la gestion à long terme rencontre souvent des problèmes non structurés. (exemple classique : la création d'un nouveau produit est un problème non structuré.)

Les problèmes semi-structurés : il s'agit de problèmes pour lesquels au moins un des 3 aspects suivants :
1° identification de la solution
2° les alternatives
3° le choix d'une solution
peut être spécifié d'une manière précise et complète. Ces problèmes ont donc une spécification partielle. Dès lors, le support du problème à résoudre, c'est-à-dire le système d'information, a une structure relativement pauvre. En effet, une série de faits ne sont pas connus et certains liens sémantiques sont inconnus. La structure des informations doit être évolutive afin de pouvoir prendre en considération la découverte d'un nouveau fait ou d'un nouveau lien sémantique. A l'heure actuelle, on observe que les systèmes d'aide à la décision (S.A.D.) s'intéressent surtout aux problèmes semi-structurés. Dès lors, les S.A.D. doivent être conçus dans une perspective évolutive car les spécifications fonctionnelles doivent évoluer à travers un processus d'apprentissage de l'utilisateur.

1.3.1.3 Manière de prendre une décision dans une organisation

Afin de déterminer un cadre de référence pour les systèmes d'aide à la décision (S.A.D.), la théorie établit des modèles qui définissent la manière dont les gens prennent des décisions. Ces modèles se basent sur deux

aspects opposés : - l'aspect rationnel d'un processus de décision;
- l'aspect psychologique du processus de décision.

Nous nous limiterons à décrire brièvement quatre modèles qui sont très répandus dans la littérature.

1) le modèle du comportement rationnel du décideur
Dans ce modèle, on suppose que le décideur a un comportement rationnel et qu'il identifie parfaitement son problème. Le problème de décision est structuré en termes de variables de décision, variables indépendantes, paramètres, relations, contraintes. Ce modèle recherche une solution optimale. Dès lors, les décisions peuvent être prises à l'aide de la recherche opérationnelle et à l'aide de procédures bien déterminées. Bien spécifiés, ces problèmes traitent de l'information routinière.

2) le modèle politique
Dans ce modèle, on considère que toute situation décisionnelle provient d'une situation conflictuelle. Le processus de décision est donc un processus de résolution de conflits. La décision est vue comme l'application de stratégies et de tactiques par des groupes de décision qui, chacun, essayent d'orienter la prise de décision vers l'intérêt du groupe. La puissance et l'influence au sein de l'organisation déterminent le résultat de la décision. Dès lors, il s'agit de déterminer les différents groupes impliqués dans la décision et leur opinion, l'importance respective des groupes et leur poids et enfin les moyens d'influencer les opinions des autres.

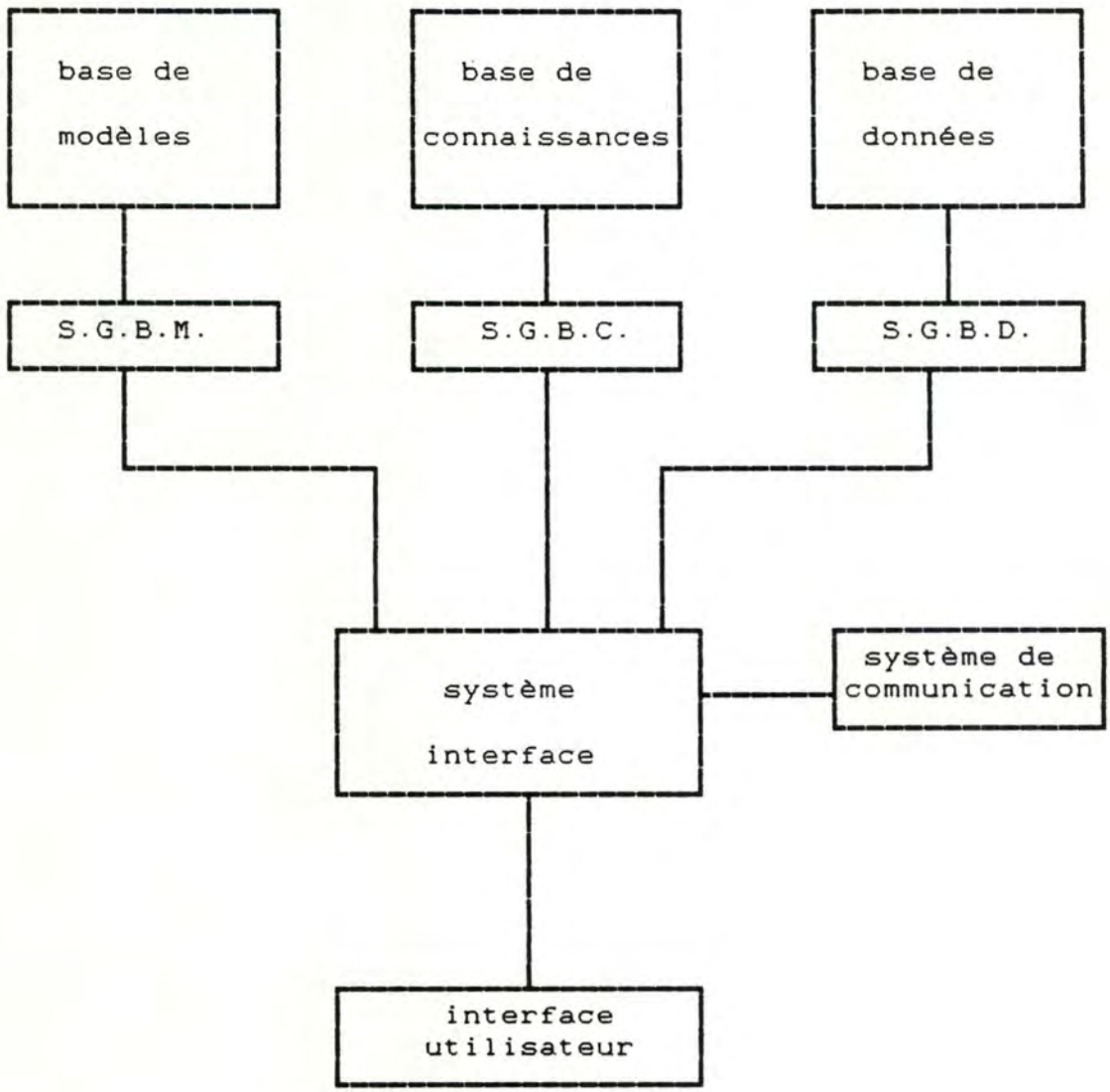
3) le modèle de la poubelle
Dans ce modèle, une situation de décision résulte de deux circonstances : une opportunité de décision et une opportunité de moyens. Une situation decisionnelle naît donc de la mise en oeuvre de nouveaux moyens de résolution et de l'existence d'une décision à prendre. Les informations nécessaires à ce modèle sont :
- les problèmes pouvant survenir, leur probabilité et une estimation du moment où ils pourraient se produire;
- les solutions possibles, leur probabilité;
- les occasions de choix potentielles, leur probabilité.

4) le modèle des processus ou modèle du comportement satisfaisant
Ce modèle, développé par Simon, se base sur le fait que le gestionnaire ne cherche pas à optimiser sa décision mais à progresser par rapport à une situation antérieure. Les décideurs raisonnent en se fixant une cible qui est établie en fonction de standards ou de procédures. Ces

standards sont déterminés en fonction de certaines heuristiques (exemple de standard : les ratios en analyse financière). Dans ce modèle, on doit avoir une information sur l'ensemble des procédures et des standards gérés par l'organisation.

A partir de ces différents modèles, on peut conclure que les systèmes d'aide à la décision doivent contenir les informations demandées par les quatre modèles. Dès lors, les S.A.D. doivent traiter de l'information formelle et informelle. En effet, au fur et à mesure que l'on s'écarte du modèle rationnel et du modèle du comportement satisfaisant, il est indispensable de pouvoir gérer de l'information informelle (par exemple l'influence des parties dans le modèle politique). Les S.A.D. doivent donc gérer des informations non structurées.

1.3.2 Schéma général des systèmes d'aide à la décision [5]



[5] extrait du cours de système d'aide à la décision donné par François Bodart à l'Institut d'Informatique à Namur.

Cette architecture est très théorique et se compose :
- d'une base de modèles ou boîte à outils avec un système de gestion de base de modèles (S.G.B.M.);
- d'une base de connaissances qui contient les spécifications sur les données et sur les modèles dans un formalisme unique avec un système de gestion de base de connaissances (S.G.B.C.);
- d'une base de données avec un système de gestion de base de données (S.G.B.D.);
- d'un interface utilisateur afin de permettre à l'utilisateur d'exploiter lui-même le système;
- d'un système de communication.

1.3.3 Les problèmes rencontrés

A l'heure actuelle, l'informatique opérationelle se voit dans l'incapacité de prendre en charge l'informatique décisionnelle, car modéliser le processus de prise de décision présente des caractéristiques opposées à celles du développement classique. L'**approche évolutive**, présente dans l'informatique décisionnelle, nécessitent la réalisation d'un prototype. Sur base du prototype, on procède ensuite à un développement incrémental basé sur les demandes de l'utilisateur.

Les facteurs psychologiques, présents dans certains modèles (cfr. supra), rendent indispensable la gestion de données informelles.

L'usage d'un S.A.D. doit être personnalisé. Dès lors, il s'agit que le S.A.D. soit conçu dans **une optique interactive**. Cette interactivité doit permettre d'observer le comportement du système. Les systèmes experts sont une excellente approche. En effet, le système expert a la possibilité de montrer le cheminement qu'il emploit pour obtenir la solution.

Le problème du dialogue homme-machine est présent dans les S.A.D. . La **convivialité** doit être la préoccupation du concepteur d'un S.A.D. .

Dans le cas où les différents aspects évoqués (évolutif, interactif et convivialité) ne sont pas présents dans un S.A.D. livré aux gestionnaires de

l'entreprise, ceux-ci seront éventuellement frustrés et abandonneront le système pour reprendre leurs anciennes méthodes.

D E U X I E M E C H A P I T R E

L ' A P P R O C H E Q U A T R I E M E G E N E R A T I O N

Section 2.1 : Conflit de générations

S'il est une génération de langages dont il est difficile de donner une définition à la fois complète, précise et sans ambiguïté, c'est de la quatrième génération qu'il s'agit. Trop de gens, en effet, se targuent de "faire de la quatrième génération" dès lors qu'ils se voient mettre à leur disposition un logiciel un peu plus complet ou un peu plus performant que celui de leurs confrères.

Le générique "quatrième génération" a une fâcheuse tendance à devenir, dans l'esprit de beaucoup, une sorte de "fourre-tout" de luxe, où l'on range tout produit n'appartenant ni à la troisième, ni à la cinquième génération. Certains promoteurs de nouveaux produits logiciels ne sont d'ailleurs pas étrangers à ce phénomène. Nous allons tenter, à travers les lignes qui suivent, de cerner la philosophie "quatrième génération" et d'en présenter les caractéristiques marquantes.

Loin de nous l'idée de vouloir donner ici une définition immuable et sans reproches, mais au moins avons-nous l'ambition de dégager, tout au long de ce travail, les particularités propres aux langages de quatrième génération.

A cette fin, il est bon, dans un premier temps, de rappeler brièvement ce que nous entendons par chacune des générations de langages...

2.1.1. Première génération

C'est le règne du langage machine. Le langage machine est le répertoire des instructions possibles pour un processeur particulier; chaque instruction est constituée d'une suite de symbôles binaires. Ce langage est le seul qui soit directement "compris" par le processeur.

2.1.2. Deuxième génération

L'assembleur. Le moins qu'on puisse dire est qu'écrire un programme sous forme de suites de symbôles binaires n'est vraiment pas très pratique, et presque personne ne programme en langage machine. Le langage d'assemblage, ou

assembleur, a la même structure que le langage machine, mais les suites de symbôles binaires ont été remplacées par des symbôles mnémoniques qui donnent aux programmes une forme un peu moins obscure.

2.1.3. Troisième génération

Les langages dits "de haut niveau". L'écriture de programmes d'application en assembleur présente un certain nombre d'inconvénients quelque peu rédhibitoires. Tout d'abord, les programmes en assembleur, tout comme les programmes en langage machine, ne sont pas transportables, c'est-à-dire qu'ils sont exclusivement utilisables sur le type de machine pour lequel ils ont été écrits. Cela limite leur diffusion et peut donner énormément de travail à l'utilisateur qui veut changer de machine : il devra réécrire tous ses programmes.

Ensuite, la programmation en assembleur exige un très grand nombre d'instructions, puisque celles-ci sont élémentaires. Il en résulte que l'écriture de programmes en assembleur est fastidieuse et entraîne beaucoup d'erreurs.

Une troisième remarque est que les programmes écrits en assembleur sont en général difficiles à lire, à comprendre et à modifier, même avec une bonne documentation. Ajoutons enfin que l'évolution technologique a rendu le travail par le processeur de moins en moins cher et, par comparaison, le temps de travail des programmeurs de plus en plus précieux.

Toutes ces raisons sont à l'origine du développement des langages de haut niveau, plus adaptés aux utilisteurs. Chaque instruction d'un tel type de langage exprime d'une manière compacte une opération qui nécessiterait une suite parfois assez longue d'instructions si elle était codée en langage machine ou en assembleur. Mais avant de pouvoir être compris et exécutés par le processeur, les instructions et les programmes rédigés en un langage de haut niveau doivent être traduits en langage machine; c'est le rôle des compilateurs et des interpréteurs. Sans vouloir entrer dans le détail, remarquons néanmoins que cette étape supplémentaire avant l'exécution d'un programme ne pourra être franchie que moyennant un supplément en consommation de ressources machine.

Il est difficile d'énumérer de manière exhaustive l'ensemble des langages de haut niveau disponibles actuellement sur le marché : il y en a des centaines ! Epinglons, pour la forme, les plus connus, tels COBOL,

PL/1 pour les problèmes de gestion; C, FORTRAN, PASCAL pour des problèmes à caractère plus scientifique.

COBOL est le langage de programmation le plus utilisé dans le monde sur les gros ordinateurs, en raison du grand nombre de progammes existants, écrits dans ce langage. Nombreux, cependant, sont ceux qui considèrent COBOL comme un langage archaïque et mal structuré...

2.1.4. Quatrième génération

Les "L4G". Les problèmes inhérents à la troisième génération ont été amplement développés dans le premier chapitre de cette partie. Nous nous bornerons simplement à rappeler qu'il y était notamment question d'une productivité peu élevée en matière de développement d'applications, ce qui ne manquait pas de provoquer un retard sans cesse croissant dans la livraison des applications et, par là, un sentiment de frustration bien compréhensible chez l'utilisateur final.

Les langages de quatrième génération sont donc censés apporter une solution, au moins partielle, à ces problèmes de l'informatique traditionnelle.

Ainsi, un L4G digne de ce nom doit constituer un environnement logiciel intégré, articulé autour d'une base de données, et permettant de faire accéder les différents acteurs du système d'information aux sous-systèmes qui les intéressent et ce, moyennant une simple description de la requête qu'ils souhaitent voir satisfaire.

Une formation minimale devra permettre à un utilisateur final d'accéder, à l'aide d'un langage simple et puissant, à une base de données, de gérer les données, de les consulter, de les extraire et de les mettre sous forme de rapport, sans avoir recours à l'arsenal du service informatique. En outre, l'utilisateur final aura la possibilté de construire des procédures, et ainsi de constituer progressivement un système d'information complet.

Un informaticien professionnel trouvera dans un L4G le moyen de développer des applications dans des délais sensiblement réduits par rapport à ceux nécessités par un langage classique de programmation, tout en offrant la possibilité d'exprimer des algorithmes complexes.

Cette définition met l'accent, d'une part, sur la possibilité que reçoit l'utilisateur final de résoudre lui-même une partie de ses problèmes informatiques - ce

qui ne devrait pas manquer de soulager la charge du service informatique - et d'autre part, sur l'accroissement de productivité escompté de ce même service informatique. Nous verrons dans la deuxième partie de cette étude, si les L4G répondent bien à ces attentes...

2.1.5. Cinquième génération

C'est le domaine de l'intelligence artificielle. Le développement inéluctable de l'intelligence artificielle résulte de trois types de contraintes imposées par l'outil informatique traditionnel. Tout d'abord, l'échange d'informations entre l'homme et la machine ne peut se faire qu'au moyen de périphériques lents, limités et sans nuance. La deuxième contrainte concerne le langage dont la rigidité syntaxique implacable ne tolère aucune erreur. Le troisième impératif auquel tout informaticien est obligé de se plier est le plus contraignant de tous : il concerne la logique, la structure du raisonnement. En effet, l'accès à l'informatique est obligatoirement algorithmique : l'ordinateur ne peut résoudre un problème que si on lui founit la méthode, les formules, les séquences à suivre.

"La machine se pliera un jour à la manière dont l'utilisateur aura envie de communiquer et de poser une question. Ce renversement dans le rapport de force peut être considéré comme une définition de l'intelligence artificielle" [DEBR 87].

Plus prosaïquement, l'intelligence artificielle est une science, dont la matière d'étude est la connaissance, c'est-à-dire tous les processus nécessaires et mis en oeuvre par un système autonome (comme l'homme, par exemple) pour résoudre différents types de problèmes. La structuration de la connaissance apparaît donc comme un concept clé de l'intelligence artificielle.

Parmi les grandes applications de l'intelligence artificielle figure le langage naturel, mais aussi les systèmes experts pour lesquels il a fallu commencer par comprendre le mécanisme de la découverte : lors de la résolution d'un problème, quel qu'il soit, on est toujours en présence d'un espace de possibilités, et il faut trouver la solution dans cet espace. La difficulté à laquelle on est généralement confronté, c'est qu'on peut bien connaître cet espace, mais que celui-ci est trop vaste pour qu'on puisse l'explorer complètement.

Un expert est quelqu'un qui trouve rapidement, dans cet espace, la voie la plus efficace à suivre pour résoudre un

problème déterminé : il connaît un très grand nombre de situations typiques, et est capable de réaliser de très nombreuses associations entre des cas nouveaux et des cas déjà rencontrés, en y discernant des similarités. C'est cela l'objectif des systèmes experts développés en intelligence artificielle : définir les règles heuristiques qu'utilisent les hommes-experts et les faire mettre en oeuvre par l'ordinateur.

Ainsi, un système expert se compose : d'une base de faits contenant des propositions reconnues comme vraies; d'une base de règles d'inférence contenant un ensemble de règles qui permettent de dériver de nouveaux faits à partir de faits existants; d'un moteur d'inférence capable de choisir des règles et de les appliquer à la base de faits pour en dériver de nouveaux faits ou pour évaluer une proposition.

Section 2.2 : Climat favorable à l'approche 4ème génération

2.2.1. Les bases de données

Introduction

Selon certaines sources, plus de 80 pourcents du temps du processus de maintenance consacré à un programme typique de haut niveau (COBOL) concerne, au sens large, les accès aux fichiers. Ceci inclut la définition des fichiers et de leurs tampons, les opérations de consultation, de sélection et de mise à jour, ainsi que la validation des données en entrée et l'édition de données traitées. Moins de vingt pourcents du code se rapportent au traitement proprement dit des données, après qu'elles aient été lues, et avant qu'elles ne soient écrites.

Un autre problème provient de ce que le même code (relatif à l'accès aux données et à leur édition) est souvent répété dans des programmes différents.

L'apparition des Systèmes de Gestion de Bases de Données (SGBD), à la fin des années '60, résultait de la volonté des concepteurs de systèmes de libérer - en partie - les informaticiens des tâches répétitives d'accès aux fichiers et d'édition des données. Elle reposait également sur une nouvelle prise de conscience des grands

principes de l'informatique, simples mais quelque peu oubliés : chaque donnée n'est encodée qu'une seule fois; chaque donnée n'est mémorisée qu'en un seul endroit. Malheureusement, dans de nombreux cas, la réalité est tout autre : toute nouvelle application développée met en oeuvre ses propres fichiers, et il n'est pas rare qu'il faille encoder plusieurs fois les mêmes données avec, comme conséquence immédiate, la redondance de celles-ci au sein d'une même entreprise. A côté du gaspillage d'espace mémoire et de temps d'encodage que cette situation implique, se posent aussi des problèmes d'incohérence de données, et d'absence de standardisation des formats de celles-ci.

L'incohérence des données résulte de ce que, par exemple, si deux informations équivalentes sont mémorisées à deux endroits différents, elles ne seront sans doute pas mises à jour en même temps, puisqu'elles sont exploitées par des programmes différents.

Un système de gestion de base de données est un logiciel qui a pour fonction essentielle de remédier à cette anarchie découlant de la multiplication incontrôlée des données. Un SGBD devra permettre de gérer un grand nombre de données communes à plusieurs programmes ou utilisateurs et ce, de manière cohérente et fiable. Les données ainsi gérées constituent la base de données (BD).

Une base de données est donc une collection partagée de données, conçue pour satisfaire les besoins de nombreux types d'utilisateurs.

SGBD et systèmes conventionnels

Les systèmes de fichiers conventionnels (ainsi d'ailleurs que nombre de SGBD non relationnels) sont dépendants de la façon dont les enregistrements sont stockés dans les fichiers (séquencement physique et liens entre enregistrements) et de la façon dont les champs sont arrangés dans les enregistrements (positionnement, longueur, format). Cette dépendance physique des données fait perdre beaucoup de temps aux programmeurs : dans chaque programme, ils sont peut-être amenés à déclarer les mêmes fichiers, les mêmes tampons, les mêmes champs,... que ceux d'un programme existant !

Bien plus contraignants encore sont les changements des programmes source nécessités par un changement de la structure physique des données (par exemples, ajouter un champ, modifier la longueur d'un champ,...).

Un SGBD, par contre, dispose d'un Langage de Définition de Données (LDD) qui permet de définir, de

manière centrale, dans un Dictionnaire de Données (DD), les entités et leurs propriétés. Ainsi, un SGBD permet à un programme de faire référence à une propriété (attribut) par son nom (ex : 'nom-client') sans qu'il soit nécessaire de spécifier son emplacement physique. De plus, il est possible de modifier la longueur ou le format de ces propriétés, sans que les programmes qui y font référence en soient affectés.

Le LDD permet en outre de spécifier, dans le dictionnaire des données, les contraintes d'intégrité. Celles-ci sont vérifiées chaque fois qu'un enregistrement est écrit dans la base de données. Si une contrainte est violée, la mise à jour de la BD n'a pas lieu et une erreur est signalée.

Enfin, alors que dans un système de fichiers conventionnel, la stratégie nécessitée pour accéder à des enregistrements inter-reliés doit être implantée dans tout programme qui nécessite d'établir des relations entre enregistrements, un SGBD permet, toujours par le biais de son LDD, de définir dans le DD des relations entre données. Idéalement, l'implantation de ces relations est cachée au programme, qui ne sait pas, par exemple, si des pointeurs sont utilisés. Ainsi, un SGBD assure des opérations logiques d'entrées/sorties telles que : "lire le premier client", "mettre à jour l'enregistrement courant", "sélectionner tous les enregistrements ayant telle propriété",...

La partie manipulation de données d'un SGBD est assurée par son Langage de Manipulation de Données (LMD) qui procure des fonctions équivalentes aux opérations d'entrées/sorties d'un langage de haut niveau. Cependant, dans le cas d'un système de fichiers conventionnel, les opérations d'E/S devront porter sur un enregistrement bien particulier, auquel on aura accès via un numéro relatif, une valeur de clé explicite, ou une lecture séquentielle. Un SGBD, par contre, permet généralement l'accès à un ou plusieurs enregistrements sur base du contenu d'un champ, que ce dernier soit ou non une clé. De plus, à partir des relations définies dans le dictionnaire, un SGBD peut retrouver un enregistrement et tous ceux qui lui sont liés, sans qu'il faille lui indiquer un chemin d'accès explicite.

Les caractèristiques que nous venons de rapidement passer en revue constituent, à n'en point douter, des facteurs d'amélioration non négligeable de la productivité. Nous allons maintenant aborder, de manière plus systématique, les fonctions essentielles d'un SGBD.

Fonctions d'un SGBD

Les fonctions essentielles d'un SGBD peuvent se résumer comme suit :

- diminuer la redondance des données;

- diminuer, voire supprimer complètement l'incohérence des données;

- permettre le partage des données;

- assurer la standardisation des formats, de manière à permettre la migration de données entre systèmes différents;

- permettre la consultation, la modification, la suppression de données;

- assurer la sécurité. C'est tout le problème de la reprise en cas d'incident (panne machine, par exemple) : il faut permettre aux utilisateurs de poursuivre leur travail en leur assurant que les données détruites ou dans un état douteux sont restaurées;

- assurer en permanence l'intégrité de la base de données, c'est-à-dire le respect de toutes les contraintes d'intégrité spécifiées;

- assurer la régulation de la concurrence lorsque plusieurs utilisateurs accèdent en même temps à la BD, et que l'un d'entre eux au moins effectue des opérations de mise à jour;

- permettre le contôle d'accès et la confidentialité de certaines données;

- assurer l'indépendance des données par rapport aux programmes, de manière telle que ceux-ci puissent restés inchangés suite à une modification de la structure physique des données;

- donner à un utilisateur (par exemple un programme) une perception adéquate de la base en fonction de ses besoins. Ce concept est réalisé par le mécanisme des vues.

- ...

Cette liste des fonctions d'un SGBD n'est pas exhaustive; voir [DATE 81] pour plus de détails sur le sujet.

Les trois niveaux de représentation d'une BD

La description d'un base de données peut se faire à différents niveaux, suivant que l'on regarde plus du côté de l'utilisateur que du côté du stockage physique des données. On considère généralement qu'il existe trois niveaux de représentation d'une BD : le niveau externe, le niveau conceptuel et le niveau interne.

- Le niveau externe : c'est le niveau utilisateur. Le schéma externe, ou vue, décrit la façon dont seront perçues les données par un utilisateur (c'est-à-dire, par exemple, un programme d'application). En effet, il est rarement nécessaire qu'un utilisateur ait une vision globale de la base de données; il lui suffit de se limiter à la partie qui englobe les informations dont il a besoin. Ainsi, un schéma externe peut généralement être considéré comme un sous-schéma du schéma conceptuel.

- Le niveau conceptuel : c'est le niveau du schéma conceptuel, décrivant la réalité d'une organisation et de ses processus de gestion et ce, de manière abstraite mais fidèle. Le passage du monde réel au schéma conceptuel correspond à un processus de modélisation où les objets du monde réel sont classés et désignés par des noms. Le schéma conceptuel est spécifié par le Langage de Définition des Données, fourni par le SGBD.

- Le niveau interne est représenté par le schéma physique, qui spécifie comment les données seront physiquement stockées sur disque.

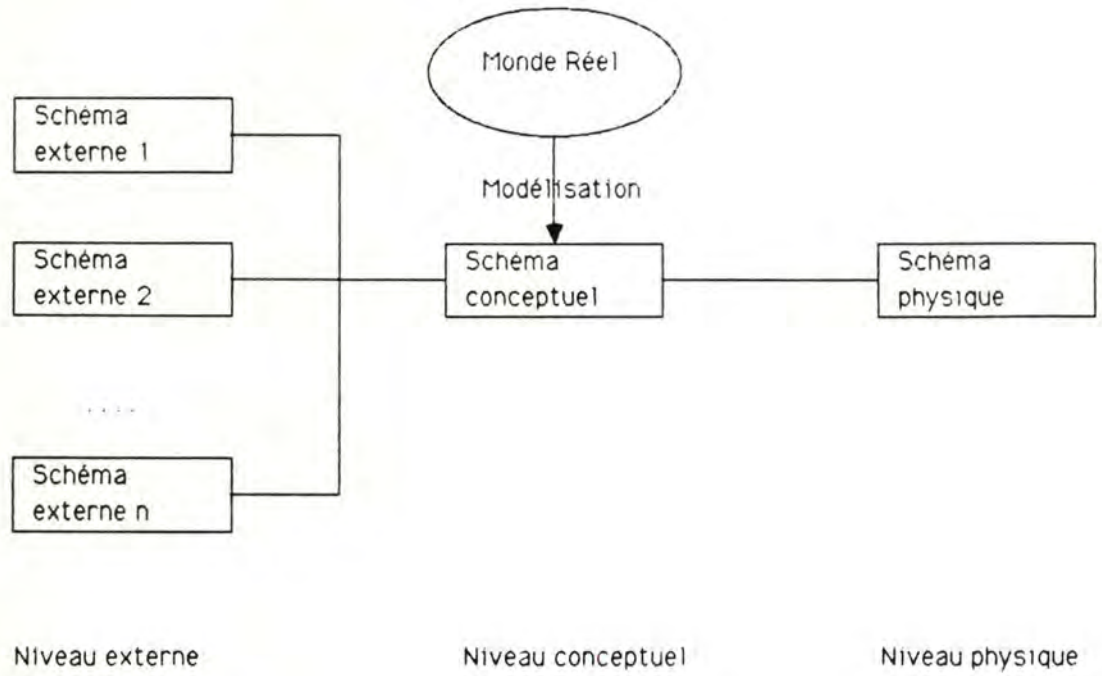


Fig. II.1 : **les différents niveaux d'une BD.**

Modèles de base de données

Ainsi que nous l'avons vu, le schéma conceptuel d'une BD peut être visualisé de trois manières différentes, suivant le modèle de données adopté. Nous allons présenter ces trois approches à l'aide d'un exemple.

Considérons les types d'entité suivants :

ETUDIANT, qui représente les étudiants inscrits dans une université déterminée;

DOMAINE, qui représente l'éventail des possibilités offertes aux étudiants en matière de choix des cours;

COURS, qui représente les cours dispensés à l'université en question.

Considérons ensuite les associations :

INSCRIPTION, qui exprime que tout étudiant est inscrit

dans un et un seul domaine, et qu'un domaine peut avoir été choisi par un nombre quelconque d'étudiants;

COMPOSITION, qui exprime qu'un domaine est composé d'un certain nombre de cours, et que chaque cours peut être repris dans un nombre quelconque de domaines;

HISTORIQUE, qui exprime si un étudiant a réussi ou échoué (booléen) dans un domaine au cours d'une année t.

SCHEMA CONCEPTUEL DES DONNEES

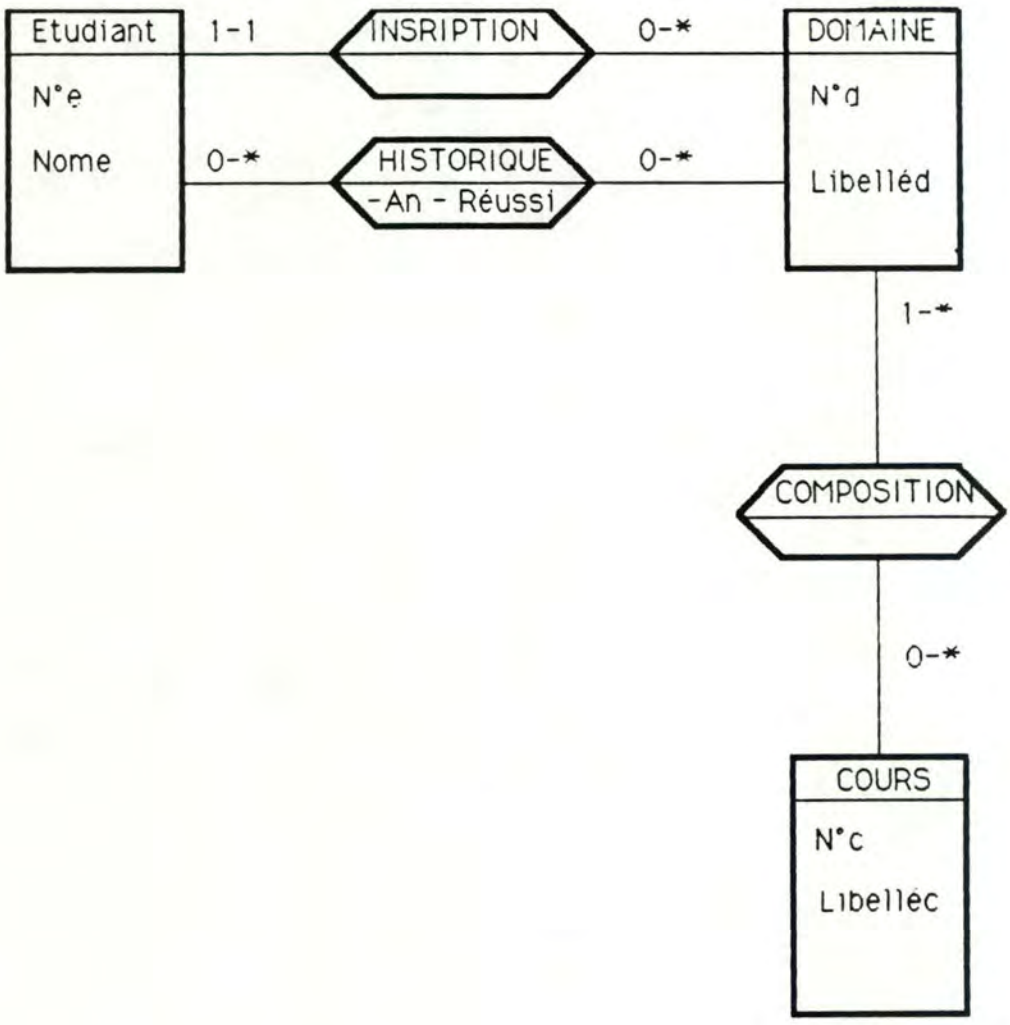


Fig. II.2 : schéma conceptuel des données - exemple

SCHEMA DES ACCES NECESSAIRES

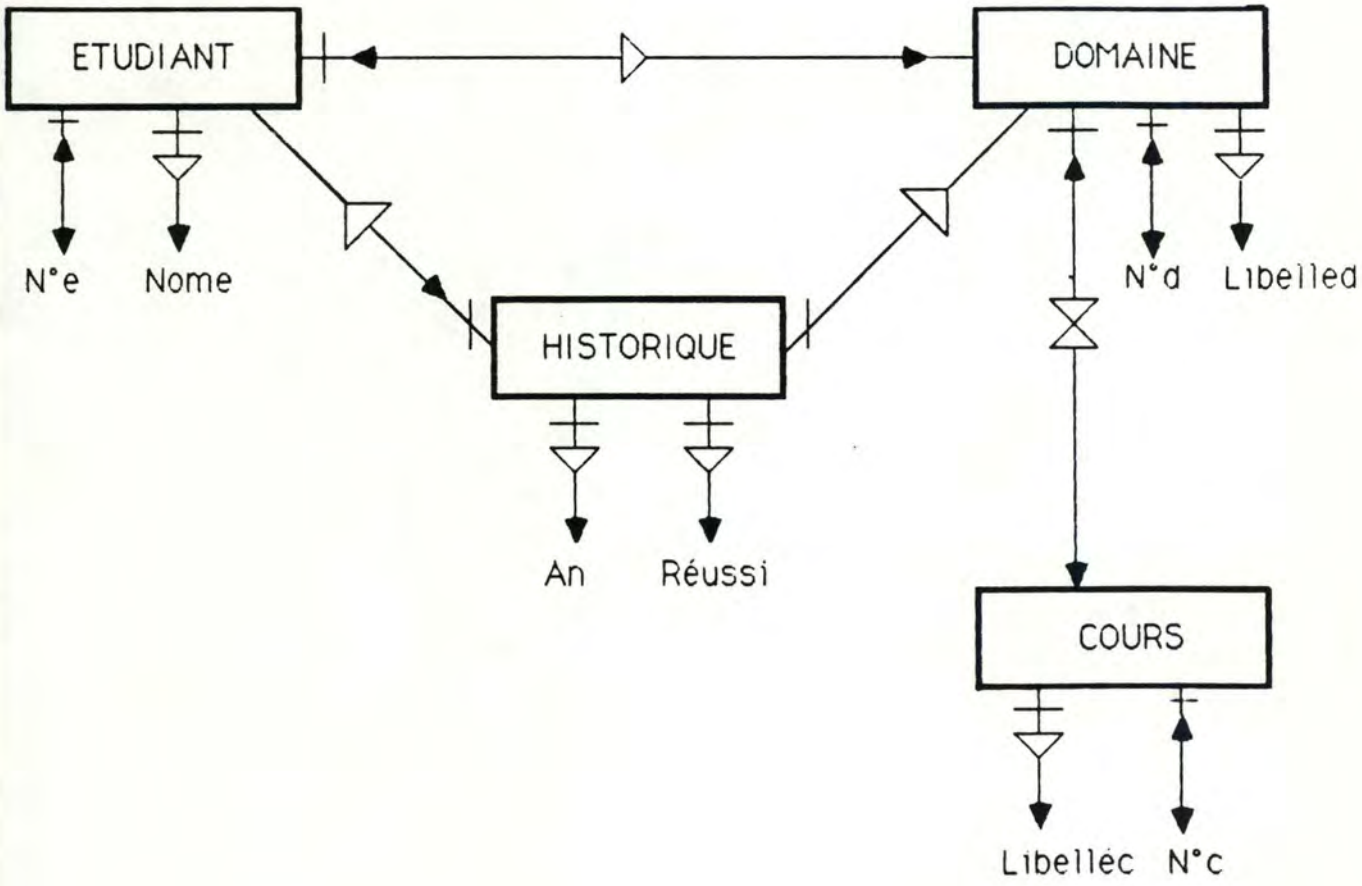


Fig. II.3 : schéma des accès nécessaires - exemple 1

a) Le modèle hiérarchique (ex : IMS d'IBM). Un SGBD hiérarchique représente les données suivant une structure d'arbre. Ainsi, à un article peuvent être associés 0, 1 ou plusieurs articles fils, mais un seul article père.

Etant donné que le modèle hiérarchique n'admet pas les associations N-N, le schéma des accès nécessaires conforme au hiérarchique sera le suivant :

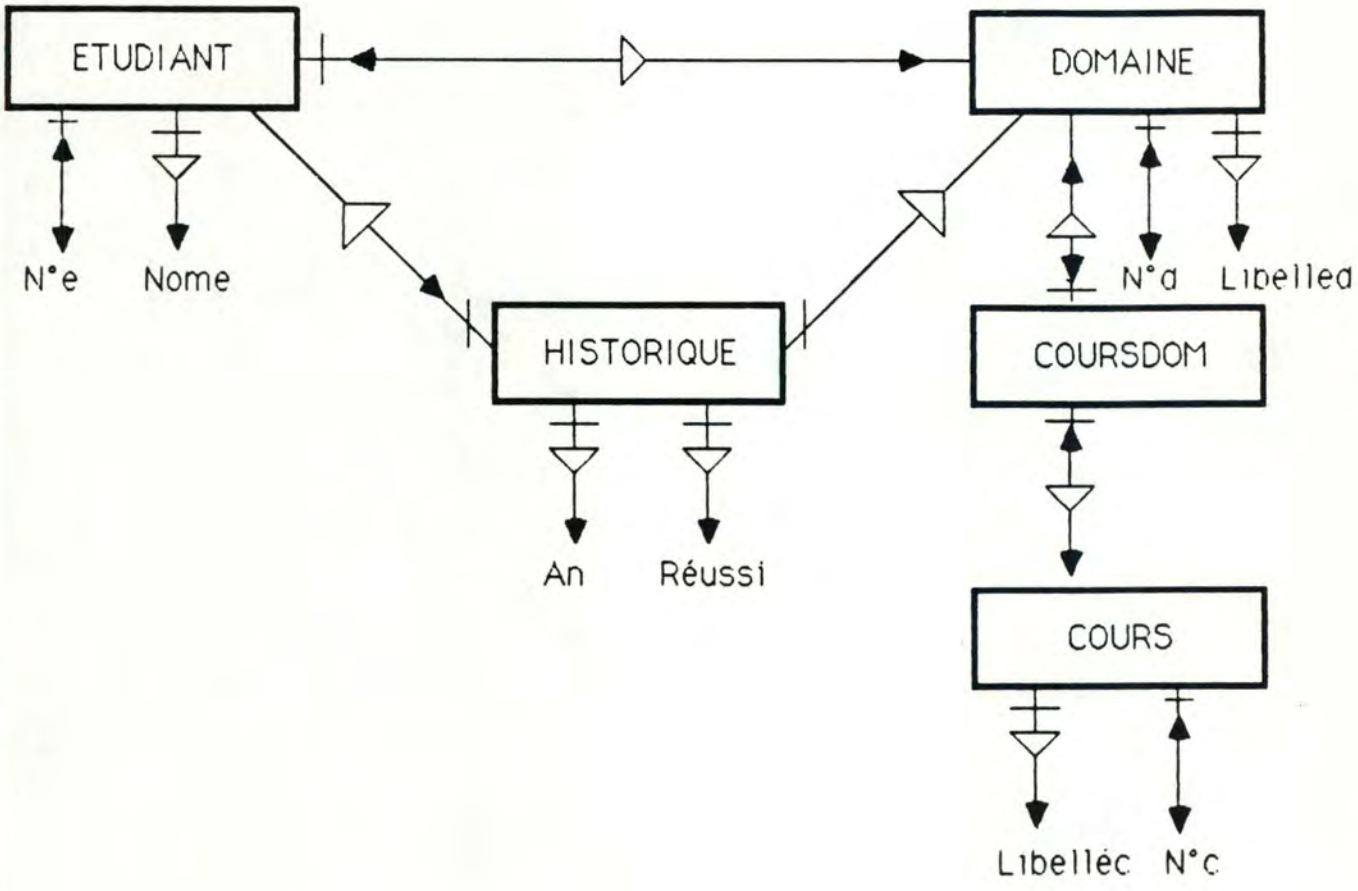


Fig II.4 : schéma des accès nécessaires - exemple 2

Nous pouvons encore le représenter sous forme de diagrammes de Bachman :

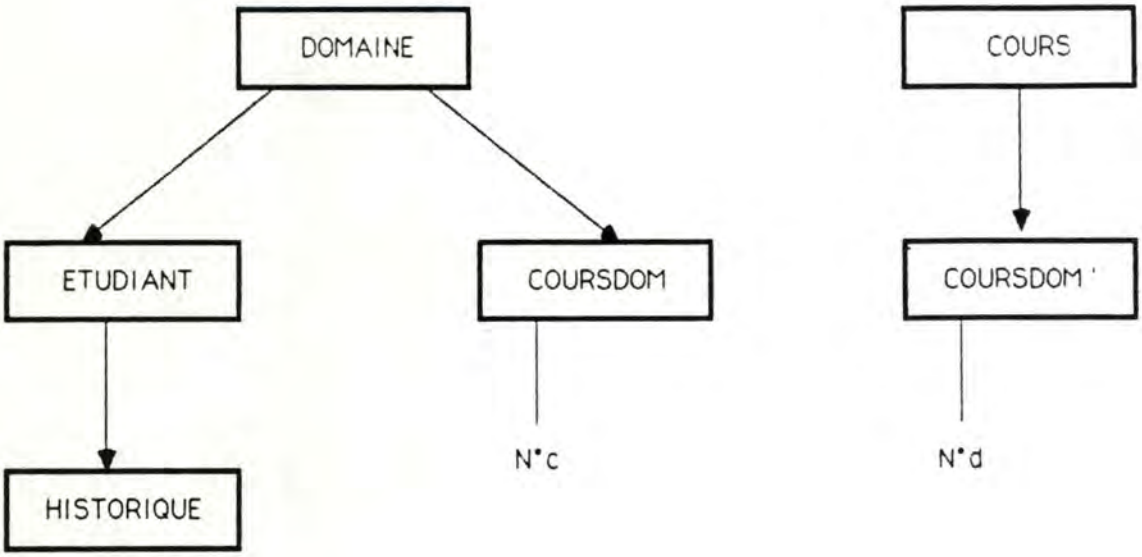


Fig II.5 : le modèle hiérarchique - exemple

L'association qui exprime qu'un étudiant a choisi un domaine particulier peut être matérialisée en introduisant le numéro de domaine (N°D) dans le type ETUDIANT.

Remarquons que le résultat auquel on aboutit dépend de l'ordre dans lequel on choisit la racine des schémas hiérarchiques.

b) Le modèle réseau (CODASYL, SOCRATE). Un SGBD de type réseau permet de stocker des articles ou enregistrements logiques, mais aussi des liens entre ceux-ci. Un lien est la représentation d'une association entre deux types d'enregistrements; il offre un mécanisme d'accès qui, à partir d'un enregistrement r de type R, permet d'accéder aux enregistrements s,s',s",... de type S qui lui sont reliés. Il y a peu de contraintes sur les types de liens que l'on peut établir entre les articles.

En fait, la représentation du schéma conceptuel conforme au modèle réseau peut se faire sous la forme d'un graphe général, où les noeuds sont les types d'enregistrements, et les arcs, les associations. La

représentation sous forme de diagramme de Bachman de notre exemple pourrait être la suivante :

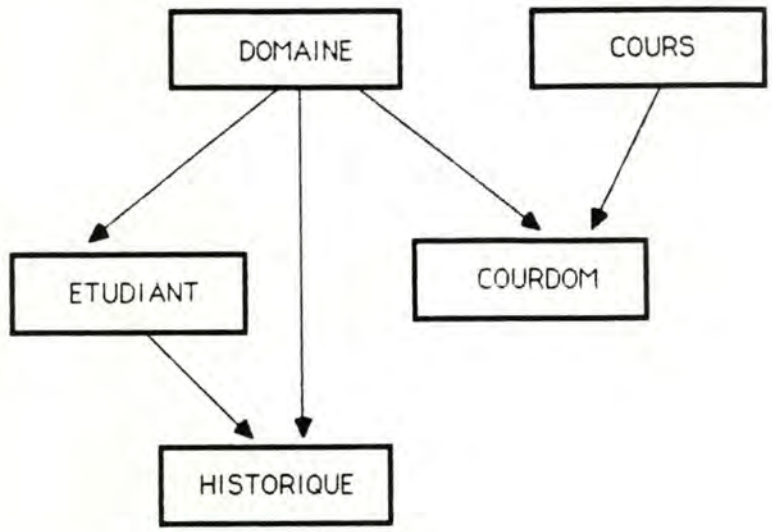


Fig. II.6 : **Le modèle réseau**

c) Le modèle relationnel. Un SGBD relationnel représente les données sous forme de tables organisées en lignes et colonnes. Dans un tel modèle, les structures de données utilisées sont très simples, et aucune référence à la façon d'accéder aux données ne doit être mentionnée, contrairement aux modèles hiérarchique et réseau qui définissent de façon explicite les chemins d'accès aux données. On parle ici de navigation automatique, car c'est au système de la base de données qu'il appartient de rechercher la solution conduisant aux performances les meilleures. Cette phase d'optimisation dans un SGBD relationnel est une des caractéristiques essentielles de ces systèmes, et conduit généralement à une dégradation des performances par rapport aux SGBD traditionnels, car d'une part le système doit consommer du temps machine pour effectuer cette tâche d'optimisation, et d'autre part les techniques développées ne permettent pas toujours de garantir que la solution optimale ait été trouvée.

Notre exemple pourrait ici se voir représenter de la façon suivante :

COURS

N°c	Libelléc

COURSDOM

N°d	N°c

HISTORIQUE

N°e	N°d	An	Réussi

ETUDIANT

N°e	Nome	N°d

DOMAINE

N°d	Libelléd

Fig. II.7 : **Le modèle relationnel; les tables.**

La plupart des SGBD relationnels offrent en outre des langages de manipulation de données à la fois simples et puissants, et accessibles à l'utilisateur final. Dans cette optique, un SGBD relationnel constitue très souvent la clef de voûte d'un environnement de quatrième génération.

2.2.2. Les micro-ordinateurs

"Les ordinateurs individuels s'infiltrent partout ! En cause, essentiellement, l'abaissement spectaculaire de leur prix". Ces deux affirmations constituent sans aucun doute un cliché maintes fois répété; elles ne sont pas moins révélatrices d'un état de faits. C'est ainsi que, lassé d'attendre une application sans cesse promise par le service informatique, l'utilisateur final a la volonté de développer avec les moyens du bord l'outil informatique indispensabe à sa gestion.

Cette tendance à la multiplication des PC conduit à une sorte de démystification de l'informatique, autrefois jugée incompréhensible et inabordable. Les informaticiens ne sont plus - ou tendent à le devenir moins - ces hommes marginaux enfermés dans leur tour d'ivoire. Si l'utilisateur final et le professionnel de l'informatique ne parlent pas encore tout à fait le même langage, ils commencent néanmoins à trouver un terrain de compréhension mutuelle. Tout est en place, en principe, pour que, du choc des PC naisse un dialogue constructif entre utilisateurs et informaticiens.

Mais cette évolution ne se fait pas sans heurts. D'un côté, quelques informaticiens voient d'un mauvais oeil des "bits de pouvoir" s'échapper de leur système, de l'autre, les utilisateurs regrettent de ne pas être passé à l'acte plus vite et ont bien l'intention de rattraper le temps perdu.

Et puis il y a "ceux qui n'ont pas suivi", se jugeant trop âgés, incompétents, ou encore peu motivés par la chose. Ceux-là voient le mur les séparant de la science informatique devenir plus infranchissables que jamais...

A ces facteurs humains, non négligeables, viennent se greffer un certain nombre de problèmes techniques. Car l'utilisateur s'est rendu compte que toutes les informations dont il avait besoin étaient en fait disponibles sur l'ordinateur central, et qu'une connexion réseau se révélait dès lors indispensable. Suit alors le cortège d'incompatibilités de données, de protocoles et de normes. Le budget, quant à lui, devient trop étroit; "l'abaissement spectaculaire des prix" se transforme en gouffre financier...

Cette vision des choses peut paraître quelque peu pessimiste, ou à tout le moins fort caricaturale; elle vise à rappeler que donner à l'utilisateur final - par l'intermédiaire de son PC - un rôle actif dans un système

d'information ne va pas sans poser un certain nombre de problèmes. Ceux-ci ne sont cependant pas insolubles : le développement d'un SGBD et d'un réseau, en même temps que l'introduction d'un L4G (portable sur PC) devraient permettre d'aborder ces difficultés avec une plus grande sérénité.

Bref, le défi qui se pose à nombre d'entreprises qui ont vu les micro-ordinateurs s'infiltrer dans tous les départements est d'exploiter efficacement ces PC en concert avec l'ordinateur central et les logiciels de quatrième génération, de manière à réduire le retard en matière de développement d'applications, et à fournir aux cadres un accès direct et immédiat à l'information dont ils ont besoin pour assurer une gestion efficace de l'organisation.

2.2.3. Les réseaux

Nous avons évoqué, dans la section précédente, les problèmes auxquels serait confronté un utilisateur désireux d'exploiter, à partir de son micro-ordinateur, les données de l'ordinateur central. Un de ces problèmes concernait les moyens de télécommunications.

Les problèmes liés à la communication entre systèmes hétérogènes sont de plusieurs ordres. A titre exemplatif, citons la vitesse limitée des supports de transmission, les erreurs de transmission sur ces supports, la différence entre les puissances des machines utilisées.

A ces problèmes techniques s'en ajoutent d'autres, d'ordre logiciel. Lorsqu'on désire connecter un micro sur un mini ou sur un gros ordinateur, il est fréquent de constater que le logiciel du micro est complètement différent de celui des machines plus puissantes. Cette incompatibilité logicielle a longtemps rendu difficile l'exploitation sur micro des données stockées sur plus grosses machines. L'échange des programmes, quant à lui, était quasiment impossible. Actuellement, plusieurs fabricants de langages de quatrième génération proposent des logiciels pratiquement universellement transportables, du micro vers le mainframe, en passant par le mini; ce problème d'incompatibilité se pose donc avec moins d'acuité.

Il est évident qu'au sein d'une entreprise d'une certaine importance, il est difficile d'envisager une base de données centrale, accessible à un grand nombre

d'utilisateurs, sans avoir recours à la technique des réseaux. En particulier, quand la distance entre les équipements est faible (quelques centaines de mètres), on parle de réseaux locaux, qui se distinguent des réseaux à grande distance par leur fiabilité, leur faible taux d'erreur, la vitesse élevée des rafales d'informations. Un réseau local peut être physiquement réalisé au moyen d'un câble coaxial (réseau décentralisé en bus de type Ethernet), ou au moyen de fils téléphoniques reliés à un autocommutateur (réseau étoilé de type PABX), ou encore au moyen de fibres optiques. Le modèle général d'un réseau local est présenté à la figure II.8.

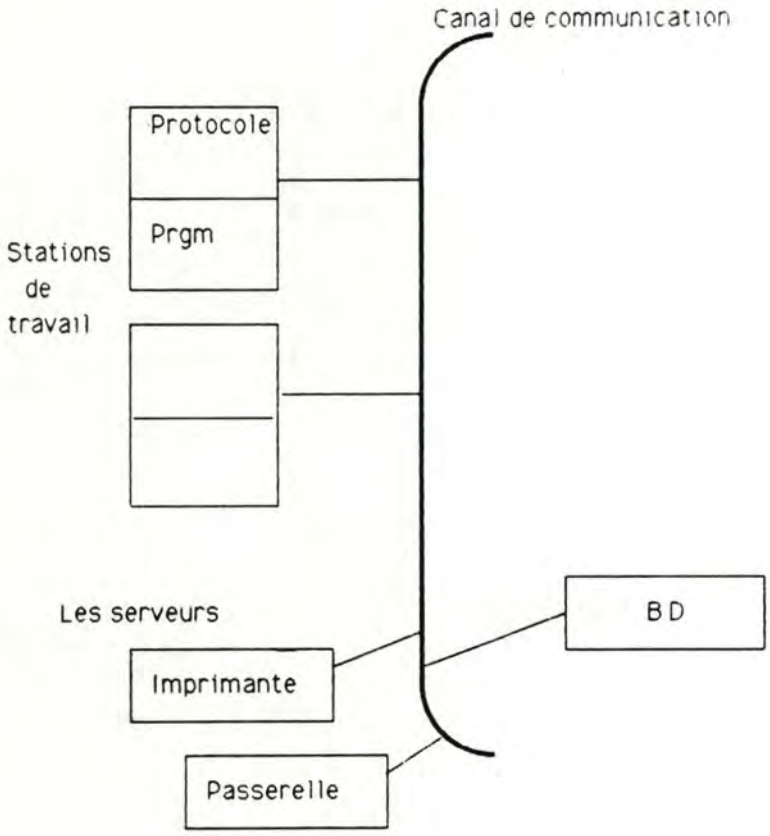


Fig. II.8 : **Modèle général d'un réseau local**

Dans le domaine de la consultaion de bases de données à grande distance, il existe notamment le service DIANE (Direct Information Access Network for Europe), accessible par le réseau téléphonique commuté ou le réseau DCS (Data Communication Service), et permettant l'accès à des bases de données partout en Europe.

Dans les années à venir, la RTT envisage de mettre sur pied le Réseau Numérique à Intégration de Service (RNIS), qui offre la possibilité de connecter à partir d'un seul câble et d'une prise de courant unique (ce sont les promoteurs de ce projet qui l'affirment) téléphones, télétex, télécopieurs, terminaux de traitement de données, vidéotex et ordinateurs.

Tous ces développements, et bien d'autres encore, du domaine des télécommunications participent pleinement à la "philosophie quatrième génération" en ce sens qu'ils donnent à un grand nombre d'utilisateurs la possibilité d'accéder, de manière simple, efficace et rapide, à un nombre sans cesse croissant de données pertinentes à leurs yeux et ce, quelle que soit la localisation de ces données. Ce dernier aspect doit d'ailleurs rester parfaitement transparent à l'utilisateur...

2.2.4. La bureautique

La bureautique désigne "l'assistance aux travaux de bureau procurée par des moyens et des procédures faisant appel aux techniques de l'électronique , de l'informatique, des télécommunications et de l'organisation administrative" [DEBL 83].

Il est assez remarquable de constater que l'évolution de la bureautique s'inscrit dans la même philosophie que celle des langages de 4ème génération : l'automatisation des tâches de bureau résulte de la faible productivité du travail administratif, au même titre que l'introduction d'un L4G sera motivée en grande partie par le manque de productivité des informaticiens et des utilisateurs.

L'augmentation des salaires des cols blancs et l'inflation des exigences en matière de service ont gonflé très fortement les coûts liés au fonctionnement des bureaux.

C'est ainsi qu'après avoir résolument engagé l'amélioration de la productivité de son secteur secondaire, l'entreprrise fait le constat que la gestion

de son tertiaire devient une priorité; elle se dirige alors sur la voie de la bureautique, mais il convient de pouvoir maîtriser harmonieusement cette évolution, car au-delà du phénomène technologique qu'elle implique, ce sont les facteurs humains qui seront les plus difficiles à contrôler...

Nous allons brièvement passer en revue un certain nombre d'outils que la bureautique met à la dispositon de l'utilisateur.

. **Le traitement de texte** : ce terme désigne le concept d'automatisation des travaux de dactylographie, depuis la création d'un texte jusqu'à sa diffusion. Le traitement de texte permet donc toutes les opérations de saisie, corrections, manipulations, mise en forme, mémorisation, édition et transmission de tout document manipulé dans un bureau. Des fonctions "intelligentes" telles les dictionnaires de correction de l'orthographe, la traduction automatique, ... prennent progressivement un rôle de plus en plus important.

. **La messagerie électronique** : il s'agit des services de transmission de messages à distance entre terminaux, avec stockage de ces messages dans une "boîte aux lettres" électronique. La communication peut ainsi être différée, le destinataire n'étant pas nécessairement présent lors de la transmission. Il peut donc prendre connaissance des messages reçus en son absence par consultation de sa "boîte aux lettres".

. **L'archivage électronique** : c'est un procédé permettant de mémoriser et de rendre accessibles des documents de toute nature, après les avoir préalablement enregistrés sur des supports magnétiques ou optiques.

. **La télécopie** : il s'agit de la "photocopie à distance", permettant la transmission de documents papier entre deux points distants. L'appareil émetteur assure une lecture et un codage du document à transmettre sur la ligne; l'appareil récepteur décode et imprime le document transmis.

. **Audio- et vidéoconférence** : l'audioconférence est un procédé permettant d'éviter la limitation du téléphone à deux correspondants. Elle permet ainsi la concertation entre un groupe de personnes de chaque côté de la ligne, évitant ainsi les déplacements. La vidéoconférence présente la particularité supplémentaire de voir l'image, sur un écran TV, des interlocuteurs se trouvant dans un studio distant.

. **etc...**

Bien que séduisants au premier abord, les systèmes bureautiques ne sont pas dépourvus de toute contrainte, en ce sens qu'ils présupposent des investissements massifs en matériel, d'abord, en formation humaine, ensuite.

Les langages de quatrième génération, quant à eux, peuvent être vus comme le prolongement naturel de l'"effort bureautique" consenti par une organisation, car ils ont pour but de faciliter ou de rendre plus performant l'accomplissement de certaines tâches de bureau de "haut niveau", en même temps qu'ils devraient permettre d'accroître la productivité de certains acteurs jusqu'à présent peu touchés par les efforts de rationalisation de l'organisation : les informaticiens.

2.2.5. Les ateliers logiciels

Un atelier logiciel est un ensemble intégré d'outils complémentaires destinés à couvrir, du moins en partie, le processus de production des applications. Ces outils permettent

- d'introduire et de mémoriser, à l'aide d'un langage approprié, les spécifications d'un système d'information dans une base de données des spécifications;

- d'interroger la base de données et d'effectuer des contrôles de cohérence des descriptions introduites;

- d'évaluer les spécifications selon des critères de performances, de consommation de ressources, et d'adéquation par rapport aux besoins des utilisateurs. Dans cette optique, les outils de simulation permettent d'évaluer de manière expérimentale l'efficacité d'un système d'information et de prévoir ses performances compte tenu d'un environnement donné et ce, avant son implémentation. De même, les outils de prototypage peuvent générer des "modèles réduits" du système d'information qui permettent de vérifier que les spécifications rencontrent bien les besoins exprimés par les utilisateurs.

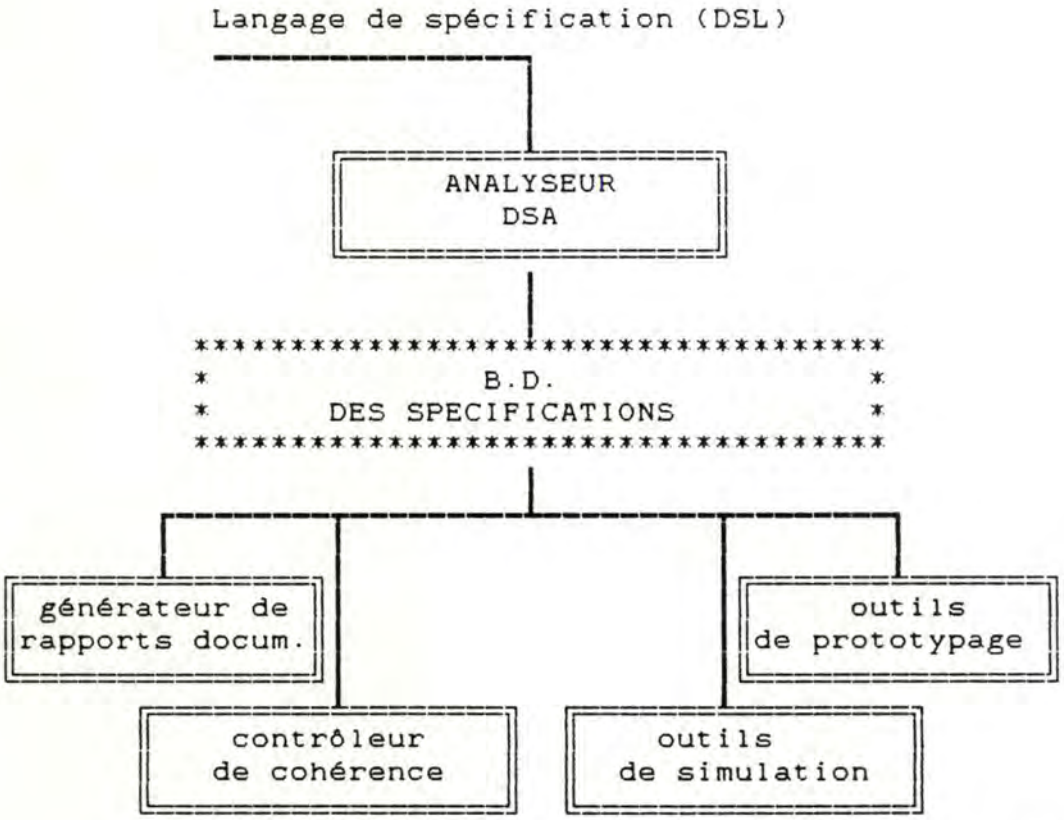


Fig II.9 : **Flux d'information au sein de l'atelier logiciel IDA [BHLP].**

En outre, une fois les spécifications bien établies et validées, l'atelier logiciel pourra guider la transformation des spécifications des traitements et des données en programmes et en bases de données. Néanmoins, une automatisation trop forte à ce niveau risquerait de rendre rigide le fonctionnement du système d'information. C'est pourquoi l'intervention humaine n'est pas du tout exclue, mais est guidée par l'atelier logiciel qui prévient toute distorsion excessive entre le système réel et le système spécifié.

La différence essentielle entre un système de quatrième génération et un atelier logiciel réside dans leur "philosophie" : dans le premier cas, on a affaire à un environnement très fermé, disposant d'interfaces très conviviales; dans l'autre, on a affaire à un environnement tout à fait ouvert, permettant de travailler pratiquement au niveau Entité/Relation, et où l'informaticien construit lui-même ses propres outils d'accès aux données.

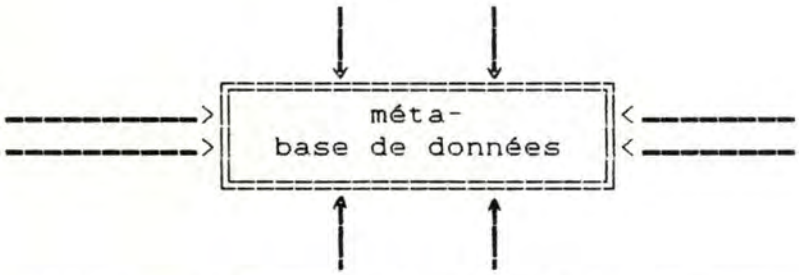


Fig. II.10 : **Philosophie "Atelier Logiciel" : on décrit l'application sous forme d'un modèle; on crée autant de fonctions que l'on veut, on agrandit l'environnement à la demande.**

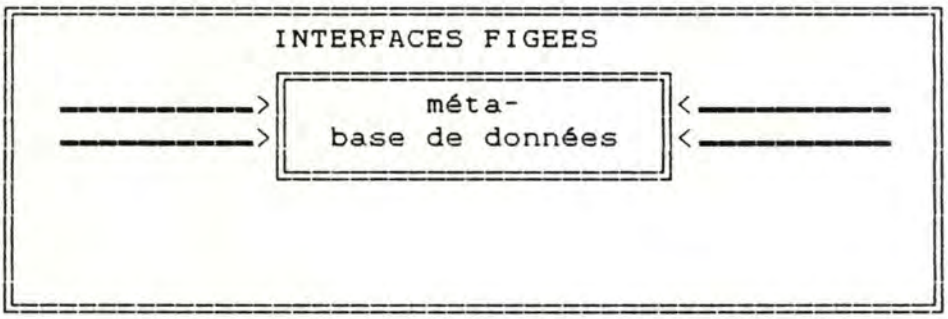


Fig. II.11 : **Philosophie L4G : on travaille de manière figée, en utilisant les interfaces existantes.**

En fait, le but de chacune des deux approches est différent. Un atelier logiciel est un outil d'aide à la conception d'applications, destiné à automatiser les tâches répétitives de l'informaticien, et de lui seul. Un L4G est un outil convivial de développement d'applications, souvent abordable par un utilisateur final.

2.2.6. Le concept d'infocentre

Le concept d'infocentre est étroitement lié à la notion de langage de quatrième génération. En effet, il se veut être une structure d'accueil pour l'utilisateur final, conçue au sein du service informatique. Son but est de servir directement et rapidement l'utilisateur

final : les consultants de l'infocentre créent, avec l'utilisateur, les applications que celui-ci désire, ou mettent à sa disposition les moyens nécessaires à la satisfaction de ses besoins informatiques.

Les principaux moyens mis à la disposition de l'utilisateur sont les outils graphiques permettant de générer des graphiques simples, les générateurs d'états imprimés, les générateurs d'écrans, et surtout les langages conviviaux d'interrogation de BD.

Lorsque l'utilisateur a des besoins plus complexes, et ne faisant pas partie de la stratégie globale et planifiée de l'entreprise, il fait appel au consultant qui s'asseoit avec lui au terminal. Ils essayent alors ensemble de développer le système d'aide à la décision ou la procédure d'interrogation personnalisée dont l'utilisateur a besoin, tout de suite.

Cette manière de procéder évite de devoir passer par le cycle formel du développement classique d'une application, ce qui aurait nécessité, dans le meilleur des cas, plusieurs jours, voire plusieurs semaines de délai... Or, l'application dont l'utilisateur a besoin ne présente un intérêt que si elle est exploitable rapidement.

D'ailleurs, si l'utilisateur avait dû se contenter de la "filière classique", il n'aurait sans doute même pas formulé sa demande au service infomatique, car il ne sait que trop bien que celui-ci est "surchargé de travail en ce moment"...

A titre d'exemple (non nécessairement représentatif), nous citerons APL, comme langage d'un infocentre envisageable pour une certaine classe d'utilisateurs : en particulier les ingénieurs et les financiers, qui manipulent beaucoup de chiffres et formules. En effet, APL a été conçu, à l'origine, comme un langage mathématique, dont les qualités essentielles étaient la concision dans l'écriture des programmes et la capacité à manipuler des matrices. Ainsi, dans un premier temps, APL peut être proposé à des utilisateurs finals en remplacement de leur calculateur programmable. L'avantage est que l'utilisateur, une fois introduit dans le "monde APL", peut découvrir certaines extensions de ce langage (avec APL PLUS ou ADRS, par exemples) lui permettant d'interroger une base de données, de mettre à jour des fichiers, de produire des rapports et des diagrammes simples...

La vision de l'infocentre que nous venons de présenter peut paraître idyllique. Néanmoins, il serait

injuste de passer sous silence un certain nombre de problèmes que peut provoquer ce type d'organisation.

Tout d'abord, l'exploitation par l'utilisateur d'outils informatiques parfois très puissants nécessite une importante formation et un bon entraînement. Trop de légèreté dans ce domaine conduirait inévitablement à une dégradation de la productivité humaine et matérielle.

Ensuite, le développement anarchique de nouveaux programmes et de nouvelles données peut amener redondance et incompatibilité. Il y a donc lieu de tenir à jour un catalogue des applications et des bases de données, de manière à contrôler et à canaliser le développement par l'utilisateur de nouvelles applications.

Section 2.3 : L'approche 4ème génération dans un cadre opérationnel

2.3.1. Productivité des programmeurs

Nous avons expliqué, au chapitre 1, que la croissance de la demande en matière de développement d'applications dépassait plus que largement l'accroissement de la capacité de production du service informatique. La conséquence inéluctable de cette situation était que la file d'attente devant le service informatique ne cessait de s'allonger, en même temps d'ailleurs - et c'est une conséquence - que les retards dans la livraison des applications.

Il y a deux approches complémentaires pour aborder ce problème : la première consiste à mettre l'utilisateur final en contact direct avec l'information stockée sur disque; la seconde vise à accroître la productivité des professionnels de l'informatique. C'est de cette dernière approche dont il va être question dans cette section.

Une première façon d'améliorer la productivité des programmeurs est de travailler de manière plus structurée et plus rigoureuse, suivant une conception modulaire. La complexité et la dimension croissantes des programmes d'application a rendu nécessaire, en effet, la découpe de ces programmes en modules, suivant les principes exposés dans le premier chapitre.

Une deuxième approche, non exclusive par rapport à la première, consiste à faire usage d'un langage de très haut niveau. En effet, le passage de l'assembleur à un langage de troisième génération avait amélioré la productivité des programmeurs; il devrait en être de même lors du passage du COBOL ou du PL1 à un langage de quatrième génération : ce dernier permet souvent d'écrire en quelques lignes une opération qui en nécessiterait plusieurs dizaines en COBOL. De plus, les L4G permettent à l'informaticien de raisonner à un niveau plus élevé, c'est-à-dire plus proche de l'analyse conceptuelle. Nous verrons, dans la deuxième partie de cette étude, si ces assertions se trouvent vérifiées par les utilisateurs...

Langage de 4ème génération et approche relationnelle

Un L4G est toujours associé à un SGBD. Celui-ci peut être hiérarchique, en réseau, ou relationnel. Nul doute cependant que c'est ce dernier type qui cadre le mieux avec la philosophie 4ème génération. La raison la plus importante à cela est que le modèle relationnel réalise mieux que les autres l'objectif d'indépendance des données, c'est-à-dire l'établissement d'une frontière bien nette entre les niveaux logique et physique.

Ainsi, la nécessité pour les programmeurs de naviguer au travers de chemins d'accès prédéfinis pour atteindre les données désirées n'existe plus : les systèmes relationnels assurent une navigation automatique. C'est là un facteur non négligeable d'amélioration de la productivité.

De plus, dans un système relationnel, toutes les informations sont représentées par des valeurs dans des tables. C'est ainsi que l'accès aux données se fait sur base de leur valeur, et non en fonction de leur position. L'opérateur de sélection (SELECT) de l'algèbre relationnelle prend une relation (TABLE) comme opérande et produit une nouvelle relation constituée d'articles (LIGNES) de la première. L'opérateur de projection (PROJECT) construit une nouvelle relation constituée d'attributs (COLONNES) de la première. L'opérateur d'équi-jointure (EQUI-JOIN) prend deux tables comme opérandes et en produit une troisième, constituée de lignes de la première concaténées avec des lignes de la seconde, et parmi lesquelles on ne retient que celles dont les colonnes spécifiées ont des valeurs communes dans les deux premières tables.

Etant donné que le modèle relationnel traite les relations comme des opérandes, la notion traditionnelle de

boucle est généralement peu utile pour les problèmes simples; c'est là un autre facteur d'amélioration de la productivité des programmeurs.

2.3.2. Facilité de développement et souplesse d'utilisation

Un L4G facilite le développement de nouvelles applications grâce, essentiellement, à son langage de très haut niveau. C'est déjà un gros avantage. Pourtant, dans de nombreux cas, il ne suffit pas à un programmeur de développer rapidement une application. L'utilisateur final, en effet, sera rarement satisfait de la première version proposée par le programmeur, pour des raisons qui ont été développées par ailleurs dans cette étude.

Dès lors, au moment où l'informaticien fait la première démonstration de son programme, il doit s'attendre à un certain nombre de remarques émanant de l'utilisateur final. Il est rare que ces remarques concernent un fonctionnement intrinsèquement incorrect du programme; elles se rapportent plus généralement à une mauvaise adéquation du programmme aux attentes de l'utilisateur.

Il faut avoir vécu cette situation pour savoir qu'elle ne fait pas partie des moments agréables du métier de programmeur-analyste. Devoir modifier, de façon parfois fondamentale, la structure d'un programme minutieusement établie pendant plusieurs semaines n'est pas très plaisant !...

Dès lors, il s'agit de disposer d'outils permettant d'ajuster rapidement le programme aux besoins "revus et corrigés" de l'utilisateur final. La plupart des L4G revendiquent cet atout : autoriser une grande souplesse en matière d'adaptation rapide de programmes existants.

L'amélioration de la souplesse d'utilisation peut avoir deux conséquences positives, la deuxième découlant directement de la première. Tout d'abord, elle implique une amélioration de la qualité du travail fait par l'analyste-programmeur : le programme revu et corrigé coïncide mieux avec les exigences de l'utilisateur. Ensuite, ce dernier n'a plus ce sentiment de frustration qui était le sien lorsqu'il s'entendait dire par le programmeur : "si vous vouliez autre chose, il fallait y penser avant"...

Bref, le climat général s'en trouve amélioré, et l'informaticien remonte dans l'estime de l'utilisateur...

2.3.3. Maintenance et documentation

L'aspect "maintenance" représente généralement une part importante du budget informatique d'une organisation. Tout facteur susceptible de réduire son coût sera donc toujours vu d'un très bon oeil par le directeur financier.

Plusieurs moyens sont à la disposition du service informatique pour réduire, ou du moins faciliter la maintenance des programmes.

Tout d'abord, le service informatique peut se montrer rigoureux sur la documentation des programmes. Il peut exiger que celle-ci soit abondante, et corresponde à des "standards" définis par lui, de telle sorte que n'importe quel programmeur puisse se sentir à l'aise dans la correction d'un programme rédigé par un autre. La plupart des L4G offrent des outils de documentation, dont la principale qualité est la convivialité. En aucun cas, cependant, il ne contraindront l'utilisateur à rédiger sa documentation suivant des normes strictes. Dans de nombreuses entreprises, toute nouvelle application développée se voit attribuée un dossier d'analyse, répondant point par point à une sorte de "table des matières" établie par la direction informatique et connue de tous les programmeurs. Les L4G n'apportent donc pas ici un avantage décisif par rapport aux langages classiques de programmation.

Ensuite, l'utilisation de bases de données, en lieu et place d'un environnement fichiers, rend possible la modification de la structure physique des données, sans devoir simultanément revoir tous les programmes s'appuyant sur cette structure. C'est là un élément déterminant dans la réduction d'une certaine forme de maintenance, à condition bien sûr que l'environnement base de données ait été bien conçu. Notons que cet avantage n'est pas spécifique des L4G, mais se retrouve dans tout environnement base de données, quel que soit le langage.

Enfin, il est possible de réduire la maintenance des programmes en utilisant des langages de très haut niveau

comme outils de développement. Et c'est ici que les L4G gagnent du terrain sur les langages comme COBOL : ils permettent de modifier les programmes d'application existants de manière rapide et simple.

Dans cette optique, il est souvent intéressant, pour une organisation qui se serait équipée d'un L4G, de développer à nouveau les anciennes applications (celles qui avaient été développées en L3G) à l'aide de leur nouvel outil, de manière à éliminer la nécessité d'encore programmer en L3G lors des travaux de maintenance. Cette stratégie se justifie par le fait que le coût de la nouvelle création d'une application avec un langage de très haut niveau est souvent inférieur à celui qu'entraînerait la continuation de son exploitation avec maintenance...

2.3.4. Rapidité de la formation de base

L'acquisition d'un L4G est un investissement qui, comme tout investissement consenti au sein d'une entreprise bien gérée, doit se voir justifier par un taux de rentabilité économique suffisant. Nous reviendrons sur cet aspect financier dans le chapitre 3 de cette étude.

Il faut souligner, cependant, qu'aux investissements matériels et logiciels s'ajoutent toujours des investissements humains : c'est tout le problème de la formation des développeurs.

Un des fers de lance des L4G est précisément le caractère réduit de cette formation de base : quelques jours de cours intensifs suffisent généralement à procurer aux développeurs une certaine aisance dans la manipulation de leur nouvel outils. Après quelques semaines de pratique, les meilleurs programmeurs devraient déjà atteindre un rendement optimal. On est très loin du COBOL, et plus encore de l'assembleur, qui nécessitaient plusieurs mois, voire plusieurs années d'expérience avant d'en arriver à une maîtrise partielle !...

La plupart des fabricants de L4G proposent des stages de formation pour programmeurs-analystes, mais leur coût est généralement assez élevé. Seules les plus grosses entreprises, dotées d'un département informatique important, consentiront à la dépense pour quelques-uns de leurs développeurs. Le rôle de ceux-ci sera alors de transmettre à leurs collègues les connaissances acquises à l'extérieur.

Beaucoup de programmeurs, aussi, sont autodidactes; ils (ou leur entreprise) jugent trop onéreux les cours dispensés par le fabricant, et estiment qu'un tel type de formation ne met pas suffisamment en évidence les éventuelles faiblesses du langage qu'ils sont appelés à utiliser. Ils préfèrent découvrir par eux-mêmes les subtilités propres à leur nouvel outil.

Le contenu de la formation sera essentiellement technique et pratique (utilisation correcte du L4G), mais pourra se voir complété par une partie théorique et/ou méthodologique. Par exemple, il peut être très utile de procurer à certains programmeurs quelques rudiments de la théorie relationnelle, lors de l'acquisition d'un SGBD relationnel et d'un L4G associé.

De toute manière, quel que soit le type de formation adopté, celle-ci sera en principe largement moins fastidieuse que celle nécessitée par des langages de troisième génération.

2.3.5. Standardisation des interfaces

Une des qualités essentielles d'un langage de quatrième génération est qu'il doit procurer des outils intégrés. C'est ainsi qu'un bon L4G doit pouvoir exploiter les possibilités de mise à jour de la base de données et de génération de rapports ou d'écrans en utilisant le même style de dialogue et de syntaxe.

Pour utiliser une nouvelle fonction offerte par le L4G, le programmeur n'aura pas à assimiler une brochure spécifique, puisque toutes les fonctions forment un environnement logiciel intégré, et que toutes travaillent en conjonction avec la base de données associée au L4G.

Cet aspect de standardisation des interfaces contraste assez fort avec certains environnements de troisième génération, où les fonctions offertes sont parfois nombreuses (gestionnaire d'écrans, générateur de rapports, fonctions graphiques,...), mais aussi et surtout très dispersées et peu homogènes entre elles.

2.3.6. Rôles des utilisateurs et des informaticiens

Le chapitre 1 de cette étude présentait notamment l'état d'esprit dans lequel doivent travailler la plupart des programmeurs-analystes : ils doivent satisfaire, aussi parfaitement que possible aux besoins informatiques d'utilisateurs n'ayant généralement pas exprimé ces besoins de manière claire, rigoureuse, complète et non ambiguë. D'ailleurs, les utilisateurs eux-mêmes ne connaissent généralement pas très bien leurs besoins avant d'avoir vu ce que l'informaticien était en mesure de leur proposer... C'est dire combien cette situation est proche de la quadrature du cercle !

De plus, même dans le cas idéal et rarissime où l'utilisateur peut, au départ, exprimer parfaitement ce qu'il désire, il n'est pas rare de constater qu'il se sent amené à nuancer quelque peu ses propos à mesure que des solutions concrètes lui sont proposées. A première vue, ces problèmes relèvent plus de la psychologie humaine que de l'informatique lourde; pourtant, cette dernière doit tenir compte du caractère mouvant de l'esprit humain.

Ainsi, l'informatique doit s'adapter; le développeur professionnel doit abandonner ses préjugés et ses anciens réflexes. Plutôt qu'exiger de l'utilisateur final son paraphe au bas du document des spécifications et s'enfermer pendant plusieurs semaines pour confectionner un programme "sur mesures", il doit essayer de rendre plus actif le rôle de l'utilisateur durant tout le cycle de développement de son application. Cette manière de procéder lui évitera à coup sûr de devoir remodeler son programme, alors qu'il le croyait tout à fait au point et prêt à être exploité.

Nous avons vu que les L4G permettaient de revoir et d'adapter un programme beaucoup plus rapidement qu'un L3G; l'objectif ici est d'aller plus loin encore, en exploitant toutes les possibilités d'un langage de très haut niveau à chaque étape du cycle de développement d'une application, de façon à réduire au minimum, voire à supprimer complètement toute "retouche" ultérieure. L'idée est que, si l'utilisateur participe à chaque étape du cycle de développement d'une application, le risque de correction en amont s'en trouve réduit, et la modification éventuelle sera moins onéreuse que si elle avait dû être faite à un stade plus avancé du cycle de développement.

C'est ici qu'intervient la notion de prototype. Par analogie, dans l'industrie automobile, par exemple, jamais un modèle ne sera mis sur le marché avant d'être passé par

le stade de prototype. Un prototype est ici une voiture à part entière, disposant de toutes les caractéristiques prévues lors de sa conception, et destinée à être expérimentée sur la route afin de déceler toutes les failles qui n'auraient pas pu être mises en lumière au stade de la conception. Ce prototype sera mis à rude épreuve durant de longs mois, et de très nombreuses modifications y seront apportées avant le passage à la fabrication en grande série, c'est-à-dire avant les gros frais.

Le concept de prototype dans les sytèmes de traitement de données est fort similaire : la voiture est ici un programme, et le lieu d'expérimentation, le bureau de l'utilisateur final. Il s'agit de proposer à ce dernier une image concrète de ce que pourrait être l'application qu'il a commandée; assis devant son écran, l'utilisateur pourra alors suggérer, en connaissance de cause, l'une ou l'autre modification. Le développeur pourra, quant à lui, rapidement ajuster son prototype aux besoins de l'utilisateur, et lui en proposer une nouvelle version, revue et corrigée. Ce processus est itératif et se poursuivra jusqu'à ce qu'il y ait correspondance parfaite entre le prototype et les attentes de l'utilisateur.

Avant l'appartition des L4G, l'usage des prototypes était chose très peu courante, car le coût de leur programmation était aussi élevé que celui de la programmation du système proprement dit. Actuellement, la plupart des L4G proposent des fonctions permettant de réaliser des prototypes de façon relativement économique. Par exemple, il est possible de créer rapidement des dialogues d'interrogation de bases de données et de manipulation d'informations à l'écran, des présentations de listings, etc... L'analyste propose et l'utilisateur dispose : le rôle du premier devient celui d'un consultant qui essaye de cerner les besoins de son client.

A mesure que l'analyste et l'utilisateur communiquent, le prototype devient le centre du débat; ils peuvent s'assurer alors qu'ils parlent tous deux de la même chose. L'utilisateur est invité à réfléchir quelque temps à la solution qui lui est proposée sous forme du prototype final. Après avoir marqué son accord définitif, il demande son application le plus tôt possible.

Dans certains cas, il pourra en disposer très vite, car la base de données existe, et le prototype peut devenir l'application finale. Cependant, lors de l'élaboration du prototype, l'analyste aura complètement ignoré les questions de volume des transactions et de performances de la machine. C'est ainsi que des travaux

de mise au point sont souvent nécessaires pour, par exemples, assurer le rendement, la sécurité, les validations syntaxique et sémantique, la création de la BD, etc... Dans ce cas, le prototype peut constituer le document des spécifications. L'avantage est que ce document peut être considéré comme très fiable, et pratiquement immuable. Cependant, à ce stade, beaucoup de choses encore restent à programmer !

Quoi qu'il en soit, les L4G bénéficient de ce gros avantage par rapport aux L3G de permettre à l'analyste de proposer rapidement une solution concrète aux problèmes de l'utilisateur final.

Section 2.4 : L'approche 4ème génération dans un cadre décisionnel

2.4.1. Qualités de l'information

Il est évident que, pour un gestionnaire, ou un cadre en général, disposer du maximum possible d'information avant de prendre une décision est d'une nécessité vitale : la marge de manoeuvre des décideurs, en effet, est d'autant plus réduite que le droit à l'erreur, s'il existe, s'accomode de moins en moins de sa forme plurielle !...

Mais pour être parfaitement exploitable par celui à qui elle est destinée, l'information doit présenter certaines qualités.

Tout d'abord, elle se doit d'être "présentable". Il est peu utile de donner, à chaque responsable, un listing aussi volumineux que peu clair, reprenant, certes, toutes les informations relatives à la société pour une période donnée, mais dont seul un faible pourcentage présente un intérêt pour chaque cadre pris individuellement. Le scénario classique est alors le suivant : chacun retire du listing les résultats qui l'intéressent, et en fait un résumé ... manuscrit ! La perte de temps est importante; la frustration des cadres ne l'est pas moins. Dès lors, il serait souhaitable de fournir à chaque décideur une information personnalisée, adaptée à ses besoins spécifiques, et présentée sous la forme qu'il privilégie : tableau de chiffres, courbes, histogrammes, graphiques de toutes formes et couleurs...

Une deuxième qualité, essentielle, d'une information "décisionnelle" est qu'elle doit pouvoir être obtenue rapidement. Ce critère de rapidité d'obtention est primordial, car l'information décisionnelle a pour caractéristique de ne présenter d'intérêt que si elle peut être obtenue dans des délais très courts : à quoi pourrait bien servir de connaitre l'implication sur le chiffre des ventes du produit X d'une diminution de dix pour cent de son prix et d'une augmentation de vingt pour cent de son budget publicitaire, si cette information n'est disponible que deux jours après que la décision marketing ait dû être prise ? C'est la raison pour laquelle le cycle lourd de l'informatique opérationnelle ne pourra généralement pas satisfaire ce type de contrainte.

La particularité de l'informatique décisionnelle par rapport à l'informatique opérationnelle consiste dans la satisfaction de besoins ponctuels, personnalisés, non planifiés et très urgents.

2.4.2. Procéduralité et non-procéduralité

Une solution, radicale mais très onéreuse, à l'inadéquation de l'informatique opérationnelle aux exigences de l'informatique décisionnelle serait d'assigner à demeure un informaticien dans le bureau de chaque décideur ! Cette situation est évidemment irréaliste...

Une autre solution consiste à confier à l'utilisateur final le soin de satisfaire lui-même une partie de ses besoins informatiques. Cette idée est fort proche du concept d'infocentre : mettre à la disposition des décideurs des outils logiciels leur permettant de développer, en interactif à partir de leur terminal ou micro, de petites applications ponctuelles et personnalisées.

Idéalement, le décideur devrait disposer d'un langage d'interrogation proche du langage naturel : "Donnez-moi, sur graphique, le taux de pénétration nationale, pour les six derniers mois, de notre produit X, et celui du produit équivalent de nos cinq plus gros concurrents; quel aurait été notre taux de pénétration présumé si nous avions consenti une baisse de prix de cinq pour cent, toutes choses égales par ailleurs". Cette question pourrait être

posée par le directeur du service commercial d'une grosse société.

Ce genre de question nécessite aussi une réponse immédiate; inutile, dès lors, d'envisager le passage par le cycle classique de développement d'une application : cette démarche nécessiterait trop de temps, d'abord. Ensuite, la multiplication du nombre d'intermédiaires (analystes ou programmeurs) augmenterait considérablement le risque d'erreur d'interprétation. Le gestionnaire seul connaît les nuances qu'il veut apporter à sa requête...

Certains fabricants proposent des logiciels d'interrogation de BD permettant à l'utilisateur de communiquer avec la machine en langage naturel (ON-LINE ENGLISH de Cullinane, RENDEZ-VOUS, DIALOGUE II de Bull, etc...). La formule peut paraître séduisante, mais il faut se rendre compte que le langage naturel peut être très ambigu. Et la machine est incapable de résoudre les ambiguïtés ! De plus, certains problèmes de dépassements sémantiques peuvent survenir : l'utilisateur pose une question qui lui paraît très logique, mais la machine ne peut pas y répondre.

C'est pourquoi il est souvent préférable de proposer à l'utilisateur final des logiciels dont la sémantique de dialogue est plus artificielle. Ceci ne veut pas dire qu'il faille lui demander d'apprendre des langages de types COBOL ou PL1; ils sont trop procéduraux, en ce sens qu'il faut donner des instructions détaillées et précises pour chaque action à accomplir. Par contre, un langage dit "non-procédural" sera en général mieux adapté, car il s'agit d'un langage où l'on doit spécifier ce qui doit être fait, mais pas comment il faut le faire.

Ainsi, dans un langage non procédural, l'aspect algorithmique revêt une importance moins grande qu'avec les langages de programmation classiques. Par exemple, la notion de boucles et celle de tests ne doivent généralement pas être traduites de manière explicite. C'est là un avantage capital quand on sait qu'un utilisateur final n'aura certainement pas reçu une formation méthodologique très avancée : dans le meilleur des cas, il pourra exprimer clairement ce qu'il désire, mais sera tout à fait incapable de le traduire sous forme d'algorithme. D'ailleurs il n'en a ni le temps, ni l'ambition : ce n'est pas son travail.

Un langage d'accès tel que SQL correspond assez bien, selon nous, à la notion de non-procéduralité que nous venons d'exposer.

2.4.3. Modularité d'apprentissage

Lorsque les utilisateurs se trouvent pour la première fois devant un terminal où ils peuvent obtenir ou manipuler des données, l'usage doit en être simple. Dans la section précédente, nous avons brièvement évoqué la raison pour laquelle un langage de manipulation de données ne pouvait pas être - ou pouvait l'être mais avec de nombreuses restrictions - le langage courant (français, anglais).

Il s'agit dès lors de proposer un langage présentant au moins deux caractéristiques essentielles.

Premièrement, le langage doit être facile à utiliser par un néophyte de l'informatique. La syntaxe et le dialogue doivent être de style courant, faciles à retenir, en deux mots extrèmement conviviaux. En effet, la plupart des utilisateurs ne réfléchissent pas comme des programmeurs et ne seront pas enclins à mémoriser de la mnémonique et des formats très ésotériques. Ceci est d'autant plus vrai si l'utilisateur n'utilise qu'occasionnellement son terminal.

Ensuite, le langage proposé se doit d'avoir une conception modulaire : dans un premier temps, l'utilisateur pourra n'apprendre qu'une petite partie de la syntaxe et déjà pouvoir effectuer des requêtes simples. Puis, progressivement, à mesure qu'il se sentira plus à l'aise, il effectuera des requêtes plus complexes, ira trouver dans la brochure le moyen de satisfaire une requête un peu particulière...

L'aspect pédagogique revêt ici une importance non négligeable : il serait par exemple tout à fait regrettable de confier à un utilisateur novice une brochure de six cents pages en anglais, alors qu'il est déjà surchargé de travail, et qu'il n'a plus pratiqué la langue de Shakespeare depuis dix ans ! Ceci reviendrait à tuer dans l'oeuf toute velléité d'apprentissage par l'utilisateur d'un langage de haut niveau.

2.4.4. Rapidité de la formation de base

Il s'agira d'abord de garantir à l'utilisateur une formation de courte durée. Un jour, deux maximum, devraient normalement suffire pour convaincre tout

utilisateur potentiel de l'intérêt qu'il aurait à s'impliquer pleinement dans le projet proposé.

La formation se fera généralement à l'intérieur de l'entreprise, pour des raisons évidentes de coûts. Dans un premier temps, elle s'attachera à démontrer toutes les possibilités du logiciel nouvellement acquis. Ensuite, elle aura pour tâche d'enthousiasmer l'utilisateur, en lui offrant les moyens d'élaborer lui-même ses requêtes élémentaires.

Un deuxième volet de la formation, essentiellement méthodologique, aura pour objectif de former l'utilisateur à l'analyse d'un problème. En effet, un outil puissant, mis entre des mains inexpérimentées, peut provoquer une forte dégradation des performances du système.

Le but d'un formation méthodologique est donc de responsabiliser davantage les utilisateurs et de leur faire prendre conscience du risque qui pourrait résulter d'une utilisation inconsidérée des possibilités du nouvel outil.

De toute façon, une formation de base n'est pas suffisante. Il faut en plus concevoir un système d'encadrement constant et personnalisé, assuré par des gens compétents et disponibles.

2.4.5. L'effet de levier.

Si, dans toute organisation, il existe toujours des "réticents" à tout changement, il existe aussi leur contraire : les "enthousiastes". Dans le cas qui nous concerne, ces derniers pourront jouer, volontairement ou non, un rôle prépondérant dans la diffusion du nouveau produit logiciel.

Généralement, la période de formation prévue est insuffisante pour l'exploitation maximale de l'outil qui leur est confié. C'est ainsi qu'ils n'hésiteront pas à "éplucher" une brochure apparemment rébarbative pour y déceler les subtilités du produit que le cours de base n'aurait pas mises en évidence; en principe, ils atteindront assez rapidement des résultats remarquables. Le problème ne sera plus alors de leur apprendre une syntaxe, mais plutôt de canaliser leur enthousiasme.

Dès l'instant où les "enthousiastes" atteindront des résultats valables, les moins téméraires commenceront à

mieux comprendre l'intérêt de leur nouvel outil; ils se renseigneront davantage, demanderont conseils et aide à leurs confrères plus "érudits" qu'eux. Si le climat est bon au sein de l'organisation, l'expérience des "pionniers" évitera certains écueils aux nouveaux adeptes. Peu à peu, l'idée d'échanges mutuels entre les différents acteurs verra le jour, et une collaboration efficace s'averera positive ...

Ce principe peut être qualifié d'effet de levier, par analogie avec la terminologie économique.

2.4.6. Dialogue avec l'informaticien.

Le fait que l'utilisateur ait accès aux données communes à toute l'organisation ne sera pas sans effet sur ses relations avec l'informaticien. En effet, désormais ils pourront avoir une même vue sur la base de données : tous deux accèdent aux données de la même façon. Le dialogue ne pourra que s'en trouver amélioré.

De plus, pour peu que le SGBD soit uniformément relationnel, c'est-à-dire qu'il dispose d'un langage d'interrogation utilisable à la fois en mode interactif et intégré à un programme rédigé en un langage hôte, informaticiens et utilisateurs auront davantage encore l'impression de "parler" un même langage.

CHAPITRE III

LES PROBLEMES ET LES CONTRAINTES

Après avoir abordé les aspects attrayants des langages de 4ème génération dans le chapitre précédent, celui-ci a pour objectif d'analyser les problèmes et les contraintes éventuellement rencontrés par une organisation face à la solution proposée par les L4G. Cette partie met le lecteur en garde face à l'opinion trop optimiste d'un représentant enthousiaste à l'égard des langages de 4ème génération; elle permet aussi d'examiner les brochures publicitaires d'un oeil plus critique.

Ce chapitre se subdivise en 4 sections :

- l'aspect humain;

- l'aspect technique;

- les applications développées par un L4G;

- le choix d'un L4G.

Section 3.1 : L'aspect humain

Réticences psychologiques

L'introduction de l'informatique dans une organisation a quelque peu bouleversé le travail de certaines personnes. L'automatisation de certaines tâches nécessite un changement au sein de l'organisation. Face à ces changements, certains dangers ou difficultés apparaissent.

Les acteurs d'une organisation auront, sans doute, tendance à se cramponner à des avantages acquis. La réaction de certaines personnes face aux changements indispensables à l'établissement d'un équilibre perçu comme incertain est de protéger leur situation actuelle.

L'introduction d'un processus d'automatisation peut être perçu comme un transfert de savoir de l'homme vers la machine. Certaines personnes éprouvent donc le sentiment d'être dessaisies de leur savoir. Dès lors, les acteurs d'une organisation seront peut-être très passifs à l'égard des changements.

Bien souvent l'ordinateur est vu comme un outil puissant, surdoué, ... Les capacités de l'ordinateur sont donc quelque peu surévaluées. Le danger d'une telle vision réside dans une absence d'esprit critique face aux résultats d'un traitement informatique, ce qui a comme conséquence de transformer en certitude ce qui n'était qu'hypothèse.

Après cette énumération non exhaustive des problèmes humains liés à l'introduction de l'informatique, on peut conclure par le dilemme suivant :

- d'une part, la libération des tâches routinières peut accroître la satisfaction humaine et augmenter la créativité;

- d'autre part, la rigidité de certaines tâches automatisées et l'accroissement du rythme de travail peuvent déboucher sur un appauvrissement des tâches.

En ce qui concerne les langages de 4ème génération, on peut rencontrer, éventuellement, les problèmes évoqués ci-dessus lors de l'introduction de ceux-ci dans une entreprise. En effet, le L4G est "un pas supplémentaire" dans l'automatisation de certaines tâches d'une entreprise. De plus, dans certains cas, le L4G a pour

effet d'introduire, pour la première fois, une machine chez l'utilisateur.

La formation

Le chapitre 2 "L'approche 4ème génération" insiste sur la nécessité d'une formation méthodologique et technique. Le but de cette section est d'établir les problèmes liés à cette formation.

D'une manière générale, la formation repose sur une série d'exposés. La première difficulté consiste à bien déterminer le contenu de chaque exposé afin que le "rendement pédagogique" soit optimal.

Au niveau formation, on distingue les L4G orientés "utilisateurs" et les L4G orientés "développeurs".

Les L4G orientés "utilisateurs" nécessitent bien évidemment une formation de l'utilisateur final qui est généralement peu disponible. En effet, les utilisateurs n'ont pas toujours le temps de suivre une formation. Plus les personnes occupent une place élevée dans la hiérarchie de l'entreprise, moins celles-ci ont la possibilité de consacrer deux ou trois jours entiers à une formation. Il s'agit de leur faire comprendre le rendement de cette formation. En effet, deux jours entiers de cours intensifs sur un outil qui permet un gain de 2 heures chaque jour s'avèrent déjà rentable à court terme.

Dans le cadre des L4G destinés à l'utilisateur, l'hétérogénéité des formations initiales peut constituer un handicap. En effet, ingénieur, secrétaire, comptable possèdent des formations très différentes et leur base en logique aura sans doute un impact sur leur capacité d'apprentissage.

Ajoutons enfin la nécessité pour un utilisateur final de pouvoir recourir à du personnel compétent en cas de problème.

En ce qui concerne les langages de 4ème génération orientés "développeurs", la nature des problèmes de formation est sensiblement différente des L4G orientés "utilisateurs". On note que la logique des langages de 4ème génération n'est pas toujours comparable à celle du langage "Cobol". Construire un programme à l'aide d'un L4G avec la logique Cobol ne constitue pas nécessairement une utilisation optimale de l'outil de 4ème génération. Mais ce problème est spécifique à chauque L4G.

Section 3.2 L'aspect technique

L'intégration dans un environnement existant

Dans la plupart des cas, l'informaticien doit tenir compte d'une série de facteurs hérités du passé. Ceux-ci apparaissent comme un fardeau, un poids, ... face aux techniques actuelles.

En ce qui concerne la reprise des données, le L4G est conçu, bien souvent, autour d'un schéma de données relationnel. La transformation d'un schéma de données conforme à Codasyl en un schéma de données exploitable par le L4G n'est pas chose aisée et peut poser des problèmes. Les L4G proposent des outils différents :

1 - exploitation en ligne par un autre SGBD. Le système fait une requête à un autre système pour obtenir les données qui répondent à cette demande ponctuelle;

2 - transfert de schémas. Seul le schéma est transféré. Il y a donc une description de l'ancien schéma dans le nouveau SGBD;

3 - transfert total des données et des schémas c'est-à-dire que toutes les données et leurs schémas sont recréés dans le nouveau système.

A partir de cette découpe, deux problèmes apparaissent. En effet, dans la première solution, l'interface ainsi instaurée entre les deux SGBD ne permet généralement pas les opérations de mises à jour générées par le système "étranger". On remarque, dans le troisième outil, la charge de conserver l'ancien schéma afin de conserver les anciennes applications. Il faut, dès lors, gérer deux SGBD différents, ce que certains systèmes d'exploitation ne permettent pas. Cette dernière remarque évoque le problème de la reprise des programmes existants qui mérite quelques commentaires.

Les applications qui sont développées autour d'une base de données hiérarchique ou réseau ou autour des fichiers classiques sont difficilement transportables dans un environnement relationnel. A l'heure actuelle, il existe des ateliers d'aide à la conception d'une base de données qui transforment un schéma conceptuel en un schéma conforme aux propriétés d'un système de gestion de données (Codasyl, Cobol, relationnel). Mais il n'existe pas, à notre connaissance, d'outils capables de transformer des

programmes écrits en Cobol en programmes exploitables dans un environnement relationnel.

Implication au niveau des ressources

Par rapport aux langages classiques, les L4G consomment plus de ressources. Il serait intéressant d'effectuer une comparaison chiffrée sur l'impact au niveau des ressources machine entre les langages de haut niveau (Cobol, PL1,...) et les langages de 4ème génération. Les unités de mesure indispensables dans cette comparaison sont les suivantes :

- le temps CPU;

- la mémoire centrale utilisée;

- le temps d'utilisation des canaux pour les I/O;

- le temps d'attente;

- le temps total de présence en machine.

Cette liste est loin d'être exhaustive. Pour mener à bien une étude comparative, les unités de mesure doivent être indépendantes de la configuration, indépendantes du système d'information ainsi que facilement calculables et contrôlables. Ces conditions respectées, il est plus aisé de choisir un nouvel équipement (matériel/logiciel).

Un langage puissant pour l'utilisateur (par exemple : Query-by-example) permet aux utilisateurs[1] de se lancer dans des opérations consommatrices de temps machine. En effet, l'utilisateur peut réaliser une opération de liaison relationnelle portant par exemple sur deux tables de 10000 enregistrements. Dès lors, si plusieurs utilisateurs effectuent des opérations similaires, le temps de réponse des autres sera plus long - ce qui constitue un problème.

---

[1] lorsque l'utilisateur lance une opération, il n'a souvent aucune indication des frais occasionnés par celle-ci. Peu de SGBD annoncent le coût avant que l'utilisateur ne déclenche l'opération.

Performance

Le concept de performance est très lié à la notion des ressources évoquée ci-dessus. En effet, une dégradation des performances entraîne souvent la nécessité d'augmenter les ressources.

Certains programmes écrits dans un langage de 4ème génération offrent des performances moins "bonnes" que les programmes écrits dans un langage classique. Certains L4G n'ont pas été conçus dans le cadre d'une optimisation gobale du code produit. Dès lors, une application développée dans un langage de 4ème génération peut avoir un temps d'exécution non acceptable. Pour résoudre ce problème, un spécialiste de ce L4G devra optimiser toute l'application ... si c'est possible.

Les problèmes de performance peuvent aussi survenir lorsque le L4G est interprété plutôt que compilé.

Garder un langage classique

La faiblesse des structures algorithmiques de certains langages de 4ème génération pose des problèmes. En effet, lorsque l'application atteint un degré de complexité assez élevé, il est indispensable de disposer des structures conditionnelles, appels de procédure, boucles, ... chers aux langages classiques. Dès lors, un langage de 4ème génération doit permettre, si possible, un interface avec les langages comme le Cobol, Fortran, C afin de résoudre les problèmes qui apparaissent, en L4G, lorsqu'on atteint un seuil critique ...

Sécurité et recouvrement d'erreur

L'introduction d'un langage de 4ème génération implique généralement une augmentation du nombre d'utilisateurs de la base de données. Les problèmes de droits d'accès des utilisateurs et de reprises des données en cas d'incidents seront donc plus fréquents. Ces problèmes sont traités en profondeur dans les ouvrages de DATE : "An Introduction To Data Base Systems" tomes 1 et 2.

Gestion des utilisateurs

Les unités de mesure d'un système (cfr. supra) constituent les éléments de base pour déterminer la consommation des utilisateurs. Sans vouloir aboutir obligatoirement à une facturation effective, il est intéressant d'informer l'utilisateur du coût des

prestations qu'il demande au service informatique. Dans le cas où l'utilisateur développe lui-même ses applications, il est aussi avantageux de lui renseigner le coût de leur exécution sur la machine. Dans le cadre d'une comptabilité budgétaire, l'imputation des coûts permet d'effectuer le suivi budgétaire des services informatiques.

Section 3.3 Les applications développées à l'aide d'un L4G

Les premières étapes de la conception d'un système d'information consistent en une modélisation d'un existant. Une solution automatisée essaye le plus possible de se rapprocher du réel perçu et essaye, autant que faire se peut, d'anticiper les changements qu'elle occasionne. En effet, il est évident que créer une application informatique pour remplacer un processus manuel modifie ce processus.

Dans le cadre des langages de 4ème génération, la méthode de conception qui se base sur la souplesse du langage pour remanier l'architecture d'une application au fur et à mesure de la découverte des besoins n'est pas souhaitable. La facilité de réaliser certaines applications présente un réel danger : la souplesse des L4G ne permet pas de se passer d'une rigueur dans le raisonnement lors de la conception d'une application.

L'introduction d'un L4G permet à l'utilisateur, dans certains cas, de développer lui-même ses applications. Or, l'utilisateur ne dispose pas souvent d'une formation méthodologique approfondie. Les débuts - par la création d'applications courtes et simples - sont très encourageants. Mais, lorsque le programme s'allonge, la maintenance pose de gros problèmes. Etant donné que les utilisateurs finals ont des formations très hétérogènes et qu'ils n'ont pas - ou très peu - de formation méthodologique, ils conçoivent leurs applications de façons très différentes. De sérieux problèmes de maintenance surviennent lorsque l'utilisateur quitte son service et laisse ses applications entre les mains de son remplaçant.

En conclusion, afin d'éviter ces problèmes, il reste indispensable d'avoir une architecture modulaire correcte. De plus, il n'est pas raisonnable de penser que la conception et la réalisation d'applications peuvent être faites avec du personnel moins rompu aux techniques informatiques classiques ...

Section 3.4 : Le choix d'un L4G

L'aspect financier : la méthode de la valeur présente nette

"Les décisions d'investissement revêtent pour les entreprises une importance tout à fait capitale car elles orientent leur activité d'une manière difficilement réversible et cela pour une période assez longue ...".

D'une manière générale, avant tout investissement, il est nécessaire de calculer sa rentabilité c'est-à-dire, en d'autres termes, d'examiner si les recettes qu'il génère sont supérieures aux dépenses qu'il entraîne. Il existe différents éléments intervenant dans le calcul de la rentabilité : la mise investie, les flux nets annuels relatifs à l'investissement, la fiscalité et le coût du capital.

Dans le cadre plus restreint de l'acquisition d'un nouveau logiciel, il s'agit de déterminer si "cela en vaut la peine". L'acquisition d'un L4G implique un accroissement de la productivité qui, en termes de coût, débouche sur des gains au niveau des salaires, sans parler de l'aspect qualitatif du traitement de l'information, qui peut aussi être à la base de flux financiers positifs.

La mise investie comporte les investissements matériel/logiciel, le coût d'aménagement des locaux, les études, les coûts de formation,... Les flux nets annuels relatifs à l'investissement se calculent en effectuant la différence annuelle entre l'accroissement des recettes et l'accroissement des coûts engendrés par l'adoption d'un projet d'investissement. En ce qui concerne l'adoption d'un nouveau logiciel, les flux nets peuvent être constitués par des sommes d'argent qui ne seraient pas sorties de l'entreprise. Dès lors, les investissements qui consistent à acquérir un langage de 4ème génération sont des investissements de rationalisation dans lesquels les dimminutions de coûts font office de flux nets. Parmi les dépenses annuelles qui résultent de l'acquisition d'un L4G, on peut citer : les contrats de maintenance, le coût des matières de fonctionnement, les dépenses en personnel (maintenance, exploitation, administration).

Avec tous les paramètres décrits ci-dessus, il est possible de comparer, dans une certaine mesure, l'impact de l'apport d'un L4G au point de vue financier. Par la méthode de la "valeur présente nette", on peut estimer, tout d'abord, les flux nets (FNt) relatifs aux différentes années t qui vont s'écouler depuis la mise en route de

l'investissement, jusqu'à la fin de sa durée de vie, à l'année T.

FNt = RECt - DEPt

FNt = - DEPt (dans le cadre d'un investissement de rationalistaion; exemple : l'acquisition d'un L4G)

RECt : recettes à l'année t
DEPt : dépenses à l'année t

On calcule ensuite la somme des valeurs actualisées, au coût du capital i, de chacun de ces FNt.

$$VO = \sum_{i=1}^{T} \frac{FNt}{(1+i)^t}$$

Le calcul de la "valeur présente nette" (VPN) s'obtient comme suit :

VPN = VO - I

où I est la mise investie

Etant donné que FNt est négatif, la valeur présente nette (VPN) est donc négative. Dès lors, le choix se porte sur la solution pour laquelle la VPN est la moins négative. Pour déterminer le coût du capital, on peut prendre comme référence le taux d'intérêt octroyé à la société pour son dernier investissement.

La première critique que l'on peut faire à propos de cette méthode est la difficulté de déterminer T.[2] En effet, il est rarement possible d'estimer la durée pendant laquelle un service informatique utilisera un logiciel. On remarque que cette difficulté disparaît lorsque l'acquisition d'un logiciel s'effectue dans le cadre d'un seul projet informatique où une contrainte de temps est imposée. A partir de ces deux extrêmes, on peut trouver, éventuellement, un compromis qui consiste à effectuer une

---

[2] Pour les projets informatiques, l'horizon doit être limité à cinq ans - ce qui permet de tenir compte assez rapidement des nouvelles technologies.

comparaison entre la solution existante et l'acquisition d'un langage de 4ème génération sur base d'un nombre limité de projets informatiques de taille moyenne. A titre exemplatif, si l'acquisition d'un langage de 4ème génération est déjà rentable pour la réalisation de 5 projets informatiques répartis sur 3 ans, il est évident que l'entreprise optera, sans hésitation, pour l'acquisition d'un L4G si le cahier des charges du service informatique dépasse largement 5 projets sur la période considérée. On peut donc imaginer une méthode qui consiste à déterminer un seuil exprimé en nombre de projets informatiques pour une période déterminée pour lequel l'acquisition d'un L4G est déjà "rentable".

La deuxième critique est le fait que cette méthode ne prend pas en considération les aspects qualitatifs. Il est illusoire, par exemple, de quantifier en unités monétaires la satisfaction d'une personne qui développe en L4G plutôt que dans un autre langage (Cobol, PL1, Pascal, ...).

Le fait de pouvoir déterminer le gain (en argent) potentiel d'une solution préconisant l'acquisition d'un L4G peut conduire à limiter les engagements dans le service informatique et, dans une certaine mesure, à licencier du personnel. Cette dernière stratégie n'est pas toujours applicable ou elle ne correspond pas toujours à la politique social de l'entreprise ...

Ajoutons enfin, en ce qui concerne cette méthode de la valeur présent nette, que si celle-ci est satidfaisante d'un point de vue théorique, elle n'en est pas moins rarement appliquée en pratique (pour les investissements logiciels) ! Si nous l'avons évoquée, c'est pour souligner le fait que, dans une entreprise, l'acquisition d'un L4G est perçue par la direction comme un investissement - au même titre qu'une machine - qui doit, à terme, se révéler rentable : on n'investit pas dans un projet pour la beauté du geste !...

Méthode d'évaluation sur base d'un score

Cette méthode (appelée "scoring method" dans la littérature anglo-saxonnes) est habituellement utilisée pour sélectionner une configuration matérielle/logicielle; mais elle peut être ajustée pour choisir un outil de 4ème génération. Il s'agit d'établir, tout d'abord, la liste des caractéristiques significatives pour la sélection d'un L4G. On pose ensuite Sij, le score obtenu pour un L4G j pour la caractéristique i. De plus , on renseigne le poids Wi accordé à chaque caractéristique. A partir de toutes ces données, on établit :

$$Pj = W1*S1j + W2*S2J + \ldots\ldots + Wn*Snj$$

$$\sum_{i=1}^{n} Wi = 1 \quad et \quad Wi > 0$$

Pj représente le score obtenu d'un L4Gj. Cette méthode a l'avantage d'être simple mais n'explique pas les relations de cause à effet. De plus, elle présente des défauts pratiquement rédhibitoires : comment établir une liste valable de caractéristiques ?; comment donner un poids à chacune de ces dernières ?; et surtout, comment attribuer des scores sans tomber dans les travers de la subjectivité ? En fait, cette méthode nécessite beaucoup trop d'appréciations subjectives pour être appliquée avec succès...

Bref, en matière de logiciels, il est extrèment difficile, concrètement, de trouver des outils fiables d'évaluation financière. Ceux qui existent ont pour la plupart le mérite essentiel de sensibiliser les décideurs aux différents facteurs qu'il y a lieu de prendre en compte avant d'acquérir un logiciel en général, et un langage de quatrième génération en particulier...

CONCLUSION

Jusqu'il y a peu, l'informatique prenait en charge un certain nombre de tâches répétitives : imprimer les factures; gérer la production, les stocks; produire le bilan des sociétés, etc... Mais elle se doit de remplir un tout autre rôle également : valoriser la capacité d'intelligence et de réflexion de chaque individu dans son poste de travail. J.P. de Jamblinne parle à ce sujet d'"esprit d'oeuvre", par opposition à la main d'oeuvre.

La maîtrise de cette évolution rendra l'industrie en général, et les sociétés de services en particulier, capables de produire de nouvelles générations de biens et services inconcevables dans le contexte antérieur, d'accroître considérablement leurs performances, et de maîtriser leurs coûts.

C'est dans ce contexte que s'inscrit l'émergence des langages de quatrième génération. En effet, ces derniers devraient apporter une contribution non négligeable à la mise en place d'organisations où le personnel est en permanence motivé, ouvert à la communication; où il est responsable et prend des décisions, quel que soit son niveau.

Mais cet idéal est loin d'être accompli. Les L4G, puisque c'est d'eux qu'il s'agit dans le cadre de cette étude, permettent certes, en théorie, des gains de productivité intellectuelle importants. En pratique, cependant, les obstacles sont nombreux : ils vont de l'inertie viscérale de toute organisation face au changement, au nécessaire effort de formation à accompir, en passant par une multitude de problèmes techniques spécifiques à chaque organisation...

Le tableau peut paraître bien sombre. Loin de nous, cependant, l'idée de vouer aux gémonies les L4G : à côté des problèmes qu'ils peuvent engendrer, il faut souligner les avantages qu'ils présentent. Un langage de quatrième génération, dans la majorité des cas, accroît de manière parfois impressionnante la productivité des professionnels de l'informatique, tant au niveau du développement d'applications qu'à celui de leur maintenance. Quant aux cadres, ils peuvent trouver dans un L4G le moyen de rendre plus sûre et efficace leur gestion. Dans cette optique, un L4G est un investissement, qui doit se montrer rentable d'un point de vue financier. Si c'est le cas, il ne fait aucun doute que les langages de quatrième génération, en dépit des réticences qui peuvent se manifester à leur égard, marqueront d'une pierre blanche l'évolution des

organisations vers une plus grande aisance à prendre en charge le traitement de l'information.

PARTIE 2 : ENQUETE

INTRODUCTION

La deuxième partie de cette étude sur les langages de quatrième génération est une enquête, réalisée sur base d'un échantillon de neuf entreprises utilisant un L4G. Dans chacune de ces entreprises, nous avons interviewé un informaticien professionnel, généralement le responsabe du département informatique, et lui avons soumis un questionnaire, reproduit intégralement et sous sa forme originale en annexe A.

La technique utilisée lors de nos entrevues était la suivante : un exemplaire du questionnaire était remis à notre interlocuteur; celui-ci en prenait connaissance et y répondait oralement, question par question; nous prenions note de ses réponses.

Ce questionnaire n'a pas été prétesté. L'intérêt d'une telle technique n'était pas capital dans notre cas, puisque lors d'une mauvaise interprétation d'une question par notre interlocuteur, il nous était facile d'intervenir et d'expliquer en d'autres termes l'objet de cette question; le malentendu était ainsi rapidement dissipé. C'est là le gros avantage d'une entrevue directe.

La plupart des questions posées étant ouvertes, la durée des entrevues fut généralement plus longue que celle prévue initialement (entre une heure quinze et deux heures trente, alors que nous prévoyions une moyenne de trois quarts d'heure). Ceci nous a conduit à parfois déborder quelque peu le cadre du questionnaire, ce qui ne s'est pas avéré sans intérêt pour une meilleure compréhension du fonctionnement du système d'information de l'entreprise.

Toutes les interviews ont eu lieu aux mois de février et mars 1987; aucune technique de stratification n'a été mise en oeuvre, eu égard à la faiblesse de notre échantillon.

Nous tenons à remercier ici encore toutes les personnes ayant répondu à notre questionnaire, qui, malgré un emploi du temps généralement très chargé, se sont toujours montrées très disponibles et ouvertes.

Les objectifs généraux de cette enquête, et les points particuliers qu'il nous paraissait intéressant de souligner, sont définis dans le premier chapitre de cette partie.

Le deuxième chapitre présente une synthèse des résultats de l'enquête. Il y est parfois fait référence,

à titre de comparaison, à une étude relativement semblable à la nôtre, réalisée aux USA à la demande d'Applied Data Research, par l'organisme de sondage INPUT [INPUT 85]. Cette étude présente l'avantage, par rapport à celle que nous avons réalisée, de porter sur un échantillon plus large : quatre-vingts utilisateurs y ont participé. Nous nous attacherons à établir un parallèle entre les deux enquêtes, lorsque l'opportunité se présentera.

Remarquons enfin que les SGBD et logiciels dont il est question dans cette enquête ne sont pas nécessairement représentatifs de ce qui existe en Belgique dans ce domaine...

CHAPITRE PREMIER

L'INTERET D'UNE ENQUETE

Section 1.1 : Objectif général

Le but essentiel de l'étude qui suit est d'analyser comment, concrètement, sont perçus et utilisés les langages de quatrième génération. Notre objectif, à travers cette étude, est donc de découvrir et de localiser ce qui différencie la théorie de la pratique, et d'en mesurer l'ampleur.

Notre ambition n'est pas de réaliser un sondage où nous alignerions des pourcentages dans des tableaux, mais plutôt d'essayer de dégager certaines tendances, certains aspects des L4G qui, sur le terrain, se sont révélés positifs ou, au contraire, non conformes à ce qu'on en attendait... C'est pourquoi il nous est apparu digne d'intérêt de rencontrer, sur leur lieu de travail, les utilisateurs de L4G.

Dans cette optique, nous avons rédigé un questionnaire découpé en un certain nombre de fonctions, afin de systématiser, dans la mesure du possible, l'élaboration de notre démarche. Nous abordons, dans chacune de ces fonctions, un point qui nous paraît être essentiel quant aux possibiités théoriques d'un langage de quatrième génération de "se distinguer" par rapport à un langage de troisième génération comme COBOL.

Section 1.2 : Outils

L'étude des langages de quatrième génération implantés dans une entreprise demande en premier lieu une description de ceux-ci. Cette description contiendra l'identification du L4G ainsi que celle de l'environnement dans lequel il est utilisé.

En outre, il nous a paru digne d'intérêt de découvrir les raisons qui ont poussé l'entreprise - ou plutôt ses dirigeants - à acquérir un L4G en général, et le L4G qu'elle utilise en particulier.

Section 1.3 : Possibilités techniques exploitées

Il est intéressant d'établir la liste des fonctions offertes par le L4G, et de déterminer celles effectivement utilisées, d'une part par le développeur, et d'autre part par l'utilisateur.

Parmi les composants du L4G, quels sont ceux offerts:

- aux développeurs d'applications;

- aux utilsateurs, dans le cadre de systèmes d'aide à la décision ?

Le L4G présente-t-il, en outre, des particularités intéressantes ou originales ?

Section 1.4 : Organisation

Le thème abordé ici a pour but de décrire de manière statique l'organisation du personnel qui gravite autour du L4G. En d'autres termes, nous cherchons à cerner le rôle de chacun des acteurs du sytème d'information utilisant le L4G.

Quelle relation existe-t-il entre l'informatique de l'utilisateur et l'informatique opérationnelle ?

Ces trois thèmes (sections 1 à 3), spécifiques à une entreprise, permettent de définir la "carte de visite" du L4G. Nous allons aborder maintenant l'aspect utilisation

proprement dite du L4G, et les éventuels probèmes qui y sont liés.

Section 1.5 : Problèmes d'intégration

Deux types de problèmes peuvent se présenter à ce niveau :

- les problèmes d'intégration vus sous l'angle technique : ce domaine comporte entre autres les problèmes de transportabilité des applications; de compatibilité avec les méthodes d'accès, avec les machines existantes; de reprise de données et de programmes existants;...

- les problèmes d'intégration axés sur l'aspect humain, c'est-à-dire la formation des utilisateurs et des informaticiens, leur degré de réceptivité face au L4G,...

L'étude s'attachera à déceler l'existence éventuelle de problèmes et, le cas échéant, la façon dont ils ont été résolus.

Section 1.6 : Aide à la décision

Les cadres gestionnaires sont appelés à prendre des décisions stratégiques et financières qui conditionnent l'avenir de leur entreprise. Or, en gestion, un problème comporte souvent plusieurs solutions dont il faut apprécier les conséquences immédiates et futures. Certains L4G offrent des systèmes d'aide à la décision. Notre objectif est de mesurer l'impact que ces S.A.D. ont sur la qualité de l'information récoltée par les gestionnaires avant leurs prises de décision.

Section 1.7 : Développement d'applications

Le but est de déterminer les types d'applications développées à l'aide du L4G. Quel est le domaine d'application du L4G ? Quelle est l'envergure des applications développées ? Les applications légères, urgentes, personnalisées et non planifiées sont-elles prises en compte de manière satisfaisante ?

En outre, il serait intéressant de découvrir si de nouvelles méthodes de développement d'applications ont vu le jour suite à l'intégration d'un L4G au sein de l'entreprise.

Le rôle des utilisateurs a-t-il été modifié ? Quel est-il durant chacune des phases du cycle de vie d'une application informatique ?

Section 1.8 : Maintenance

On utilise le terme de maintenance pour désigner la mise à jour d'anciens programmes; le but est d'adapter ces derniers à de nouvelles exigences de l'utilisateur, ou à de nouveaux terminaux, systèmes d'exploitation, ordinateur ou structures de données.

La maintenance est un gros problème de l'informatique traditionnelle. Les L4G apportent-ils une solution valable à ce problème ? Qui assure la maintenance ? Quels sont les moyens qui permettent de la réaliser avec un maximum d'efficacité ?...

Section 1.9 : Productivité

Développement d'applications

L'intégration d'un L4G doit en théorie permettre une meilleure productivité en matière de développement d'applications. Le but de cette section est de vérifier si le L4G offre effectivment une meilleure productivité par rapport aux langages de troisième génération tels que COBOL.

L'introduction d'un L4G a-t-elle permis de satisfaire un plus grand nombre de besoins endéans des délais déterminés ?

Aide à la décision

Les systèmes d'aide à la décision permettent normalement un accroissement de la productivité des utilisateurs en termes de réduction du coût et du délai de

utilisateurs en termes de réduction du coût et du délai de certaines études. Qu'en est-il dans la réalité ?

DEUXIEME CHAPITRE

PRESENTATION SYNTHETIQUE DES RESULTATS DE L'ENQUETE

Section 2.1 : Outils

Nous reprenons ci-après les réponses aux questions 1 à 8 de la première section.

Nom L4G 1.1	Fabricant 1.2	Date insta 1.3	Nbre Cadres 1.7	Taille inf 1.8
ORACLE SAS	Oracle cp. Inst. Cary	12/86 08/84	6	2
MANTIS	Cincom	10/82	21	21
ADABAS Natural	Software AG	06/86	450	45
SQL/DS	IBM	06/84	14	8
ORACLE	Oracle Corp.	06/86	97	55
RALLY	DEC	06/86	15	20
IC1	IBM	04/86	3500	25
FOCUS	IDI	11/84	2100	50
MANTIS	Cincom	07/83	600	20

Avec :

- 1.3 : "Date insta" = Date d'installation du L4G;

- 1.7 : "Nbre Cadres" = Nombre de cadres travaillant dans l'entreprise;

- 1.8 : "Taille inf" = Taille (en nombre de personnes) du service informatique.

Il est intéressant de remarquer que les entreprises détiennent leur langage de 4ème génération depuis relativement peu de temps. En effet, dans les entreprises

que nous avons visitées, le L4G est installé en moyenne depuis 22 mois (avec un écart-type très important). Ceci constitue une illustration de la relative jeunesse du "phénomène quatrième génération".

Nom L4G 1.1	Marque ordi. 1.4	Modèle 1.5	Syst. exploit 1.6
ORACLE SAS	B.A.S.F.	7/65	VM/CMS
MANTIS	IBM	4381/P13	VM/DOS VSE
ADABAS Natural	IBM	4381 modèle 12	VM/CMS DOS VSE
SQL/DS	IBM	4381/P12	VM/CMS
ORACLE	WANG	PC compatible IBM	MS-DOS 2.11, 3.1
RALLY	DIGITAL	VAX	VMS
IC1	IBM	4381 modèle 2	VM/SNA
FOCUS	IBM	3081	MVS/CSO
MANTIS	IBM	4381/P12	VM/VSE

Avec :

- 1.4 : "Marque ordi" = marque de l'ordinateur sur lequel est implanté le L4G;

- 1.6 : "Syst. exploit" = système d'exploitation.

Question 1.9

Les raisons qui ont poussé nos interlocuteurs - ou, de manière plus générale, la direction de leur entreprise - à acquérir leur L4G sont à classer en deux catégories.

Tout d'abord, il y a les motivations tout à fait générales, indépendantes du contexte et de l'environnement propres à l'entreprise.

Le maître-mot est ici la **productivité**. Sur les neuf entreprises que nous avons visitées, on retrouve à sept reprises cette volonté non dissimulée de trouver, à travers l'utilisation d'un langage de 4ème génération, le moyen de "développer plus rapidement les applications", de "diminuer le temps de développement", ou encore d'"accroître la vitesse de programmation et de mise au point". Toutes ces réflexions convergent vers un même objectif : accroître la productivité, afin de diminuer les coûts liés au développement d'applications, ou à tout le moins afin d'éviter leur augmentation.

Un L4G devrait permettre aussi, selon certains de nos interlocuteurs, d'établir une **meilleure adéquation** entre les services offerts par le département informatique et les besoins réels des utilisateurs : "permettre de faire de belles synthèses, adaptées aux besoins"; "améliorer la qualité du travail de l'informaticien"; "permettre une meilleure présentation";...

Un autre aspect concerne la **réduction de la maintenance**; c'est une des motivations d'acquisition d'un L4G de trois de nos interlocuteurs.

L'**approche base de données** constitue aussi un élément important des L4G. L'idée est que les traitements changent dans le temps; l'entrée dans l'environnement d'une base de données (relationnelle) doit permettre l'utilisation de "méthodes par les données". Le L4G est tout naturellement associé à cette approche.

Dans le cas des L4G orientés - notamment- utilisateurs finals (IC1, ORACLE, SQL/DS et FOCUS), on trouve la volonté de mettre à la disposition de l'utilisateur final un outil qui lui soit accessible. C'est ici qu'émerge généralement le concept d'**infocentre** (bien que ce terme ne soit pas toujours explicitement utilisé). Nous traitons ce thème plus en détail par ailleurs dans ce compte-rendu.

Enfin, certains voient dans leur environnement de quatrième génération un outil qui a un bel **avenir** devant lui.

Remarquons que la possibilité de réaliser aisément des **prototypes** n'a été citée qu'une seule fois en tant que motivation d'achat.

A titre de comparaison, et dans le but bien compréhensible de confirmer ou de nuancer les tendances que nous avons pu dégager de notre enquête basée sur un échantillon statistiquement peu représentatif, nous donnons ci-après les résultats d'une enquête réalisée aux USA, à laquelle ont participé quatre-vingts utilisateurs de ADR (Applied Data Research) [INPUT 85]. Il était demandé aux personnes interviewées de donner un poids, sur une échelle graduée de 0 à 5, à un certain nombre de facteurs susceptibles d'avoir joué un rôle dans leur décision d'acquisition d'un L4G en général. Les résultats sont présentés dans la figure II.1 et dans le tableau II.1.

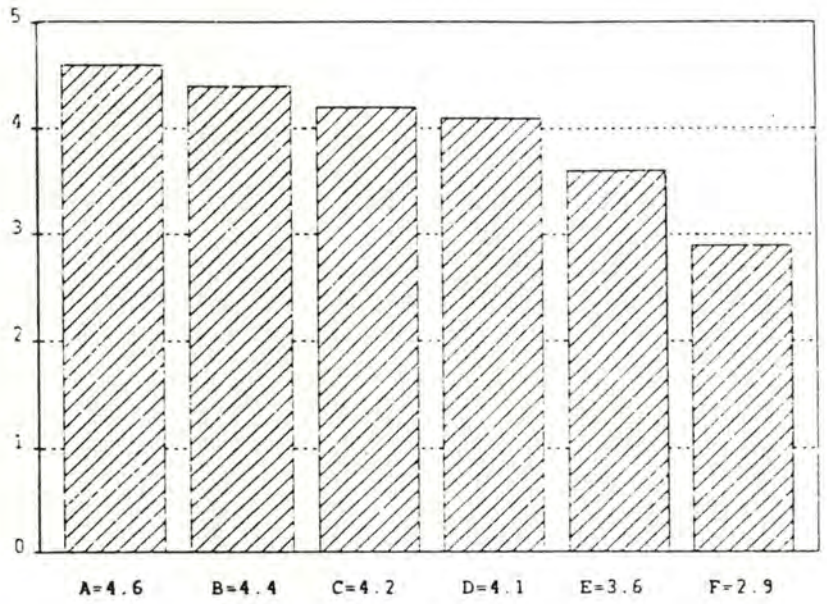


Fig. II.1 : **Facteurs de choix d'un L4G.**

A - Accroissement de productivité.

B - Réduction de la maintenance des applications.

C - Capacité à développer rapidement des applications.

D - Facilité d'utilisation.

E - Réduction du retard informatique.

F - Terminer un projet d'envergure dans des délais stricts.

Le tableau II.1 reprend les facteurs autres que ceux proposés dans la figure II.1, et cités spontanément par les personnes interviewées : un total de 56 réponses autres que celles proposées ont été données par 44 personnes sur 80. Seuls ont été retenus dans le tableau les facteurs qui intervenaient plus d'une fois.

	Proportion	Poids
BD relationnelle ADR déjà implantée	26%	4,8
Réduction à long terme des coûts et du temps de formation	19%	4,2
Interface avec un dictionnaire	16%	4,5
Concept de "vendeur unique"	13%	4,8
Possibilité de prototypage	8%	5,0
Complexité de CICS	8%	4,5
Standardisation basée sur un langage simple	5%	5,0
Réduction du personnel informatique	5%	4,0
	100% TOTAL	4,6 MOYENNE

Tab. II.1 : **Autres facteurs ayant motivé l'acquisition d'un L4G; seuls sont repris ceux ayant été cités au moins deux fois.** N = 44.

La comparaison de ces résultats avec les tendances que nous avons pu dégager de notre propre enquête démontre de manière éloquente que l'**aspect productivité** - qu'il soit exprimé sous la forme d'"accroissment de productivité" (A), ou sous celle de "capacité à développer rapidement des applications" - se révèle être l'élément essentiel qui pousse l'entreprise à se doter d'un langage de quatrième génération. Quant à la facilité ou à la **réduction de la maintenance**, elle constitue un facteur statistiquement aussi important.

Par ailleurs, nous avons souligné que, dans notre enquête, le facteur prototypage n'était mentionné qu'une seule fois; l'enquête américaine semble confirmer cette tendance puisque, en chiffres absolus, il n'est cité que par cinq personnes sur les quatre-vingts interviewées, mais avec un taux de pondération de 5 sur une échelle graduée de 0 à 5. Ceci montre que, si le prototypage n'intervient pas souvent comme motivation d'achat d'un L4G, quand il intervient, c'est un élément absolument prépondérant.

Remarquons enfin que l'enquête américaine ne fait nulle part mention de la notion d'infocentre. Ceci s'explique par le fait que cette étude concerne des langages essentiellement orientés développeurs, à savoir, ADR, MANTIS, FOCUS, NATURAL et ADS ON-LINE.

Nous venons d'évoquer les motivations générales qui ont poussé nos interlocuteurs - ou la direction de leur entreprise - à acquérir un L4G. Voyons maintenant quelles sont les raisons qui les ont motivés à choisir le système particulier qu'ils utilisent. C'est l'objet de la question 1.10.

L'idée générale est que les entreprises choisissent le produit qui leur **convient le mieux**, ou, suivant une vision des choses plus pessimiste, celui qui présente le moins d'inconvénients ! Généralement, une étude préliminaire permet de ne retenir comme candidats que deux ou trois produits concurrents.

Ainsi, l'objectif premier est de choisir un L4G qui **s'adapte bien à l'environnement** : il doit offrir une possibilité d'interfaçage avec le système existant.

Certaines entreprises bénéficient de l'**expérience acquise** dans le domaine par l'une ou l'autre de leurs filiales. Cela leur permet d'éviter une phase parfois longue de tests comparatifs.

Un élément non négligeable est la recherche d'un **gestionnaire de base de données relationnelle complet**, avec recouvrement d'erreurs, verrouillage des tables et enregistrements, dictionnaire de données intégré,...

Il faut aussi que le logiciel soit **portable**, de manière aussi transparente que possible.

De plus, il est nécessaire que le produit soit **facile à apprendre** et à utiliser par un utilisateur non spécialiste.

La nécessité de disposer d'un logiciel qui ne **pénalise pas trop les performances** ni le temps de réponse est citée à deux reprises.

Section 2.2 : Possibilités techniques exploitées

Les sociétés qui disposent d'un SGBD relationnel bénéficient généralement d'un **langage d'accès** associé à cette base de données. Si c'est le cas, ce langage est, dans certains cas, exploité de manière **intensive**, et par les développeurs, et par les utilisateurs finals. Ces derniers, cependant, ne recevront **jamais** la possibilité de **mettre à jour** les données; ils pourront en revanche les consulter assez librement. Et c'est ici qu'un de nos interlocuteurs justifie sa **réticence** à donner à l'utilisateur final la possibilité d'exploiter un tel langage : "ceci nécessite une bonne formation des utilisateurs, afin d'éviter la dégradation des performances par des questions idiotes. C'est la raison pour laquelle l'utilisation d'un langage d'accès n'est pas encore généralisée". Remarquons enfin que nombre de départements informatiques veillent à développer eux-mêmes des requêtes qui se répètent souvent. **Par souci de performances**...

En général, tous les L4G disposent d'un **générateur de rapports intégré**. Quand celui-ci existe, il est exploité au maximum; il se révèle être un outil très précieux pour les développeurs. "On ne vit plus sans" nous a-t-on déclaré.

Les **générateurs d'applications**, quand ils existent, sont largement sous-utilisés : les gens préfèrent en général programmer toute l'application eux-mêmes, s'estimant déjà très heureux de disposer d'un outil de développement efficace. Cependant, dans le cas d'Oracle, le générateur d'applications est toujours utilisé (dialogue interactif).

La **documentation** coûte cher; il faut beaucoup de temps pour la rédiger, et beaucoup d'entreprises que nous avons visitées se limitent au minimum nécessaire. Les descriptions fournies par le dictionnaire des données et quelques lignes de commentaires dans les programmes suffisent souvent à satisfaire la "bonne conscience" de nos interlocuteurs : les informaticiens se limitent à l'essentiel. Les autres outils de documentation que peut offrir le L4G ne sont **pas exploités**, faute de temps, ou

parce qu'ils sont jugés peu adaptés aux méthodes de documentation préconisées par la direction informatique.

La **fonction "Help"** est très largement répandue, et tout aussi largement exploitée. Nos interlocuteurs se sont montrés unanimes sur ce point.

Les **gestionnaires d'écrans** sont également très répandus; leur usage est fort apprécié; ils évitent d'allourdir les programmes par des séquences lourdes et répétitives : de plus en plus de validations de données sont faites au niveau du gestionnaire d'écrans. En outre, signalons que MANTIS, dans sa dernière version, offre des possibilités de fenêtrage fort attrayantes.

Certaines des entreprises que nous avons visitées disposaient d'**outils graphiques** intégrés à leur L4G (FOCUS, IC1), ou rendus disponibles moyennant supplément de prix (ORACLE, NATURAL). Certains L4G ne disposent pas de tels outils, mais peuvent accéder à un logiciel extérieur (MANTIS, RALLY).

Tous les L4G dont il est question dans notre étude possèdent des **fonctions statistiques** - élémentaires-intégrées (moyenne, écart-type, minimum, maximum, totalisation), exceptés MANTIS et RALLY.

ORACLE, IC1 et FOCUS disposent en outre d'un **tableur**; celui d'ORALCE est particulièrment performant (SQL calc), puisqu'à chaque cellule peut être associée une quelconque sélection, insertion, mise à jour, suppression.

Dans la rubrique "Autres informations",épinglons l'existence de modules de recherche opérationnelle PERT (ORACLE, FOCUS), ou encore des modules d'import-export de données (ORACLE). Cette liste n'est pas exhaustive.

Section 2.3 : Organisation

La première question qui se pose est de savoir quelles sont les personnes qui, au sein de l'organisation, utilisent le L4G. La réponse doit être donnée en deux volets. Tout d'abord, il faut considérer le cas des L4G exclusivement **orientés développeurs** (MANTIS, RALLY); ceux-ci sont naturellement utilisés uniquement par les professionnels de l'informatique ! Il y a ensuite les L4G **orientés utilisateurs**, qui sont exploités à la fois par les utilisateurs finals et par les développeurs. Les utilisateurs finals peuvent être des ingénieurs, des employés, des secrétaires; deux de nos interlocuteurs nous

ont fait remarquer que la motivation (à exploiter les possibilités du L4G mis à leur disposition) diminue à mesure que l'on descend dans la hiérarchie ou que l'on gravit l'échelle de l'ancienneté. C'est ainsi qu'il est relativement aisé de distinguer plusieurs classes d'utilisateurs : les "presse-boutons", "qui ne font rien et ne veulent rien apprendre" (une minorité); les utilisateurs capables de faire des requêtes simples; et ceux, enfin, faisant des requêtes plus compliquées.

La tendance vers laquelle voudraient s'orienter nos interlocuteurs est d'arriver à une **identité complète** entre développeurs et utilisateurs, "tant du point de vue système que données", moyennant un contrôle intelligent de la part du service informatique (les utilisateurs peuvent consulter, mais pas mettre à jour n'importe comment).

Quand on pose la question à nos interlocuteurs de savoir si oui ou non l'informaticien joue un rôle de **consultant** auprès de l'utilisateur, la réponse ne se fait généralement pas attendre : c'est un "oui", haut et clair. Ici encore, cependant, il faut distinguer les L4G orientés développeurs, et ceux orintés utilisateurs.

Dans le premier cas, il nous a parfois été fait remarquer que le rôle de l'informaticien n'a pas changé suite à l'introduction du L4G; déjà "avant", l'informaticien consultait l'utilisateur et essayait de travailler en étroite collaboration avec lui. Par ailleurs, on nous dit que ce rôle de consultant dans le chef de l'informaticien "tend à prendre de soixante à quatre-vingts pourcents de son temps; il explique à l'utilisateur, le forme, et passe de moins en moins de temps au développement proprement dit". Un autre de nos interlocuteurs exprimait à peu près la même chose, en d'autres termes : "il faut supprimer au maximum les barrières; les utilisateurs doivent jouer pleinement leur rôle; ce ne sont pas les informaticiens qui doivent décider". Epinglons encore une autre remarque, fort intéressante : cette collaboration est nécessaire car "un utilisateur ne peut définir, au départ, qu'à peu près vingt pourcents de ses besoins".

Dans le second cas, c'est-à-dire celui où l'utilisateur exploite un logiciel dans le cadre d'un infocentre, les réponses ne sont pas moins unanimes : "informaticien-consultant ?". "Oui. A tous les niveaux, que ce soit pour le choix du matériel ou celui du logiciel. Dès que l'utilisateur rencontre un problème, il téléphone."

Au vu de ces résultats, il semble donc bien que l'informaticien ne soit plus ce personnage quelque peu

associable, perpétuellement reclus dans sa tour d'ivoire. Mais ce phénomène relativement récent est sans doute autant dû à une évolution des mentalités qu'à une évolution technique. De plus, rappelons, par souci de complétude, que ces réflexions émanaient généralement des informaticiens eux-mêmes !

Autre aspect organisationnel, l'infocentre. Selon les personnes que nous avons interviewées, un **infocentre** est un service offert à l'utilisateur, qui consiste à mettre à la disposition de ce dernier une série d'outils destinés à lui procurer une certaine autonomie en matière d'extractions simples de données contenues dans une base de données centrales. Cette autonomie accordée aux utilisateurs doit cependant être canalisée, nous font remarquer certains de nos interlocuteurs, en ce sens qu'il faut un gestionnaire de l'infocentre qui centralise les différentes questions et peut, le cas échéant, y répondre pour tout le monde (si une même demande est formulée plusieures fois). De plus, un infocentre n'a pas pour objet de se substituer à l'équipe d'informaticiens; il sert uniquement à satisfaire des demandes ponctuelles.

Une autre idée que nous avons entendue (à une reprise) concernant le concept d'infocentre, consiste à dire que chaque personne responsable d'un certain type d'information introduit et met à jour elle-même cette information et la rend disponible en consultation aux autres personnes de l'entreprise qui jouissent des droits d'accès requis.

Sur les neuf personnes interviewées, trois seulement nous ont déclaré disposer d'un infocentre. Les autres, pour la plupart, tendent vers cet objectif, mais sont conscientes du danger qu'un tel type d'organisation représente : "des précautions sont à prendre si l'on veut éviter une forte dégradation des performances"... Ces précautions concernent essentiellement la centralisation des requêtes, que nous avons évoquée , et la formation des utilisateurs, dont il est question par ailleurs dans ce compte-rendu.

En matière d'**intégrité** de la base de données, il faut envisager deux aspects. Tout d'abord, il y a l'intégrité physique : celle-ci est assurée par le logiciel, c'est-à-dire le SGBD (journalisation, principe des transactions, recouvrement en cas d'erreur,...). Ensuite, il y a l'intégrité logique; dans certains cas, celle-ci est assurée par une personne qui ne fait que cela (DBA); dans d'autres, c'est le service informatique en général qui en prend charge.

Section 2.4 : Les problèmes d'intégration

En ce qui concerne la **reprise des données**, certains systèmes de 4ème génération offrent la possibilité d'exploiter des données anciennes. Cette possibilité est réalisée soit par un utilitaire de chargement de données (exemple : transformation d'un fichier VSAM en table SQL et vice versa), soit par un accès direct par le L4G aux données anciennes (exemple : RALLY de Digital accède directement aux fichiers RMS). Il est évident qu'un L4G développé par un constructeur doit permettre la reprise des données exploitées par leurs anciens produits. De même une maison de logiciels qui lance un produit de 4ème génération devra offrir des interfaces avec des systèmes de gestion de données des grands constructeurs - ceci afin d'assurer une bonne pénétration du produit sur le marché. L'enquête révèle qu'une majorité des entreprises (7/9) ont repris des données anciennes pour les exploiter dans un environnement de 4ème génération.

En ce qui concerne la **reprise des programmes** existants, 7 entreprises sur 9 développent de nouvelles applications à l'aide du L4G. Il est important de souligner qu'un L4G (RALLY) a été acheté dans le cadre du développement d'une nouvelle application. D'autre part, une minorité des entreprises ont procédé à certaines conversions ou adaptations de leurs anciens programmes. Une raison évoquée par un interviewé est la suivante : "La conversion des applications existantes en L4G coûte très cher car toute la logique de l'application doit être repensée".

Parmi les langages de 4ème génération qui apparaissent dans cette étude, ORACLE est certainement celui qui offre le plus de souplesse dans le domaine de la **transportabilité** des applications car seulement 4 % du code source d'ORACLE dépend de la machine cible. Dès lors, des applications peuvent être développées sur de gros ordinateurs et ensuite, elles peuvent être transférées, à peu de frais, sur des mini- et/ou micro-ordinateurs. A l'opposé d'ORACLE, les applications développées par RALLY (L4G de chez Digital) ne sont pas transportables sur des équipements autre que Digital. A partir de ces deux extrèmes, on peut dégager deux tendances. D'une part, la politique des constructeurs est de développer un produit de 4ème génération uniquement pour leur matériel. D'autre part, les maisons de logiciels qui offrent un L4G essayent de rendre les applications développées sur une machine transportables vers d'autres machines, mais avec une granularité différente : - machines différentes et constructeurs différents; - machines différentes mais même

constructeur ; - machines différentes, environnement semblable, même constructeur.

La plupart des personnes interviewées n'ont pas eu de gros problèmes techniques d'intégration du L4G au sein de l'entreprise. Au niveau des **ressources**, certaines entreprises ont d'emblée augmenté la taille de la mémoire et la vitesse du processeur. A titre exemplatif, l'exploitation d'une base de données relationnelle SQL par opposition aux fichiers classiques "COBOL" a nécessité deux fois plus de mémoire ainsi qu'un processeur deux fois plus rapide. La nécessité d'augmenter les ressources provient de deux facteurs :
- le L4G consomme plus de ressources que les langages de haut niveau;
- bien souvent, l'introduction d'un L4G rend l'utilisateur final plus actif; ce qui impose l'installation de nouveaux terminaux qui chargent le système.
Bien conscientes de la nécessité d'augmenter les ressources machine, les entreprises concernées par l'enquête ont très bien **anticipé** ce phénomène afin d'éviter un essoufflement de leur machine. Toujours dans le cadre des problèmes techniques d'intégration, on peut souligner les difficultés minimes rencontrées chez un utilisateur d'ORACLE en version pour "personal computer". Intuitivement, on peut déjà s'interroger sur la complétude des fonctions offertes dans la version destinée aux micro-ordinateurs par rapport à celles offertes sur les plus grosses machines.

- Au niveau du générateur d'applications d'ORACLE, la fonction qui permet de grouper dans une même page physique des données provenant de plusieurs tables n'est pas opérationnelle.

- Le générateur d'applications d'ORACLE ignore les codes ASCII supérieurs à 127. Ce détail présente certains inconvénients sur la version destinée aux micro-ordinateurs car ceux-ci disposent bien souvent d'un clavier accentué.

- Pour certaines applications, l'utilisateur aimerait développer dans un langage comprenant des structures conditionelles. Pour résoudre ce manquement, ce dernier est obligé d'effectuer des instructions "tout à fait aberrantes" pour simuler une structure conditionelle.

En ce qui concerne la **formation**, on observe à travers cette enquête deux démarches. La **première**, qui consiste à former des personnes aux L4G d'une manière **approfondie**, dure en général une à deux semaines. Ce programme s'adresse aux spécialistes possédant déjà une formation en

informatique. Le nombre de personnes qui suivent ces cours, bien souvent externes à l'entreprise, se situe entre 1 et 10. Ce nombre peu élevé s'explique par le coût élevé d'une telle formation. La **deuxième** démarche propose une formation à l'**utilisateur**. Cette formation s'adresse à des novices en informatique et elle dure en moyenne 1 ou 2 jours. Elle est réalisée, en général, par les informaticiens qui ont suivi les cours approfondis. Le nombre d'utilisateurs finals ainsi formés dépend de la taille de l'entreprise. A propos de la deuxième démarche, il est intéressant de souligner le procédé de deux entreprises. En effet, la formation de l'utilisateur final s'effectue en **deux** séances. La première séance est consacrée à un exposé des fonctionnalités du langage et à la présentation de plusieurs exemples. Après un mois pendant lequel l'utilisateur dispose de l'outil, une deuxième séance est organisée afin de rappeler, pour certains, que le produit existe mais surtout pour répondre aux interrogations des utilisateurs qui ont déjà travaillé avec l'outil. Ce procédé motive l'utilisateur final. Dans les deux démarches de formation, une formation auto-didacte peut se concevoir pour les personnes les plus brillantes. Dans le cadre de certains langages de 4ème génération (MANTIS), seule la première démarche de formation est effectuée. En effet, le produit MANTIS s'adresse essentiellement aux développeurs.

Au point de vue humain, l'intégration d'un L4G présente plusieurs avantages. En ce qui concerne les développeurs, les L4G offrent un environnement de travail beaucoup **moins contraignant** que les environnements classiques. L'informaticien trouve un certain plaisir à créer de nouvelles applications car le L4G supprime certaines parties rébarbatives du développement classique. En ce qui concerne l'utilisateur final, l'introduction d'un L4G offre à ce dernier une plus grande **autonomie**. L'indépendance de l'utilisateur par rapport à l'informaticien procure chez le premier une grande motivation qui provient, entre autres, d'une augmentation de sa responsabilité. A titre exemplatif, autrefois l'informatique produisait de "gros listings" dans lesquels chaque utilisateur allait extraire les informations nécessaires à son besoin. Bien souvent, une série de traitements étaient effectués manuellement sur base des informations extraites. A l'heure actuelle, une simple requête peut extraire les informations d'une base de données et les intégrer dans un tableur. Les personnes interrogées ne disposent pas encore d'une telle fonction mais elles la trouvent très intéressante. L'instauration d'un **dialogue** entre les utilisateurs et les informaticiens est un avantage cité par une personne interviewée dans le cadre de notre enquête. Cet avantage provient du fait que

les modifications à apporter à un logiciel sont aisées à réaliser par les informaticiens.

D'une manière générale, on note très peu d'inconvénients résultant de l'intégration d'un L4G dans une entreprise. Il existe, néanmoins, des personnes **réticentes** à l'utilisation de l'informatique et donc à l'utilisation des L4G (malgré les efforts soutenus des responsables de la formation). Toutefois, on remarque une stimulation entre les utilisateurs, une entraide spontanée entre les motivés et les "bloqués". Le fait de devoir repartir "à zéro" au point vue formation apparaît aussi comme un handicap à l'introduction d'un L4G. Enfin, les "gens" ont peut-être une fausse idée à propos des L4G. Dès lors, ils croient que les L4G permettent de "faire n'importe quoi".

Section 2.5 : Aide à la décision

En Belgique, la plupart des décisions prises dans les entreprises ne font pas l'objet d'une étude approfondie. Les gestionnaires n'ont pas les moyens de mettre en oeuvre une étude approfondie. Les outils d'aide à la décision, composants de certains L4G, sont-ils les précurseurs d'une nouvelle génération de décideurs ?

Il est important de noter que certains L4G (exemple : ADABAS/NATURAL) offrent très peu d'outils d'aide à la décision. Parmi les personnes contactées, l'une d'entre elles rappelle brièvement les deux fonctions essentielles d'un système d'aide à la décision. Le logiciel extrait des informations d'un système d'information puis, les traite, d'une manière rapide. Les composants du S.A.D. qui réalisent ces deux fonctions seront donc d'une grande utilité. D'une manière plus précise, on peut établir la liste des composants les plus utiles lors d'une prise de décision :

- analyse des données (fonctions mathématique et statistique);

- dans une moindre mesure, les graphiques.

A propos des graphiques, les avis de nos interlocuteurs sont très partagés :

"Les graphiques ne représentent pas un besoin immédiat ni important dans notre entreprise";

"Les graphiques sont encore très peu utilisés dans les entreprises belges".(sous-entendu "trop peu");

"Les graphiques sont importants si on n'a pas le temps de lire les chiffres".

Après ces réflexions très générales, nos interlocuteurs sont d'accord pour reconnaitre que les graphiques restent malgré tout créés par l'être humain. Et, dès lors, c'est avant tout l'imagination de l'utilisateur qui doit fonctionner pour produire un graphique facilement interprétable.

En ce qui concerne l'amélioration de la **qualité** de l'information, le critère de précision apparaît très souvent dans notre enquête. En effet, autrefois les décideurs se contentaient de dégager les tendances très générales d'une prise de décision. A l'heure actuelle, à l'aide d'outils comme le tableur, les gestionnaires ont la possibilité d'effectuer avec précision des calculs indispensables avant la décision finale. L'abondance d'information apparaît comme un critère moins important que la précision. Il est évident que cette abondance améliore la qualité de l'information dans la mesure où toute l'information est accessible. Dans notre enquête le critère "présentation" n'améliore guère la qualité de l'information.

Section 2.6 : Développement d'applications

Nos interlocuteurs ont éprouvé certaines difficultés à classer leurs applications dans les catégories lourde, moyenne et légère. Certaines difficultés sont apparues aussi pour déterminer la durée de développement en L4G (en nombre de jours/homme).Toutefois, certaines tendances sont apparues :

- plus l'application est **lourde** moins le gain de temps du développement en L4G par rapport à un langage de haut niveau est important. Dans le même ordre d'idées, on note que plus une application est **complexe** moins le gain de temps en développement en L4G par rapport à un langage comme le Cobol est important;

- dans un environnement SQL + Cobol, le gain de temps ne se manifeste pas à l'écriture des programmes mais plutôt pendant les phases de **conception** et de **maintenance**;

- gain de temps surtout pour les applications qui contiennent beaucoup de **saisies** et de **contrôles** de données (ORACLE);

- avec un environnement L4G, le gain de temps est plus important dans la phase de maintenance que dans la phase de développement;

- les applications développées avec un L4G manquent de **raffinements**.

Au point de vue de la productivité, la section 8 approfondit le sujet.

En ce qui concerne le générateur d'applications, notre enquête a révélé peu d'entreprises qui avaient recours à cet outil pour créer des applications.

La plupart des entreprises concernées revendiquent l'utilisation du **prototypage** et y voient beaucoup d'avantages :

- l'utilisateur n'est pas surpris en fin de processus;

- l'utilisateur ne sait pas très bien ce qu'il veut. La technique du prototypage permet de prendre en considération, d'une manière rapide, les exigences de l'utilisateur final;

- le prototypage permet aux développeurs, d'une part de visualiser leur travail, et d'autre part de montrer une partie du projet aux clients ou aux futurs utilisateurs. A l'opposé, les applications développées dans un langage comme le Cobol qui n'offre pas des fonctions de prototypage doivent être complètement terminées avant la démonstration chez l'utilisateur ou le client;

- les résultats sont plus proches des exigences de l'utilisateur.

L'avantage des L4G est de produire un prototype à un coût acceptable.

L'un de nos interlocuteurs considère que le prototypage ne doit pas être effectué d'une façon systématique ou absolue. Le prototypage est la partie visible de l'application. Dès lors, comme dans le cas d'un iceberg, il est parfois dangereux de cacher à l'utilisateur un certain nombre de données au sujet de l'application. Dans certaines applications, le prototypage réduit le niveau de réflexion des utilisateurs. A titre exemplatif, le fait d'établir un prototype pour une application comme "la gestion des stocks" donne à

l'utilisateur l'impression que tout est simple. L'informatique doit savoir imposer certaines façons de procéder. Les utilisateurs doivent discerner le réalisable du non-réalisable. La technique du prototypage n'est pas instaurée pour prendre en charge les exigences superflues et non justifiées des utilisateurs.

En ce qui concerne le **rôle de l'utilisateur** pendant le développement d'une application en L4G, environ la moitié des personnes interrogées (4/9) affirment que ce rôle n'a pas changé. Dans ce cadre, les langages de 4ème génération sont considérés uniquement comme des outils d'aide au développement. Par contre, dans certaines entreprises, l'introduction d'un L4G a donné, à l'utilisateur, un rôle plus actif qui peut être perçu à deux niveaux :
- la **communication** entre l'informaticien et l'utilisateur s'est accrue considérablement. En effet, l'informaticien montre facilement le résultat de son travail et il sait prendre en considération, d'une manière aisée, les modifications formulées par l'utilisateur;

- certains utilisateurs qui reçoivent un courte formation (cfr. supra) développent eux-même leurs applications légères et ponctuelles (exemple : consultation d'un fichier).

La majorité (6/9) des personnes contactées estiment que les langages de 4ème génération ne se substituent pas entièrement aux langages de haut niveau (Cobol, PL1,...). Les raisons évoquées sont les suivantes :

- les L4G conviennent uniquement pour la gestion de données (RALLY);

- l'impression des états est faite en Cobol et en batch (MANTIS);

- les grosses applications (comptabilité, gestion de stocks, ...) ne sont pas prises en compte par les L4G (FOCUS);

- 90 % des applications peuvent être développées à l'aide d'un L4G. Dans les autres cas, où un degré de raffinement sera exigé, on aura recours aux langages de la génération précédente (ADABAS/NATURAL);

- la faiblesse de SQL ne permet pas de se passer d'un autre langage pour le développement d'applications.

D'autres personnes suggèrent d'abandonner le Cobol, d'avoir recours éventellement au langage C pour traiter des problèmes très spécifiques et d'acheter des progiciels

pour des applications comme la comptabilité. Tout le reste est développé à l'aide d'un L4G comme ORACLE.

L'enquête, réalisée aux Etats-Unis par INPUT pour le compte d'ADR, consistait notamment à demander aux utilisateurs d'IDEAL d'évaluer le pourcentage des applications qui peuvent être prises en charge par ce L4G. Le pourcentage moyen se chiffre à 70 % . Il est intéressant d'analyser la distribution des réponses (cfr. figure page suivante). Cette analyse conduit à quelques conclusions intéressantes :

- 23,7%des utilisateurs d'IDEAL estiment que 80 à 90 % des applications peuvent être écrites en IDEAL;

- 55,3 % pensent que plus de 80 % des application peuvent être écrites en IDEAL;

- environ 1 personne sur 6 estime que 90 à 99 % des applications peuvent être supportées par IDEAL.

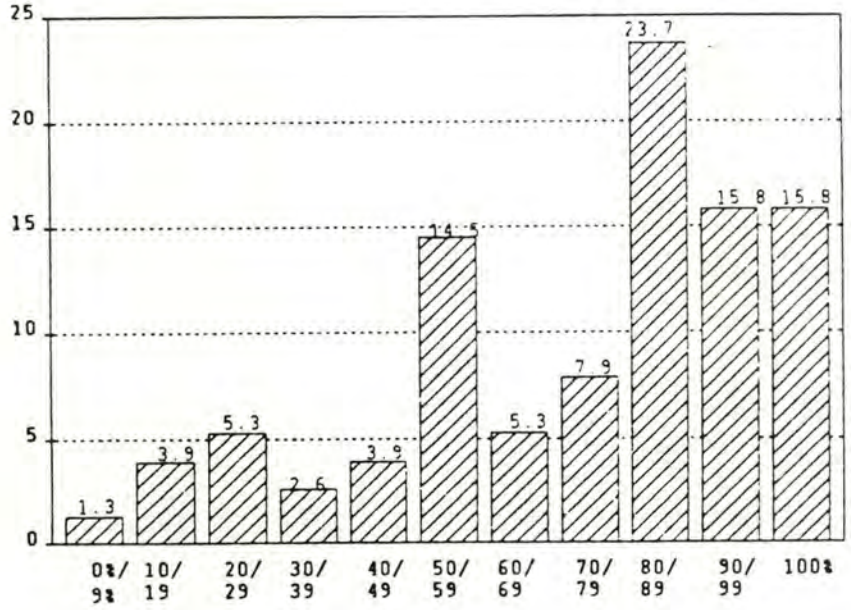


Figure II.2 : distribution du pourcentage des applications qui peuvent être développées par le L4G IDEAL

Section 2.7 : Maintenance

Les utilisateurs de L4G que nous avons rencontrés considèrent tous que la **maintenance** de leurs programmes est **allégée**, mais bien plus grâce au **langage de très haut niveau** que grâce à l'outil de documentation offert par leur L4G. Ceci ne fait que confirmer ce que nous avons déjà pu constater à la section 2 : la documentation des programmes est souvent négligée, faute de temps, et l'outil de documentation n'apporte pas un remède dans ce domaine.

Certains de nos interlocuteurs n'ont pas manqué de souligner que, s'il est vrai que les programmes rédigés en L4G sont en général faciles à maintenir, il est aussi vrai que "si on atteint les limites du langage, c'est la pire des choses".

"Mais malgré tout, concluait un de nos interlocuteurs, l'approche méthodologique est toujours nécessaire".

Ainsi, nous constatons que les craintes que nous formulions dans la première partie de cette étude quant à la difficulté de maintenir des applications développées par des utilisateurs non informaticiens ne se trouvent pas confirmées par notre enquête. C'est plutôt une bonne nouvelle, mais, sans vouloir jouer les trouble-fête, constatons cependant que les L4G orientés utilisateurs n'étaient pas installés depuis très longtemps dans les entreprises que nous avons visitées. De plus, dans de nombreux cas, les responsables de l'informatique canalisent quelque peu l'enthousiasme de certains utilisateurs en ne leur permettant que la consultation de données. Ceci explique peut-être cela ...

Quant à la question de savoir qui assure la maintenance des applications, tous nos interlocuteurs pourraient y répondre d'une même voix : "celui qui l'a développée ! ". Il semble bien que la situation où celui qui avait développé une application n'était plus présent (parce qu'il avait changé d'entreprise ou pour toute autre raison) au moment où il fallait la maintenir ne se soit pas encore présentée très souvent. Deux, seulement, des personnes interviewées y ont fait allusion :

- "quand la personne n'est pas là pour maintenir son application, celle-ce revient au service informatique";

- "nous essayons d'établir des standards, des conventions, de manière à ce que l'on puisse s'y retrouver dans une application développée par un autre".

Des **problèmes** particuliers de maintenance ? Les personnes que nous avons interviewées n'en déplorent aucun. L'une d'entre elles cependant, constate "un manque de documentation et un manque de méthode pour l'analyse et la programmation". Comme quoi le L4G ne résoud pas tous les problèmes, loin s'en faut, et surtout pas ceux inhérents à la méthodologie de développement des logiciels. Une autre déclare son produit "trop nouveau pour avoir pu rencontrer de vrais problèmes de maintenance".

Section 2.8 : Productivité

Chiffrer un gain de **productivité** en matière de développement d'applications n'est pas chose facile. Nous avons pu nous en rendre compte lors de notre enquête : à la lecture de cette question, les visages de nos interlocuteurs affichaient généralement une réelle perplexité ... Les chiffres extrêmes font état d'un **gain** de productivité de 1,3:1 à 10:1. Certains indiquent que ce n'est pas tellement lors de l'écriture des programmes que le gain de productivité apparaît, mais plutôt lors de la maintenance de ceux-ci. D'autres font état de gains variables en fonction de la dimension des programmes : "pour des applications légères et répétitives, le gain peut être de 8:1 en faveur du L4G; mais pour des applications lourdes, ce rapport tombe à 2:1 et même moins". "En fait, conclut un de nos interlocuteurs, ce rapport est fonction du type d'application et de l'expérience du programmeur avec le L4G".

De nouveau, nous allons nous référer à l'enquête américaine [INPUT 85], pour comparer les résultats. Cette enquête révèle que , pour le produit IDEAL d'ADR, l'accroissement moyen de productivité par rapport au Cobol est de 4,4:1. Compte tenu d'un intervalle de confiance de 90 %, ce taux est compris entre 4,1:1 et 4,7:1 parmi les 65 personnes (81,3 % de l'échantillon) ayant répondu à la question. De plus, plus de 15 % des répondants estiment obtenir un gain de productivité supérieur ou égal à 10 tandis que pour 23 % d'entre eux ce taux est inférieur à 4:1.

Toutes les personnes interrogées ayant à leur disposition un logiciel d'aide à la décision sont très satisfaites de ce produit. Aucun ne peut estimer avec précision le gain financier que peut procurer un S.A.D.,

mais toutes sont persuadées que "c'est rentable" ou encore "qu'il a déjà été amorti plusieurs fois". En matière de délai de réalisation des études, les avis sont tout aussi unanimes : c'est beaucoup plus rapide, car on ne doit pas passer par un informaticien". De toute façon, tous nos interlocuteurs sont convaincus que "pour une bonne gestion, c'est important ! ...".

Section 2.9 : Synthèse de l'enquête par des tableaux

Raisons	Proportion
Productivité	7/9
Qualité du service rendu	4/9
Concept d'infocentre	3/9
Réduction de la maintenance	3/9
Approche base de données	1/9
Possibiltés prototype	1/9

Tableau II.2 : raisons du choix d'un L4G

Outils	très utilisé	utilisé	peu utilisé
Générateur de rapports	X		
Générateur d'applications			X
Outils de documentation			X
Fonction HELP intégrée	X		
Gestionnaire d'écrans	X		
Outils graphiques	pas de réponse significative		
Fonctions statistiques			

Tableau II.3 : possibilités techniques exploitées

Thèmes abordés	Tendance Oui	Non
Reprise des données	X	
Reprise des programmes ou conversions		X
Problèmes au niveau des ressources		X

Tableau II.4 : Intégration

	très utilisé	pas assez utilisé	ordre d'importance
Analyse de données	X		1
Graphique		X	2
Qualité de l'information			
Précision			1
Abondance			2
Présentation			3

Tableau II.5 : aide à la décision

Type d'application où le gain est important - légère, peu complexe - applications de saisie et de contrôle de données (ORACLE)
Quand le gain est-il important ? - maintenance - conception et maintenance (SQL + COBOL)
Rôle de l'utilisateur : 4/9 inchangé
Substitution L4G/L3G : non 6/9

Tableau II.6 : développement d'applications

Prototypage	
Avantages	Inconvénients
l'utilisateur n'est pas surpris prise en compte des exigences de l'utilisateur visualiser l'état d'avancement du travail résultat plus proche des exigences de l'utilisateur	cache des données à l'utilisateur utilisateur moins raisonnable limite le niveau de réflexion de l'utilisateur

Tableau II.7 : développement d'applications

Maintenance
- allégée grâce au langage de très haut niveau - l'outil de documentation du L4G a peu d'impact sur la maintenance
Productivité
Chiffre : augmentation de 30 à 1000 % Fonction de l'application : - petite, légère : productivité élevée - complexe, lourde : productivité peu élevée

Tableau II.8 : commentaires sur la productivité et la maintenance

Section 2.10 : Commentaires sur la conduite de l'enquête

a) Le questionnaire à l'épreuve de nos interlocuteurs

La formule des "questions ouvertes" pour laquelle nous avons opté n'est pas restée sans conséquences sur la façon dont nos interlocuteurs ont répondu au questionnaire qui leur était soumis.

A plus d'une reprise en effet, nous avons pu constater que certains de nos interlocuteurs s'éloignaient de la portée de la question qui leur était posée. Bien souvent, dans ce cas, ils abordaient alors des thèmes que nous avions prévu de traiter par la suite dans notre entrevue ! Bien que ce phénomène ne soit pas un drame en soi, il nous a néanmoins contraint à restructurer la plupart des réponses que nous avons récoltées : c'est là le prix à payer d'une discussion à bâtons rompus...

Le deuxième problème engendré par un questionnaire "ouvert", et qui découle directement du premier, est que, si au début de l'entrevue, la personne interviewée se montre parfois extrèmement prolixe, il peut en être tout autrement après une heure ou plus d'entretien : le questionnaire peut alors lui paraître trop long ou trop verbeux, et certaines questions ne se voient peut-être pas accorder toute l'attention qu'elles méritent...

Pour ces raisons, sans doute eût-il été préférable de rédiger un questionnaire plus concis, aux questions plus ponctuelles,... sans pour autant sacrifier aucun des aspects qu'il nous tenait à coeur d'aborder !

En ce qui concerne maintenant la clarté des questions posées, il faut reconnaître qu'elle s'est révélée en général très satisfaisante, sauf pour les questions 6.1 et 6.4.

La première était relative aux types d'applications développées à l'aide d'un L4G; cette question était sans doute mal formulée, car non seulement nos interlocuteurs ne faisaient généralement pas la distinction entre applications lourdes, moyennes et légères, mais encore ils se limitaient à donner un rapport L4G/COBOL en nombre de jours de développement, sans autres précisions (alors que ce rapport fait l'objet d'une question particulière dans la section 8).

La question 6.4, quant à elle, consistait à essayer de percevoir l'évolution du rôle de l'utilisateur final lors du développement d'applications, d'une part suivant l'approche L4G, et d'autre part suivant une approche plus traditionnelle. Il semble bien que cette question, présentée sous forme de tableau, fut mal comprise, puisqu'aucun de nos interlocuteurs n'y a répondu de la manière que nous attendions : à chaque fois, ou peu s'en faut, il nous était demandé de reformuler la question "en français" !...

Ce problème de clarté des questions 6.1 et 6.4 aurait sans doute pu être évité si nous avions prétesté notre questionnaire. Cependant, ainsi que nous l'avons déjà évoqué, une telle technique eût été démesurée, eu égard à la faiblesse de notre échantillon.

b) Un démarche opposée ?

Nous avons voulu marquer la conduite de notre enquête du sceau de la qualité. Echantillon faible, entrevues directes, questions ouvertes : ces options ont été délibérément retenues dans le but de satisfaire au mieux nos exigences.

Il eût été possible d'aborder notre étude de manière plus quantitative, plus orientée "sondage d'opinion". Echantillon le plus élevé possible, contacts par courrier, questions plus fermées (style "QCM") sont les ingrédients d'une telle démarche. Cependant, cette façon de voir les choses n'aurait pas manqué de poser certains problèmes.

Tout d'abord, trouver un échantillon de 50 voire 100 entreprises ayant acquis un L4G n'est pas chose aisée : les langages de quatrième génération n'ont pas encore fait l'unanimité dans notre pays ! De plus, en supposant même qu'il soit possible d'obtenir une centaine d'adresses d'adeptes de L4G, le taux de réponse espéré ne devrait pas dépasser 50 pourcents. Dès lors, il faudrait, au départ, doubler le nombre de personnes à contacter, ce qui accentue encore le problème de la relative rareté des entreprises disposant d'un L4G.

Ensuite, un questionnaire fermé est incapable de rendre compte de toutes les subtilités que les personnes contactées désirent généralement apporter à leurs réponses.

Enfin, une enquête par courrier supprime tout contact

humain, souvent bénéfique et enrichissant, même s'il s'agit là d'un domaine beaucoup moins tangible...

Quoi qu'il en soit, et pour conclure cette section, il semble bien qu'il eût été difficile d'imaginer, dans le cadre relativement restreint de cette étude, une méthode de travail fondamentalement différente...

CONCLUSION

De manière générale, les entreprises que nous avons visitées se montrent satisfaites du produit de quatrième génération qu'elles ont acquis.

Les raisons qui les avaient motivées à se doter d'un tel outil - essentiellement, l'amélioration de la productivité et la réduction de la maintenance - se sont révélées, à l'usage, conformes aux promesses que les L4G laissaient entrevoir dans ce domaine, bien que, dans le cas de la productivité, les gains soient très variables d'un type d'application à un autre.

Les L4G orientés développeurs peuvent ne pas impliquer de profonds boulversements organisationnels; le rôle de l'informaticien reste relativement inchangé. Par contre, les L4G orientés utilisateurs nécessitent bien évidemment une participation active de l'utilisateur final, participation qui se voit souvent concrétisée par l'apparition, explicite ou non, d'un infocentre. Le rôle de l'informaticien s'apparente alors plus à celui d'un consultant; en même temps, l'informaticien se doit d'assurer, avec ses collègues, une gestion rigoureuse des données et des utilisateurs.

L'implantation d'un nouveau logiciel doit permettre d'utiliser les données existantes; les L4G posent peu de problèmes à ce niveau. Mais le langage de quatrième génération n'est utilisé en général que pour développer de nouvelles applications, et pas pour en reprendre d'anciennes : le coût en est trop élevé. Quant au problème de la transportabilité, il est spécifique à chaque logiciel.

Un langage de quatrième génération est gros consommateur de ressources; c'est une évidence qui n'a jamais été mise en question et qui a d'emblée été prise en compte par les entreprises, qui trouvent plus intéressant financièrement d'accroître leurs ressources matérielles et logicielles, plutôt que la taille de leur service informatique.

Une formation - plus ou moins longue selon le degré de spécialisation escompté - est nécessaire pour une utilisation optimale d'un L4G, notamment pour les décideurs qui, cependant, en Belgique, sont très loin de faire un usage intensif des S.A.D. que leur procure leur L4G.

Au chapitre des problèmes que pourrait susciter l'utilisation par un novice d'un outil logiciel très puissant, nos craintes ne se sont guère vues confirmées par nos interlocuteurs. Les responsables de l'informatique ont, semble-t-il, bien anticipé les risques d'une telle démarche, en n'accordant aux utilisateurs qu'une "liberté surveillée" : ce que le désir de pleinement satisfaire l'enthousiasme de certains utilisateurs pouvait leur conseiller, le sens des responsabilités le leur interdisait.

Tout cela peut paraître fort séduisant. Néanmoins, rappelons, encore une fois, que notre échantillon n'était pas statistiquement représentatif, et que nous gardons la conviction que les problèmes exposés dans la première partie de cette étude sont très susceptibles de se poser si l'on n'y prend garde.

Terminons, enfin, par un souhait. Nous espérons que cette étude, éventuellement complétée par une autre, plus technique, pourrait servir de base de réflexion à une entreprise désireuse de se porter candidate à l'acquisition d'un langage de quatrième génération...

B I B L I O G R A P H I E

[ADER 84] Ader, 1984, Le choc informatique, Edition Denoël, Paris 7ème, 19 rue de l'Université

[BHLP] Bodart, Hennebert, Leheureux, Pigneur, "Computer-Aided Specification, Evaluation an Monitoring of Information Systems" FNDP, Namur; HEL, Lausanne

[BOPI 83] Bodart, Pigneur, 1983, Conception assistée des applications informatiques : 1. Etude d'opportunité et analyse conceptuelle, Masson, Paris

[BROQ 84] Broquet Claude, 1984, Gestion financière de base, tomes 1 et 2, Edition CEFUC, Mons

[CHAP 84] Chapin, Jan 1984, "Software maintenance with fourth- generation languages", ACM Sigsoft software enginneering notes, Vol 9, No 1

[CONT 87] Conte, 1987, "Understanding Relational Data Bases", in Computer Language, Vol 4, Nr 5, San Fransisco

[DATE 81] Date, 1981, An Introduction to Database Systems, Addison Wesley Publishing Company

[DEAD 83] Delobel, Adiba, 1983, Bases de données et systèmes relationnels, DUNOD, Paris

[DEBL 83] De Blasis, 1983, Les enjeux clés de la bureautique, Les Editions d'Organisation, Paris

[DEBR 87] De Brabandère, 1987, Parlons Futurs, Les Editions de La Libre Belgique, Bruxelles

[FPS 87] Finkelstein, Patterson, Sirta, 1987, "Fourth Generation Language Data Bases", in Computer Language, Vol 4, Nr 5, San Fransisco

[HAIN 84] Hainaut, janvier 1984, "Introduction à SQL, un système de gestion de base de données relationnelles, Institut d'informatique, Namur

[HAIN 85] Hainaut, 1985, "Caractéristiques d'un SGBD relationnel - Grille d'analyse et exemple d'utilisation", Institut d'informatique, Namur

[HAIN 86 A] Hainaut, juin 1986, "La mini-informatique : état actuel de la technologie en matière de matériel et de logiciel", Institut d'informatique, Namur

[HAIN 86 B] Hainaut, septembre 1986, "Introduction aux systèmes relationnels et aux langages de quatrième génération - Synthèse de la conférence d'ouverture, ASAB/VEBI - Congrès CIAO", Institut d'informatique, Namur

[HAIN 86 C] Hainaut, 1986, Conception assistée des applications informatiques : 2. Conception de la base de données, Masson, Paris

[HINE 85] **Hines**, 1985 Office Automation - Tools and Methods for System Building, John Wiley & Sons, New York

[IRIA 71] Bulletin de l'IRIA - Spécial APL, mars 71

[KLEI 80] Kleijnen Jack, 1980, Computers and profits : quantifying financial - benefits of information, Addison - Wesley Publishing Company, Amsterdam

[LESU 86] Lesuisse, 1986, Bureautique, FNDP, Namur

[MART 83] Martin, 1983, L'informatique sans programmeurs, Edition d'organisation, Paris

[SABO 81] Saboureau, 1981, Contrôle des performances d'un centre informatique, Edition Hommes et techniques, Boulogne - Billancourt

[VANB 86] Van Bastelaer, 1986, Notes pour un cours de téléinformatique, Facultés Universtaires Notre Dame de la Paix, Namur

[VANH 86] Van Houtte, 1986, "Les langages de quatrième génération conjugués au futur", FNDP, Namur

[VERM 86] Vermeire, octobre 1986, "Le software étranglé", Intermédiaire , n°35

[VLAM 82] Van Lamsweerde, 1982, "Les outils d'aide au développement de logiciels : un aperçu des tendances actuelles", Institut d'Informatique de Namur, XVème journées internationales de l'informatique et de l'automatisation, Paris

[VLAM 85] Van Lamsweerde, août 1985, "Cadre général pour un modèle de cycle de vie d'un projet informatique",

[INPUT 85] INPUT Inc.,"A survey of active users of the ADR/Ideal Fourth Generation Application Development System, Saddle Brook, November 1985

M A N U E L S D E R E F E R E N C E

ADR/Ideal Applied Data Research, 1985, "Application Development Reference Manual"

MANTIS Cincom Systems, 1985, "Design Facility Tutorial", "Programming Facility Tutorial", "Prototyping Supplement", "Design Facilities Reference Manual", "TIS/XA", "File View Design Reference Manual"

ORACLE, 1984, "Oracle Overview and Introduction to SQL"

ANNEXES

ANNEXE A : Q U E S T I O N N A I R E

Statut du questionnaire

Les données mentionnées dans ce questionnaire seront traitées de manière strictement confidentielle; elles seront exploitées uniquement dans le cadre d'une étude scientifique, et en aucun cas à des fins commerciales. La synthèse ne fera pas état d'une personnalisation des réponses.

1 - OUTILS

Système de quatrième génération.

1.1 Nom de votre langage de quatrième génération :

1.2 Nom du fabricant :

1.3 Opérationnel depuis : _ _ / 19 _ _

Ordinateur.

1.4 Marque :

1.5 Modèle :

1.6 Système d'exploitation :

Environnement humain.

1.7 Nombre d'employés-cadres travaillant dans l'entreprise :

1.8 Taille du service informatique :

Raisons du choix.
++++++++++++++++++

1.9 Quelles sont les raisosns qui vous ont poussé à acquérir un système de quatrième génération en général ?

1.10 Quelles sont celles qui ont motivé votre choix pour le système particulier que vous utilisez ?

2 - POSSIBILITES TECHNIQUES EXPLOITEES

Point de vue du développeur

2.1 Votre L4G offre-t-il un langage d'accès aux données de type SQL ou équivalent ?
Quel est son degré d'utilité (jamais utilisé, parfois utilisé,.....)

2.2 Votre L4G possède-t-il un générateur de rapports intégré ?
Quel est son degré d'utilité ?

2.3 Votre L4G possède-t-il un générateur d'applications ?
Quel est son degré d'utilité ?

2.4 Votre L4G possède-t-il des outils de documentation ?
Quel est son degré d'utilité ?

2.5 Votre L4G offre-t-il des "HELP" intégrés ?
Quel est son degré d'utilité ?

2.6 Votre L4G offre-t-il un gestionnaire d'écrans ?
Quel est son degré d'utilité ?

2.7 Autres informations.

Point de vue de l'utilisateur (S.A.D.)

2.8 Votre L4G offre-t-il un langage d'accès aux données de type SQL ou équivalent ?

2.9 Votre L4G possède-t-il des fonctions graphiques ?

2.10 Votre L4G possède-t-il des fonctions statistiques intégrées?

2.11 Votre L4G possède-t-il un tableur ?

2.12 Autres informations.

3 - ORGANISATION

3.1 Qui utilise le L4G ? Quelle relation y a-t-il entre l'informatique de l'utilisateur et l'informatique opérationnelle (indépendance complète, identité complète, travail sur copie de la BD, BD unique,...) ?

3.2 L'informaticien a-t-il un rôle de consultant auprès de l'utilisateur ?

3.3 Pour vous, qu'est-ce qu'un infocentre ?
En avez-vous un dans votre entreprise ?

3.4 Existe-t-il un responsable qui maintient l'intégrité de la base de données, celle des applications ?

4 - PROBLEMES D'INTEGRATION

Point de vue technique.

4.1 La reprise des données et programmes existants a-t-elle été possible ?

4.2 La transportabilité des applications est-elle réalisable à peu de frais ?

4.3 Existe-t-il des interfaces avec des systèmes de gestion de données courants (VSAM, ISAM,...)

4.4 Avez-vous rencontré d'autres problèmes techniques d'intégration (ressources,...) ?
Quelle solution y avez-vous apporté ?

4.5 Subsiste-t-il des problèmes techniques d'intégration ?

Aspects humains.
++++++++++++++++

4.6 Suite à l'achat du L4G, un ou plusieurs membres du personnel ont-ils suivi un cycle de formation ? A l'intérieur ou à l'extérieur de l'entreprise ? Combien de temps a duré cette formation ?
Indiquez dans la case adéquate le nombre de personnes ayant suivi cette formation et le type de formation suivie.
(T = Technique, M = Méthodologique, TM = Technique et Méthodologique)

FORMATION	D U R E E			
	-1 jour	1 jour	- 5 jours	plus (précisez)
EXTERIEUR				
INTERIEUR				

4.7 Quels avantages et inconvénients voyez-vous à l'intégration d'un L4G au sein de votre entreprise (aspects humains) ?

4.8 Comment avez-vous résolu les problèmes humains d'intégration ?

5 - AIDE A LA DECISION
==========

5.1 Parmi les composants du S.A.D. enumérés au point 2, lesquels sont les plus utiles lors des prises de décision ?

5.2 Le S.A.D. que vous utilisez améliore-t-il la qualité (précision, présentation, abondance) de l'information dont vous devez disposer avant toute prise de décision ?

6 - DEVELOPPEMENT D'APPLICATIONS

6.1 Veuillez indiquer quelles applications sont développées par les informaticiens à l'aide du L4G. Quelle est la durée de développement (en nombre de jours/homme) de chacune des applications ? Quelle en aurait été la durée probable si elles avaient été développées en COBOL ?

Type d'application	durée en nbr jours/hô	
	L4G	COBOL
LOURDE		
MOYENNE		
LEGERE (1)		

(1) applications légères, urgentes et non planifiées

6.2 Si vous possédez un générateur d'applications, quels en sont les avantages et les limites ?

6.3 Travaillez-vous selon la technique du prototypage ?
Quels avantages y voyez-vous ?
Ce moyen de définition des spécifications est-il moins contraignant ?

6.4 Quel est le rôle de l'utilisateur au cours du cycle de vie d'une application développée par le service informatique ? Son rôle était-il différent avant l'introduction du L4G ? (remplir par une croix les colonnes ad hoc)

ROLE	coopération étroite		actif		consultation		nul	
CYCLE VIE	L4G	avt(*)	L4G	avt	L4G	avt	L4G	avt
Analyse des besoins								
Analyse fonction.								
Analyse organique								
Réalisation								
Tests								
Exploit./ Mainten.								

(*) avt = avant l'introduction du L4G

6.5 De manière générale, pensez-vous que votre L4G puisse se substituer entièrement aux langages de programmation classiques (COBOL, PL1, ...) ?
Sinon, quel est le pourcentage d'applications qui peuvent être développées au moyen de votre L4G ?

6.6 Autres informations.

7 - MAINTENANCE

7.1 On dit que les L4G permettent d'alléger la maintenance, notamment grâce à leur outil de documentation et à leur langage de haut niveau.
Qu'en pensez-vous ?

7.2 Qui assure la maintenance des applications ?

7.3 Si des problèmes de maintenance subsistent, à quoi les attribuez-vous ?

8 - PRODUCTIVITE

Point de vue des développeurs

8.1 Pouvez-vous quantifier (éventuellement par classe d'applications) l'accroissement de productivité résultant de l'utilisation du L4G ?

Point de vue utilisateurs

8.2 Commment qualifiez-vous l'apport que vous procure un S.A.D. en termes de coût et de délai de réalisation de vos études ?

9 - AUTRES OBSERVATIONS

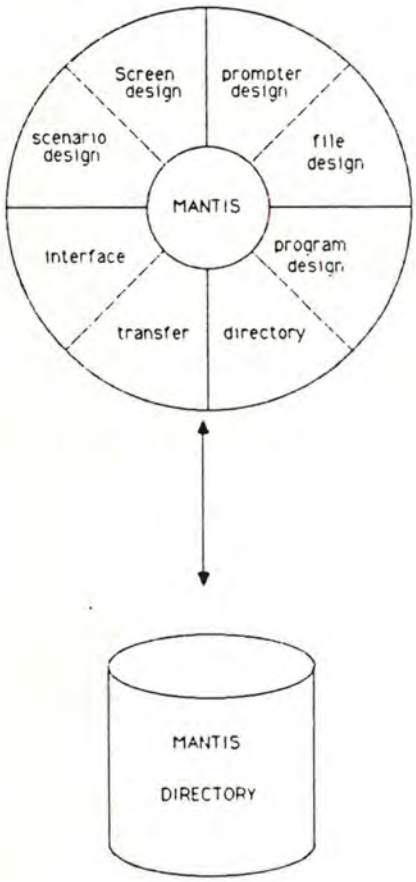
Avez-vous des commentaires à faire sur des points qui n'auraient pas été soulevés dans le questionnaire ?

ANNEXE B : Description de quelques langages de 4ème génération

M A N T I S

MANTIS est un produit de la firme CINCOM System. Ce produit aide à développer et à implémenter des applications. Il assure une meilleure productivité et est accessible en théorie à l'utilisateur final qui reçoit au préalable une formation.

MANTIS met à la disposition de l'utilisateur un ensemble d'outils standardisés au point de vue de l'interfaçage (voir graphique ci-dessous).



1. Développer une application à l'aide de MANTIS.

MANTIS permet d'établir un prototype c'est-à-dire un ensemble d'étapes où les changements seront pris en charge

facilement et à peu de frais. Pour développer ce prototype, l'utilisateur dispose de quatre outils :

Screen Design Facility permet de décrire des écrans à l'aide de deux types de zone : les zones destinées aux commentaires et les zones destinées à l'introduction des données pour lesquelles il est nécessaire de donner un nom et un type.

Prompter Design Facility : cet outil permet de créer une documentation sous forme de texte libre qui peut être visualisée dans une application à tout moment.

File Design Facilities : permet de décrire les fichiers. Chaque champ du fichier est spécifié par son nom, son type, sa dimension et ses attributs (ex. : clé d'accès). On peut aisément ajouter un champ à un fichier. Le système prend en charge d'une manière efficace cette opération. Les opérations sur les fichiers sont assurées au moyen de quatre primitives : GET, INSERT, UPDATE et DELETE. L'outil contrôle aussi l'accès au fichier (en lecture uniquement, lecture et mise à jour, ...).

Scenario Design Facility : cet outil aide l'utilisateur à créer l'enchaînement des écrans entrée / sortie, documentation, définis aux étapes précédentes.

On a donc créé un prototype. L'étape suivante consite à développer le programme de l'application. Pour réaliser cela, MANTIS dispose de deux outils :

Program Design Facility : les instructions qui composent un programme MANTIS sont de plusieurs types : instructions de contrôle, entrées-sorties, assignation. Il existe deux modes de fonctionnement :
- le mode "immédiat" : dans lequel, on introduit une instruction qui est exécutée immédiatement.

exemple : SHOW 15 + 5 return
20

- le mode "exécution" : dans lequel, on introduit une instruction ou un ensemble d'instructions qui seront executée ultérieurement.

exemple : 10 SHOW 15 + 5
20 ...

Les instructions de MANTIS sont donc interpretées.

Exemple de programme page suivante.

```
100  ENTRY CUSTOMER_ENTRY
110  SCREEN INPUT (CUSTOMER_ENTRY) * renseigne l'écran *
120    FILE  RECORD  (CUSTOMET_MASTER. ALL) * lien avec le
                                              fichier *
150  CONVERSE INPUT * affiche l'écran *
160  WHILE INPUT < > CANCEL * structure itérative *
190  INSERT RECORD * mise à jour du fichier *
200     MESSAGE_LINE   =   CUSTOMER   +   CUSTOMER_NAME   +
SUCCESSUFULLY INSERTED * affichage d'un message *
210  CONVERSE INPUT
220  END
230  STOP
240  EXIT
```

Une zone spéciale (work area) contient le programme en cours d'élaboration. Il est possible d'effectuer un certain nombre d'opérations à propos de ce programme (RUN, LIST, NEW, QUIT, ...)

Interface Design Facility : c'est un outil qui permet d'accéder à partir d'un programme écrit en MANTIS à du code COBOL, FORTRAN ou ASSEMBLER. Il s'agit tout d'abord de déterminer l'interface avec le programme hôte c'est-à-dire les paramètres du programme appelé. Ensuite, l'appel se réalise au moyen de l'instuction CALL.

2. Implémenter l'application.

Il s'agit d'introduire tout le travail effectué aux étapes précédentes dans un environnement de "production". MANTIS offre un système de répertoires (DIRECTORIES) dans lequel l'utilisateur enregistre toutes les entités de l'application (écran, documentation, fichier, ...) L'identification de l'utilisateur et un mot de passe sont les moyens mis en oeuvre pour gérer l'accès aux différents répertoires. MANTIS gère aussi le transfert de l'environnement prototype vers l'environnemnt production. Cet outil de "transfert" gère l'archivage des entités et les différentes versions de prototype. Arrivé dans l'environnement production, il est possible d'exécuter l'application en batch ou d'une manière "on-line". Aucun changement n'est nécessaire pour une exploitation en batch.

Soulignons enfin un outil qui permet de gérer les utilisateurs du système. Il existe trois types d'utilisateur :

Master User.
Cet utilisateur a tous les droits. Il détermine les droits, les priorités des autres utilisateurs. Il garde ainsi le contrôle du système.

Development User.
C'est l'utilisateur qui développe l'application. Il doit accéder à un ensemble d'outils qui l'aide à construire l'application.

Application User.
Il s'agit de l'utilisateur final qui exécute l'application. Le Master User veillera à accorder les privilèges adéquats à l'utilisateur final.

MANTIS est disponible sous plusieurs environnements.

IBM	TIS/XA-CM
	CICS
	IMS/DC
	VM/CMS
	TSO
DEC	VMS
ICL	TPMS
	TPS

etc...

De plus, MANTIS support plusieurs méthodes d'accès aux données.

IBM	SUPRA RDM
	TOTAL
	VSAM
	SQL/DS
	DB2
	DL/1
DEC	ULTRA RDM
	RMS
WANG	TOTAL
	DMS
HONEYWELL /BULL	TOTAL
	UFAS

En conclusion MANTIS est un outil de 4ème génération qui est implémenté dans 2000 entreprises réparties dans le monde. Celles-ci ont estimé une amélioration de la productivité à 10:1.

MANTIS offre une grande souplesse en ce qui concerne la reprise des données anciennes et il est exploitable dans plusieurs environnements.

En ce qui concerne les performances, une étude pratique effectuée dans un environnement déterminé s'avère souhaitable - ceci afin de mesurer l'impact du mode interpreté du langage de programmation.

Q B E

Le langage Query By Example (QBE) a été développé par IBM pour utiliser les systèmes relationnels. Ce langage a été défini par Zloof en 1975.

L'interaction entre l'homme et la machine se trouve bien étudié dans le langage QBE.

L'utilisateur de QBE travaille à partir d'un terminal à écran. La relation apparaît à l'écran sous forme d'un tableau :

nom des constituants

n-uplet

La zone de n-uplet située sur la zone des noms des constituants a une double fonction : d'une part, elle permet à l'utilisateur de définir sa requête à l'aide de conventions et d'exemples (ceci explique l'origine du nom du langage) d'autre part une fois la requête formulée elle reçoit la réponse dans les colonnes appropriées.

QBE et l'interrogation des données.

Supposons la relation

EMPLOYE (No-empl,Nom,Salaire,Département)

Si (num,n,sal,dép) EMPLOYE, alors la personne (n) de numéro (num) est employé dans le département (dép) pour un salaire (sal) fixé.

A partir de cette relation, l'utilisateur veut connaître tous les employés du département comptabilité. Avec le langage QBE, cette relation et la requête se présentent :

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
		P.(1)		Compta(2)

(1) pour PRINT
(2) pour comptabilité

On peut sélectionner des n-uplets sur base d'une condition autre que l'égalité. Il s'agit des opérateurs <, =<, >, >=.

L'utilisateur peut introduire une valeur qui n'est pas réelle en soulignant cette valeur. L'utilisateur décrit alors un exemple de résultat.

Ces deux notions apparaissent dans l'exemple suivant. On désire connaître le noms des employés qui ont un salaire supérieur à Dupont.

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
		Dupont P.(1)	x >x	

(1) pour PRINT

On désire connaître le nom des employés qui gagnent entre 35000 et 45000 frs.

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
		P.(1) x x	> 35000 < 45000	

(1) pour PRINT

On désire connaître le nom des employés qui travaillent dans le département comptabilité ou dans le département marketing.

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
		P P.(1)		Marketing Compta(2)

(1) pour PRINT
(2) pour comptabilité

Ces deux derniers exemples illustrent le ET et le OU.

Etant donné que tout langage relationnel offre la possibilité de définir des ensembles d'informations à l'aide d'expressions d'une grande puissance, on désire quelquefois appliquer un opérateur qui fournit une valeur associée à cet ensemble.

QBE offre les opérateurs suivants :

COUNT	pour le dénombrement des éléments
SUM	pour le calcul de la somme des éléments

AVERAGE pour le calcul de la moyenne
MAX pour la recherche du maximum
MIN pour la recherche du minimun
UNIQUE pour éliminer les valeurs dupliquées dans la réponse à une question

On désire connaître la moyenne des salaires des employés qui travaillent dans le département comptabilité.

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
			P. AVG	Compta(1)

(1) pour comptabilité

QBE et les opérations d'insertion, suppression et mise à jour.

Insertion.

On désire insérer l'employé Durant récemment engagé dans le département Recherche et Développement (R et D). Il reçoit le numéro 144 et un salaire de 45000 frs. L'insertion s'exprime sous la forme :

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
INSERT	144	DURANT	45000	R et D

Suppression.

L'employé Dupont qui porte le numéro 123 quitte l'entreprise. La suppression s'effectue comme suit :

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
DELETE	123			

On suppose que le numéro de l'employé est identifiant.

Mise à jour.

Le conseil d'administration décide de supprimer le département informatique et d'incorporer le personnel au département comptabilité. Cette mise à jour se réalise de la manière suivante :

EMPLOYE	No-empl	Nom	Salaire	Départe-ment
UPDATE	x x			Info(1) Compta(2)

(1) pour informatique
(2) pour comptabilité

Conclusions.

Le langage QBE a été particulièrement conçu pour des utilisateurs ayant peu de connaissance en informatique. En effet, la requête peut être spécifiée à l'aide d'un

exemple. L'ecriture de la requête s'effectue dans un squelette d'une relation apparaissant à l'écran.

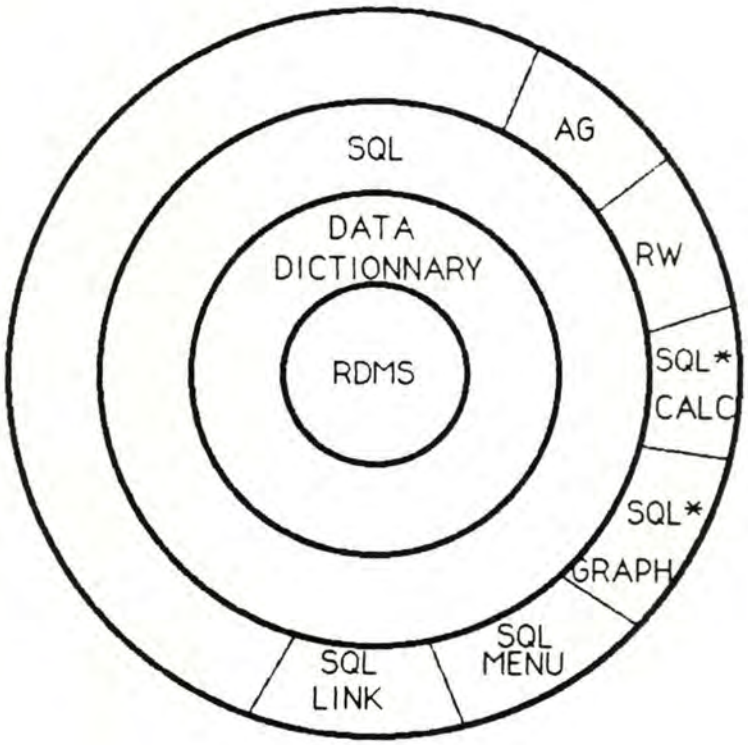
Une étude psychologique qui étudie le comportement de sujets qui utilisent QBE a été réalisée par Thomas et Gould. JC Thomas and JD Gould, "A Psychological Study of Query-by-example", Proceeding National Computer Conf, Vol 44, page 439 - 445, 1975

O R A C L E

La rapide description du logiciel ORACLE présentée ci-après est basée sur les brochures et documents qui nous ont été fournis par le fabricant. Dès lors, les éventuelles faiblesses du logiciel n'apparaîtront pas de manière explicite !...

ORACLE (d'ORACLE Corporation) est un SGBD relationnel compatible avec SQL/DS et DB2 d'IBM. Il comprend un langage SQL complet, ainsi qu'un ensemble d'outils logiciels intégrés pour le développement d'applications, l'édition de rapports, le dessin de graphiques et la gestion des réseaux. ORACLE est disponible sur un très grand nombre de grosses machines (IBM 370, 4300, 30XX), de mini- (AX, PDP, HP 9000, Bull DPS-6,...) et de micro-ordinateurs (compatibles XT/AT, Rainbow,...).

Les principaux modules d'ORACLE (version PC) peuvent être représentés comme suit :



avec : RDMS = "Relational Database Management System";

SQL = le langage d'interrogation qui constitue l'interface du RDMS;

AG = "Application Generator", qui permet à des utilisateurs d'écrire leurs propres programmes d'application à l'aide d'un dialogue interactif simple;

RW = "Report Writer", grâce auquel le résultat de toute requête peut automatiquement être sorti sous forme de rapport;

SQL-calc = un tableur permettant d'exploiter directement des données de la BD centrale;

SQL-graph = un module permettant le dessin de graphiques (couleur);

SQL-menu = un module permettant à l'utilisateur de créer et d'exécuter ses propres menus;

SQL-link = un module permettant le transfert de données entre systèmes ORACLE;

Nous allons nous attacher à décrire un peu plus dans le détail le langage SQL d'ORACLE, étant donné que les autres modules, eu égard à leur caractère essentiellement interactif, sont difficiles à présenter sur papier.

Le langage SQL d'ORACLE

- CREATE TABLE, la commande de création d'une table.

```
ex : CREATE TABLE EMPLOYE  (NOEMP     NUMBER(4) NOT NULL,
                            NOMEMP    CHAR(10),
                            FONCTION  CHAR(9),
                            DATEMB    DATE,
                            SALAIRE   NUMBER(7,2),
                            NODEP     NUMBER(2));
```

Par rapport à SQL/DS, le SQL d'ORACLE présente les particularités suivantes :

- l'existence du type "DATE";

- définition "externe" des données numériques, et non interne (INTEGER, FLOATING POINT, DECIMAL,...);

- récupération de la place mémoire non occupée (compression des données);

- ...

- SELECT, la commande d'extraction de données.

structure générale :

SELECT, suivi du nom des colonnes qui constituent chaque ligne du résultat
FROM, suivi des tables desquelles le résultat tire ses valeurs
WHERE, suivi de la condition de sélection que doivent satisfaire les lignes qui fournissent le résultat

ex :
```
SELECT NOMPEMP, SALAIRE
FROM EMPLOYE
WHERE FONCTION = 'Comptable'
AND SALAIRE BETWEEN 60000 AND 80000
ORDER BY SALAIRE DESC
```

Cette requête fournit la liste, ordonnée suivant le niveau des salaires (en commençant par le plus élevé), de tous les comptables et de leur salaire respectif qui gagnent entre 60000 et 80000 francs par mois.

Cet exemple n'est qu'une petite illustration des nombreuses possibilités d'extraction offertes par SQL; il est aussi possible, par exemples, de supprimer du résultat les lignes identiques (SELECT DISTINCT), de faire des requêtes portant sur plusieurs tables, de faire des sous-requêtes, de présenter sous forme de rapport les résultats d'une requête (avec en-têtes, clôtures, sous-totaux : commandes COLUMN, BREAK, COMPUTE, TTITLE, BTITILE).

En outre, ORACLE offre les opérateurs arithmétiques classiques (+, *, -, /) mais aussi, à la différence de SQL/DS, les fonctions d'arrondi , de troncature, de valeur absolue et d'exponentiation. ORACLE dispose également de la plupart des fonctions portant sur les chaînes de caractères (concaténation, décodage/encodage, longueur,...) et notamment la fonction **SOUNDEX** qui permet de retrouver tous les noms ayant la même phonétique, mais qui s'orthographient différemment.

Ajoutons enfin, sans vouloir être exhaustif, que de nombreuses mises en page et opérations arithmétiques sont possibles sur les dates.

- UPDATE, la commande de mise à jour des données.

```
ex :    UPDATE EMPLOYE
        SET SALAIRE = SALAIRE + 1000
        WHERE FONCTION = 'Comptable'
```

- INSERT, la commande d'ajout de lignes.

```
ex :    INSERT INTO EMPLOYE
        VALUES (30,'Dubois','Comptable')
```

- DELETE, la commande de suppression de lignes.

```
ex :    DELETE FROM EMPLOYE
        WHERE NOEMP = 30
```

- Changements dynamiques de la description de la BD.

- ALTER TABLE ADD, la commande d'ajout d'une nouvelle colone à une table;

- ALTER TABLE MODIFY, la commande d'"élargissement" d'une colonne.

```
ex : ALTER TABLE EMPLOYE
     ADD (SEXE CHAR(1));

     ALTER TABLE EMPLOYE
     MODIFY FONCTION CHAR(12);
```

Remarquons qu'ORACLE ne permet pas de supprimer une colonne ni de réduire la "largeur" d'une colonne existante.

- CREATE VIEW, la commande de définition de vues.

```
ex : CREATE VIEW EMP10
     AS SELECT NOEMP, NOMEMP, FONCTION
     FROM EMPLOYE
     WHERE NODEP = 10;
```

La commande **DROP** permet de supprimer une vue.

- GRANT, la commande permettant d'accorder des privilèges à des utilisateurs.

```
ex : GRANT SELECT
     ON EMPLOYE
     TO PUBLIC;
```

Le principe est que seul l'utilisateur qui a créé une table ou une vue peut accorder à d'autres certains privilèges sur cette table ou vue.

La commande **REVOKE** permet de supprimer un privilège.

- CREATE INDEX, la commande de création d'index.

Sur chaque table réelle peuvent être déclaré un nombre quelconque d'index, définis sur une ou plusieurs colonnes, et permettant d'accélérer l'accès et le tri des données.

ex : **CREATE INDEX** I-NODEP **ON** EMPLOYE(NODEP);

La définition d'un index peut être l'occasion de déclarer uniques les colonnes constituant cet index (CREATE UNIQUE INDEX).

- CREATE CLUSTER, la commande de définition de "clusters"

Le "clustering" est une technique destinée à améliorer les performances en cas de jointures de tables : des lignes de tables différentes sont stockées sur la même page physique.

ex : Supposons que l'on crée la table DEPARTEMENT

```
CREATE TABLE DEPARTEMENT   (NODEP NUMBER(2),
                            NOMDEP CHAR(5));
```

Définissons le "cluster" DEP-EMP (tables EMPLOYE et DEPARTEMENT) sur la colonne NODEP commune aux deux tables

```
CREATE CLUSTER DEP-EMP (NODEP NUMBER);
```

Ajoutons au "cluster" les tables EMPLOYE et DEPARTEMENT

```
ALTER CLUSTER DEP-EMP
ADD TABLE EMPLOYE
WHERE EMPLOYE.NODEP = DEP-EMP.NODEP;

ALTER CLUSTER DEP-EMP
ADD TABLE DEPARTEMENT
WHERE DEPARTEMENT.NODEP = DEP-EMP.NODEP;
```

Ainsi, le "clustering" optimise les requêtes nécessitant une jointure et de plus élimine physiquement certaines redondances (de NODEP dans notre cas).

La définition et la suppression d'index et de "clusters" se fait de manière tout à fait dynamique et transparente pour les utilisateurs.

Interfaces avec des programmes d'application.

Toute commande SQL peut être transmise au SGBD par un programme d'application (écrit en COBOL, PL/1, FORTRAN, C, PASCAL,...) en vue de son exécution immédiate. Dès lors, en matière d'extraction de données, le programme doit pouvoir traiter chaque ligne du résultat en séquence. D'autre part, il doit pouvoir passer des valeurs en arguments, et recevoir des valeurs résultats dans des variables qui lui sont propres.

Nous allons examiner au travers d'un exemple comment ORACLE prend en charge ces différentes contraintes. L'exemple qui suit est écrit en PL/1 : pour un numéro de département donné, le programme affiche, un par un le nom ainsi que le salaire de chaque employé de ce département. Après l'affichage de chaque employé et salaire, une augmentation de dix pour cent du salaire peut être accordée.

```
     PRGM:   PROC OPTIONS(MAIN);
                  DCL AA CHAR(1);
(1)               EXEC SQL BEGIN DECLARE SECTION;
                  DCL UID CHAR(20);
                  DCL PWD CHAR(20);
                  DCL XX  CHAR(40);
                  DCL YY  FIELD BIN(31);
                  DCL ZZ  FIELD BIN(15);
             EXEC SQL END DECLARE SECTION;
(2)          EXEC SQL INCLUDE SQLCA;
             DISPLAY("Entrer user id")       REPLY(UID);
             DISPLAY("Entrer mot de passe") REPLY(PWD);
(3)          EXEC SQL CONNECT :UID IDENTIFIED BY :PWD;
(4)          EXEC SQL DECLARE C1 CURSOR FOR
                  SELECT NOMEMP, SALAIRE
                  FROM EMPLOYE
                  WHERE NODEP = :ZZ
                  FOR UPDATE;
     NEXT:   DISPLAY("N° de département = ?")   REPLY(ZZ)
                  IF ZZ = 0 GO TO STOP;
(5)          EXEC SQL OPEN C1;
             DO   WHILE (SQLCODE = 0);
(6)               EXEC SQL FETCH C1 INTO :XX, :YY;
                  DISPLAY(XX, YY);
                  DISPLAY("10 % augment. (Y/N) ?")
                                              REPLY(AA);
                  IF AA NOT= 'Y' GO TO NOUP;
(7)               EXEC SQL  UPDATE EMPLOYE
                            SET SALAIRE = 1.1 * SALAIRE
                            WHERE CURRENT OF C1;
     NOUP:
```

```
(8)                EXEC SQL COMMIT WORK;
                   GO TO NEXT;
              END;
(9)           EXEC SQL CLOSE C1;
              GO   TO NEXT;
      STOP:
        END PRGM;
```

---

(1) variables partagées : les variables qui doivent être partagées entre le programme et ORACLE sont déclarées normalement, mais précédées de EXEC SQL BEGIN DECLARE SECTION et clôturées par EXEC SQL END DECLARE SECTION;

(2) à chaque exécution d'une commande SQL, ORACLE retourne un code programme, indiquant le succès ou l'échec de cette commande. Ce code de retour est appelé le SQLCA;
(3) après que l'utilisateur ait introduit son "user id" et son mot de passe (respectivement dans UID et PWD), il s'agit d'établir la connexion avec ORACLE; celle-ci ne sera effective qu'après qu'ORACLE ait vérifié dans le dictionnaire de la BD que les "user id" et mot de passe sont valides; de plus, ORACLE déterminera quelles sont les données auxquelles l'utilisateur pourra accéder;

(4) étant donné qu'un département comprend plusieurs employés, le résultat de la commande SELECT sera un **ensemble** de lignes. Dès lors, un **curseur** sera utilisé pour traiter une ligne à la fois. La commande **DECLARE** associe le curseur C1 à la commande SELECT qui suit. La clause **FOR UPDATE** indique à ORACLE que le programme pourra faire des mises à jour.

(5) OPEN provoque l'évaluation de la requête et le positionnement du curseur sur la première ligne résultat, si elle existe. A ce stade, cependant, aucune ligne résultat n'est transmise au programme;

(6) les lignes résultat sont fournies au programme une à une, par la commande **FETCH** qui garnit les variables XX et YY avec respectivement le nom et le salaire de la ligne courante. De plus, étant donné qu'il s'agit ici d'une mise à jour, la ligne (et non la table) est verrouillée durant tout le temps du FETCH. Remarquons que la boucle "WHILE" se poursuit tant que l'indicateur de diagnostic (SQLCODE) est à 0 (l'opération s'est déroulée normalement);

(7) le programme peut utiliser un UPDATE pour mettre à jour la ligne courante;

(8) une ligne restera verrouillée jusqu'à l'exécution d'une commande **COMMIT WORK**. Si la ligne a été mise à jour, le COMMIT WORK aura pour effet de rendre effective la mise à jour dans la BD. Si le programme se termine anormalement, ORACLE remet la BD dans son état depuis le dernier COMMIT WORK.

(9) La commande CLOSE ferme le curseur; toute ouverture ultérieure provoquera à nouveau l'évaluation de la requête.

D'autres caractéristiques n'apparaissent pas dans l'exemple :

- plusieurs curseurs peuvent être ouverts en même temps dans un même programme;

- contrôle de la cohérence des données lors de toute requête;

- transaction : Logical Unit of Work;

- COMMIT automatique si fin normale de programme;

- ROLLBACK automatique en cas d'incident;

- verrouillage implicite/explicite; modes partagé/exclusif; granularité = table/ligne;

- prise en charge des interblocages (ORACLE tue la plus jeune transaction);

- traitement d'erreur automatique ou par programme (examen du contenu de SQLCA);

- etc...