MASTER EN SCIENCES INFORMATIQUES

De la conception au développement d'un système d'information: cas d'un gestionnaire du parc informatique

Mifuku, Assin-Na-Idia

Award date:
2000

FUNDP
INSTITUT D'INFORMATIQUE

De la conception au développement d'un système d'information

Cas d'un gestionnaire du parc informatique.

Mifuku Assin-Na-Idia

RUE GRANDGAGNAGE, 21 • B – 5000 NAMUR (BELGIUM)

Table des matières[1]

[1] La Table des Matières plus détaillée de chaque chapitre se trouve en son début.
[2] SLLParc est le nom de l'application qui gère le parc informatique de Swiss Life Luxembourg.

Remerciements

Ce mémoire n'aurait pu voir le jour sans l'aide, ni le soutien, ni le support d'un réseau des personnes qu'il convient de remercier de tout cœur.

Tout d'abord, je remercie Monsieur Jean Ramaekers d'avoir bien voulu diriger ce mémoire, par tous ses encouragements, conseils et son déplacement jusqu'au lieu de stage. Aussi, je dis merci à Monsieur Radu Cotet, gestionnaire du parc informatique de notre Institut pour son intérêt à ce travail. Merci à Monsieur Benno Leguina qui a proposé ce sujet pour le besoin de son Service Informatique à SwissLife Luxembourg et qui a bien dirigé notre stage dans son service.

En terminant ce cycle d'études, je suis reconnaissant à tous mes professeurs qui m'encourageaient régulièrement durant mes études ; Madame Claire Lobet-Maris en particulier. Je garde une attention particulière à tous mes amis et collègues de promotion. Je ne peux pas oublier les collègues de travail à Swiss Life Luxembourg où j'ai eu de bons contacts pendant mon stage.

Que la famille Minnoye Robert trouve en ce travail ma profonde gratitude pour l'action inoubliable en mon égard. A travers elle, toutes les personnes, que je saurai citer ici, qui de près ou de loin ont contribué à me pousser jusqu'à ce niveau. Merci.

Que Ya Jean Mifuku soit remercié pour toute son action et sa vision en ma personne, qui ce jour peut prétendre lui faire valoir un honneur particulier. Merci Ya Jean. Autour de lui, je n'oublie pas mes sœurs auxquelles je suis très attaché et que j'aime beaucoup ; Ya Alphonse Bay, Dorth Mbombo, Thierry, Ya Richard, Charles Gaïse. Merci.

A toi ma chère épouse Clarisse M., ce travail est le fruit de ta persévérance, de ton aide, de ton encadrement, de tes prières et de ta patience. Une seule conviction reste gravée dans mon cœur « *je t'aime* ».

Ce mémoire est dédié à ma fille Allégresse Mifuku, qui est ma joie accomplie venue de Dieu.

A Dieu soit la gloire pour sa fidélité en ses promesses à notre endroit en Jésus-Christ son Fils Unique.

Résumé

L'informatisation d'une entreprise passe par celle des tâches régulières et habituelles d'un poste de travail. Concevoir et développer une application informatique est une branche informatique qui a des règles et une méthodologie.

L'objectif de ce travail est justement d'exposer la théorie et la pratique dans ce domaine.

En effet, partant de la théorie du développement d'une application, nous présenterons un cas pratique d'une application de gestion que nous avons développée en entreprise.

De la gestion manuelle d'un parc informatique d'entreprise par des formulaires, nous avons réalisé un outil informatique de cette gestion.

Ce mémoire, nous le souhaitons comme un document de bord qu'utiliserait tout développeur d'application informatique.

Abstract

Design with the development of an information system.
Case of a manager of the data-processing park.

The computerisation of a company occurs through the regular and usual tasks of work station. The conception and developement of a computer application is a sector of computer science which has rules and methodology.

The objective of this report is precisely to present the theory and the practice in this area.

In fact, starting from the theory of the development of an application, we will present a case-study of a management application which we have developped within the company.

From the management of a company computer pool using forms, we have pruduced a computerised tool of this management.

This report, we would like it to be as a log-book for the use of all computer application developers.

INTRODUCTION

Ce mémoire présente l'essentiel de l'informatique de gestion. L'analyse, la conception et la mise en œuvre d'un système d'information pour un secteur d'activité donné. L'entreprise se trouve englobée dans un environnement qui nécessite un effort d'adaptation qui n'avait jamais été aussi fondamental qu'à notre époque.
D'ailleurs, la multiplicité des fusions, absorptions et disparitions qui se réalisent de nos jours dans tous les secteurs est bien un symptôme de cette perturbation de l'univers économique. L'entreprise doit donc s'adapter pour survivre et se développer. A la nécessité d'optimiser la productivité, s'est ajoutée la nécessité d'optimiser les méthodes de gestion et de direction de l'entreprise. Ainsi une véritable politique de gestion doit être établie, d'une part face à l'environnement et d'autre part, sur le plan interne.

Sur le plan interne, l'entreprise s'adapte en automatisant ses activités. Les tâches formalisables sont automatisées au fur et à mesure, et l'entreprise s'informatise.

Ce mémoire s'adresse justement à ceux qui informatisent l'entreprise.

Dans ce travail, nous essayons de donner un certain nombre d'étapes et de notions pour concevoir et mettre en œuvre un outil informatique de gestion.
Plusieurs approches peuvent être considérées dans ce domaine.

Notre approche est fondée à partir d'un logiciel sur mesure que nous avons eu l'opportunité de développer en entreprise lors de notre stage de fin d'études.
Les notions théoriques que nous présentons dans ce travail sont utiles pour pouvoir mener à bien un développement d'une application.

Contenu du mémoire :

Dans ce travail se profilent deux aspects : l'aspect théorique et l'aspect pratique. Les chapitres théoriques donnent des notions essentielles d'un système d'information alors que le chapitre pratique décrit la construction du logiciel développé, qui gère un parc informatique. Nous ajouterons un dernier chapitre qui tente de jeter le pont entre ces deux aspects.

Ce travail est divisé en trois parties suivies des annexes.

La première partie s'intitule : les considérations théoriques de conception et de développement de systèmes d'information. Il s'agit de rappeler l'essentiel de la théorie pour un système d'information et son développement. Comme nous l'avons dit, il ne s'agit pas d'étudier un système d'information ni le développement de logiciels mais d'en rappeler les notions utiles et incontournables. Cette partie sera composée de quatre chapitres.

Le premier chapitre est la théorie sur l'analyse fonctionnelle. Nous y rappelons quelques notions sur la conception des systèmes d'information.

Après cela, nous rappellerons la théorie sur les bases de données au chapitre deuxième. Connaissant l'immensité de ce sujet, nous nous limiterons aux grandes lignes de cette branche pour finalement parler des bases de données relationnelles et du langage SQL.

Au chapitre troisième, nous aborderons la théorie sur les règles qui président à la conception d'une interface utilisateur.

Le chapitre quatrième se consacrera à la théorie générale sur le développement d'un logiciel. Comment développe-t-on un logiciel ? Quel est son cycle de vie ? Nous répondrons entre autre à ces interrogations.

Ensuite, la deuxième partie avec son seul chapitre qui est le cinquième du travail, expliquera comment nous avons pratiquement conçu et réalisé cet outil de gestion du parc informatique. Il gère le parc informatique de Swiss Life à Luxembourg. C'est une étude d'un cas précis.

Enfin, la troisième partie de ce travail est intitulée : de la théorie à la pratique et vice versa. Elle aura aussi un chapitre (le sixième) qui tentera de jeter le pont entre la théorie et la pratique réalisée.

Après la conclusion, nous annexerons quelques documents, entre autre ceux de la gestion non programmée du parc informatique de Swiss Life à Luxembourg, quelques lignes de code du programme écrit et la maquette de l'application.

Ce mémoire ne dispense pas au lecteur de consulter les ouvrages spécialisés sur les sujets évoqués, au contraire. Ce mémoire veut préparer le lecteur à cette consultation parfois difficile, mais ne prétend nullement se substituer à eux.

Partie I

LES CONSIDERATIONS THEORIQUES SUR LA CONCEPTION ET LE DEVELOPPEMENT DE SYSTEMES D'INFORMATION

Dans cette partie, nous essayons de rappeler les concepts essentiels des systèmes d'information et leur développement.

Elle est composée de quatre chapitres :

Chapitre 1 : Théorie sur l'analyse fonctionnelle.
Chapitre 2 : Théorie sur les bases de données.
Chapitre 3 : Théorie sur la conception des interfaces homme-machine.
Chapitre 4 : Théorie sur la méthodologie de développement du logiciel.

Chapitre 1 : Théorie sur l'analyse fonctionnelle.

Sommaire

[1] SLLParc est l'application ou le logiciel sur mesure que nous avons développé à Swiss Life Luxembourg pour la gestion du parc informatique dans le cadre du stage de fin d'études.

Chapitre 1 : Théorie sur l'analyse fonctionnelle[2].

I.1 Introduction

Les professionnels de l'informatique (les concepteurs, les analystes, les programmeurs, les développeurs,...), ceux qui pratiquent la micro-informatique individuelle, les utilisateurs, les décideurs, ... sont de plus en plus conscients de l'intérêt de procéder à "une bonne analyse et à une bonne conception" afin de réussir les informatisations dont ils sont responsables. Les avantages sont nombreux :

- systèmes d'information (et applications), et donc programmes, répondant effectivement aux attentes des utilisateurs,

- gains de temps importants car les solutions sont définies rationnellement et non par approximations successives ("en bricolant"),

- des économies financières importantes, etc.

Ces avantages sont constatés pour l'informatisation d'un système d'information ou d'une seule application ou pour la réalisation d'un seul programme.

La qualité des réalisations sérieusement analysées et conçues est appréciée lors de leur utilisation mais aussi lors de leur exploitation et lors de leur maintenance.

L'analyse conceptuelle, pour laquelle on utilise l'expression Ingénierie des Exigences actuellement, se veut de plus en plus méthodique aujourd'hui.

L'analyse conceptuelle comprend trois étapes :

- l'élaboration du schéma conceptuel des données,
- l'élaboration du schéma conceptuel des traitements,
- la consolidation des spécifications au niveau du projet.

[2] Nous adoptons la dénomination «analyse fonctionnelle » au lieu de «analyse conceptuelle » [Bod, 94].

I.2 Conception des systèmes d'information

I.2.1 Définition d'un système d'information

Qu'est-ce qu'un système d'information ?
[Cast, 87] propose la définition suivante.

Un système[3] d'information est un ensemble de procédé :
- qui reçoit et qui achemine des informations qui parviennent à un système,
- qui achemine et qui émet des informations produites par ce système,
- et qui éventuellement achemine, traite et/ou gère des informations internes à ce système.

Cette définition repose sur le concept d'information que nous différencions du concept de donnée.

Une information représente un fait ou une partie d'un fait alors qu'une donnée est une représentation codée des propriétés d'un fait ou d'un événement. L'information est une signification potentielle attachée aux données, susceptibles d'affecter le comportement des hommes et des machines dans une organisation [Bod, 94].

Exemples de système d'information :
- Un système de gestion d'une entreprise.
- Un système de gestion d'un parc informatique d'une entreprise.
- Un système de gestion familiale.
- Un système de gestion d'une chaîne de montage de voitures.

Qu'est-ce qu'une application informatique ?

Une application est un sous-système d'information qui traite de problèmes :
- qui ont une même finalité,
- qui sont fortement dépendants les uns des autres,
- et qui sont indépendants ou quasiment indépendants des autres problèmes traités par le système d'information.

I.2.2 Définition d'un système automatisé d'information (SAI)

Un SAI est un sous-système d'un système d'information dans lequel toutes les transformations significatives d'informations sont effectuées par des

[3] Parmi les nombreuses définitions du mot «système », nous retenons qu'un système est un ensemble d'éléments en interaction dynamique (généralement), organisés en fonction d'un but.

machines de traitement automatique des informations (ordinateurs).

II permet la conservation et le traitement automatique des informations.

Deux raisons justifient l'automatisation d'un système d'information.

— Simplification et amélioration du travail administratif (comptabilité, facturation, paye, etc.) par l'automatisation des procédures répétitives et fastidieuses de simple exécution,

— Aide à la décision : Si la décision appartient à l'Homme et non à l'ordinateur, celui-ci peut fournir à celui-là des éléments qui lui permettront de faire ses choix en disposant du maximum d'informations possible ; l'ordinateur pouvant sélectionner à grande vitesse parmi sa grande masse de données mémorisées les informations utiles à la prise de décision (aide au pilotage).

Exemple : Statistiques de ventes

L'ordinateur peut d'ailleurs servir d'outil de simulation (avec des logiciels tels que les tableurs) et permettre au gestionnaire de mesurer très rapidement quelles seraient les conséquences de tel ou tel choix afin de trouver de proche en proche les meilleures décisions possibles.

Pour qu'un sous-système du système d'information soit automatisable, il faut qu'il soit formalisable, c'est-à-dire que la connaissance des entrées détermine les sorties. Cela revient à dire qu'il ne doit comporter que des actions programmées éventuellement en transformant des choix en actions programmées au moyen d'un modèle.

I.2.3 Les niveaux d'abstraction d'un système d'information

Lors de la conception d'un système d'information, on sera amené à considérer trois niveaux d'études [fig. 1.1] :

- Le niveau conceptuel,
- Le niveau organisationnel,
- Le niveau opérationnel.

Le niveau conceptuel consiste à penser le système d'information sans envisager aucun concept lié à l'organisation, tant du point de vue des données que de celui des traitements. Il consiste à se poser la question QUOI ? (c'est-à-dire QUOI FAIRE ? et AVEC QUELLES DONNEES ?).

On ne préjuge en aucune manière du matériel utilisé ni de l'organisation du travail.

Le niveau organisationnel consiste à intégrer à l'analyse les critères liés à l'organisation (notions de lieux, de temps d'acteurs et donc de postes de travail). Du point de vue des traitements, on se pose les questions QUI ? OU ?

QUAND ? et on envisage le partage des tâches entre l'Homme et la machine.

Du point de vue des données, on commence à étudier leur organisation, compte tenu du logiciel utilisé, mais sans s'occuper des méthodes de stockage et d accès, c'est-à-dire en gardant l'optique de l'utilisateur sans ignorer les contraintes du matériel et du logiciel (fichiers logiques).

Le niveau opérationnel consiste à apporter des solutions techniques au problème. II consiste à se poser la question COMMENT ?

Du point de vue des données, on effectue des choix sur les méthodes de stockage et d accès (fichiers physiques).

Pour les traitements automatiques on étudie le découpage en programmes.

D'une manière générale on envisage les contraintes d'utilisations des ressources physiques.

NIVEAU	TRAITEMENTS	DONNÉES	CHOIX
CONCEPTUEL	MODÈLE CONCEPTUEL	MODÈLE CONCEPTUEL	DE GESTION
ORGANISATIONNEL	MODÈLE ORGANISATIONNEL	MODÈLE LOGIQUE	D'ORGANISATION
OPERATIONNEL	MODÈLE OPÉRATIONNEL	MODÈLE PHYSIQUE	TECHNIQUES

Fig. 1.1 Niveaux d'abstraction d'un système d'information

I.2.4 Le modèle général d'un système d'information

Par système d'information d'une organisation, nous entendons une construction formée d'ensemble :

-d'**informations**, qui sont des représentations - partielles - de faits qui intéressent l'organisation ;
- de **traitements**, qui constituent des procédés d'acquisition, de mémorisation, de transformation, de recherche, de présentation et de communication des informations ;
- de **règles d'organisation** qui régissent l'exécution des traitements informationnels ;
- de **ressources humaines et techniques** requises pour le bon fonctionnement du système d'information.

[Bod, 94] nous propose le schéma ci-après du modèle de tout système d'information.

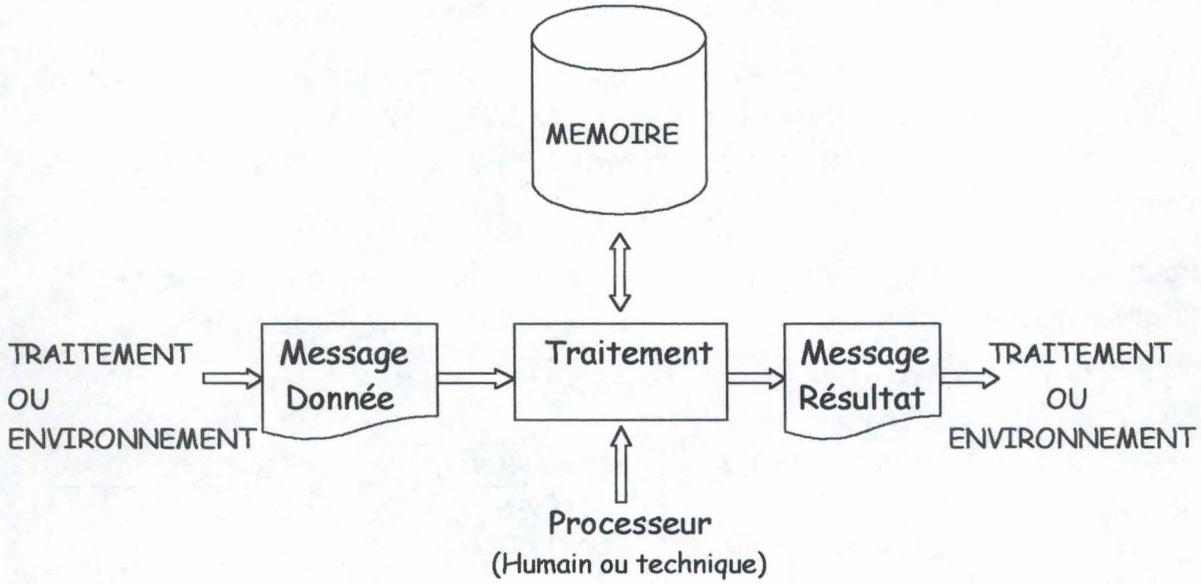


Fig. 1.2 Schéma du modèle d'un système d'information.

Ce schéma classique se lit ainsi : un message-donnée, en provenance d'un traitement ou de l'environnement du système d'information (S.I.), est communiqué au S.I. qui le traite via un processeur, éventuellement à l'aide de sa mémoire, pour engendrer un message-résultat lequel est à son tour transmis à un autre traitement ou à l'environnement du S.I.

I.2.5 Développement d'un système d'information

Le cycle de vie d'un système d'information

Le cycle de vie d'un système d'information complet peut-être décrit par les points ci-après :

- il commence lorsque l'organisation envisage ou décide de le construire,
- il passe par différentes étapes de développement préalables à sa mise en place,
- il connaît une période, plus ou moins longue, d'utilisation et de maintenance, et se termine lorsque l'organisation juge qu'il est devenu obsolète ou inadéquat.

En fonction de l'évolution de ses besoins, l'organisation est amenée à construire des versions successives d'un même S.I.

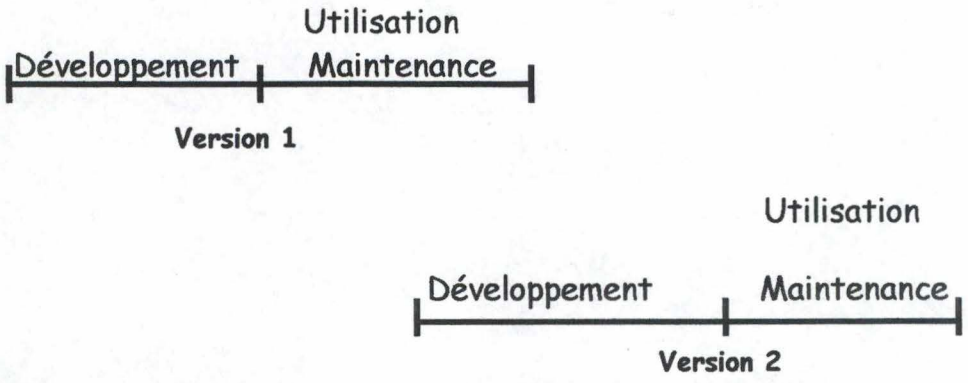


Fig. 1.3 Schéma du cycle de vie d'un système d'information.

Les étapes du développement d'un système d'information

Le développement d'un système d'information couvre différentes étapes :

- **l'analyse ou étude d'opportunité** qui prépare un avant-projet de solution à partir des besoins exprimés par l'organisation. Idéalement, l'étude d'opportunité devrait s'inscrire dans le cadre du schéma directeur des systèmes d'information de l'organisation.
- **l'analyse conceptuelle** qui, sur base de l'avant-projet, élabore une solution fonctionnelle détaillée mais indépendante de tout moyen de réalisation.
- la **conception technique** qui transforme la solution conceptuelle en une solution technique par la prise en considération logique des environnements matériels et logiciels.

La conception technique couvre, en toute généralité, la conception des fichiers et de la base de données, la conception des dialogues, la conception du système de communication et la conception de l'architecture des traitements.

- **la conception des procédures d'organisation** en vue de rendre effectif les changements dans les structures d'organisation et dans le comportement des personnes définis lors de l'étude d'opportunité. Ces procédures d'organisation porteront sur l'évolution des structures d'organisation, sur les définitions des postes de travail, sur les descriptions des fonctions, sur l'énoncé des standards et des procédures de comportement, etc.
- **la réalisation et la mise au point de la solution technique** en fonction des caractéristiques réelles des matériels et des logiciels. La mise au point couvre à la fois les tests de programmation et les tests de réception par les utilisateurs. On soulignera qu'avec le développement des générateurs, la réalisation sera de plus en plus automatisée.
- **la mise en place de nouvelles procédures d'organisation**. Il s'agit essentiellement de préparer la mise en place progressive de nouvelles procédures d'organisation. Elle inclut la formation des utilisateurs aux tâches et

aux rôles nouveaux qu'ils devront assumer.

- **l'exploitation** du système d'information.

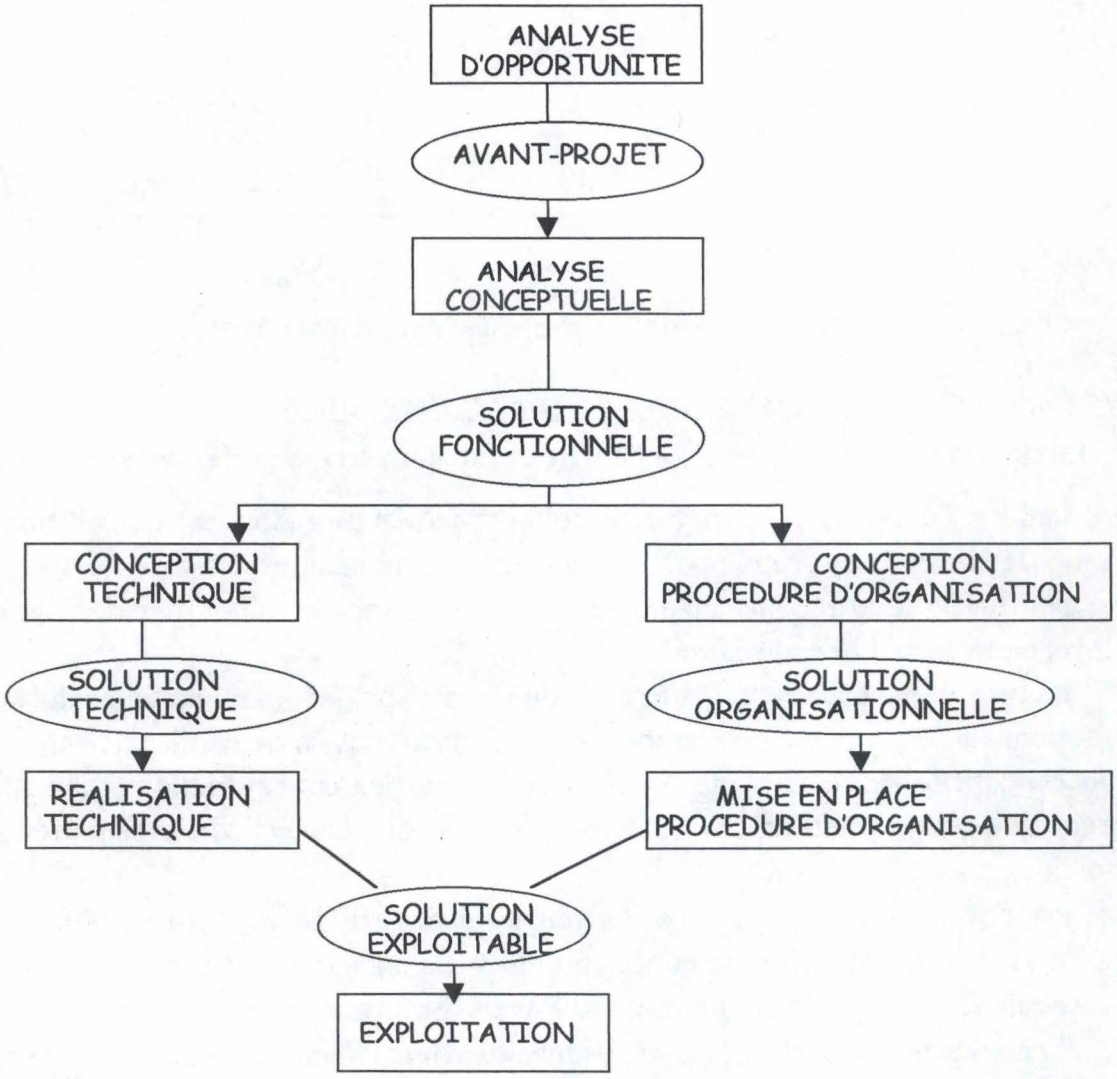


Fig. 1.4 Schéma des étapes du développement d'un système d'information.

Remarques sur ce schéma.

- II est important de noter que la linéarité dans le déroulement du cycle de vie d'un projet n'est qu'apparente : il existe des points de contrôle qui sont la source de boucles de rétroaction. L'existence des points de contrôle est liée, d'une part au fait que concevoir constitue un processus d'apprentissage et d'autre part, au fait que les particularités et les contraintes d'une étape introduisent fréquemment des effets rétroactifs sur les résultats des étapes précédentes.

- Nous considérons que l'analyse fonctionnelle couvre les étapes créatives du cycle de vie : l'étude d'opportunité et l'analyse conceptuelle.

Au départ de besoins exprimés par l'organisation, l'objectif de l'étude d'opportunité est de préparer un avant-projet de solution qui doit notamment servir de base - cahier des charges - à l'analyse conceptuelle pour construire une solution conceptuelle détaillée du système d'information à mettre en place.

I.2.6 Méthodologie de conception d'un système d'information

Afin de réduire les délais de réalisation des projets, les coûts de maintenance, les dépassements systématiques des plannings et des budgets de développement, un effort considérable a été fait au cours de la décennie passée dans la mise au point de méthode de développement des systèmes d'information. Il n'est plus discutable actuellement que toute méthode doit proposer une démarche fondée sur des modèles et mise en œuvre à l'aide d'outils logiciels. Le schéma suivant donne une méthode de conception d'un système d'information.

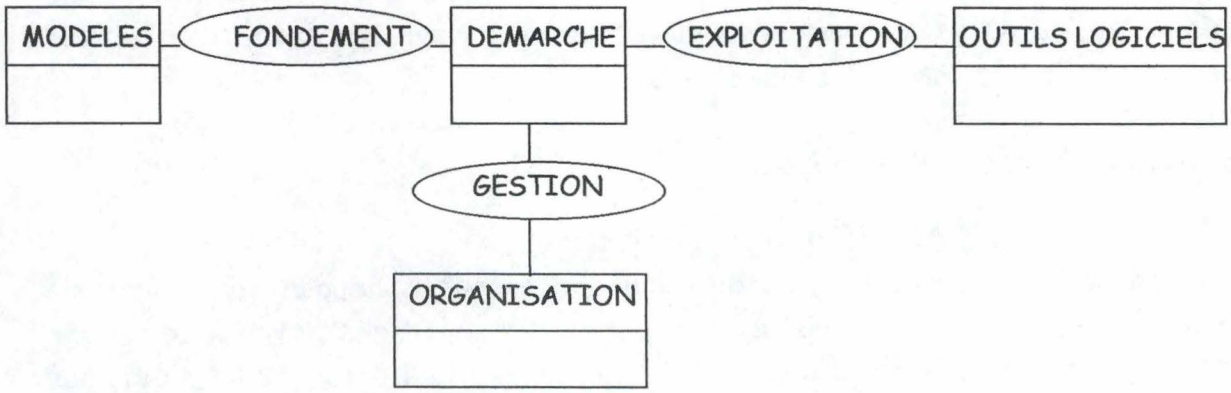


Fig. 1.5 Les pôles d'une méthode de conception d'un système d'information.

LES MODELES : La construction d'un schéma conceptuel (voir I.3.4) qui assure pleinement sa fonction de communication et de standardisation doit s'appuyer sur des concepts précis et suppose dès lors l'utilisation de modèles. Un modèle est formé de concepts et de règles relatives à leurs utilisations.

LES OUTILS LOGICIELS : Face aux difficultés rencontrées dans la conception des systèmes d'information, seul l'emploi d'outils automatisés peut aider efficacement l'analyste à vérifier que le schéma conceptuel qu'il construit est communicable, complet, cohérent, réalisable et conforme aux besoins.
Le recours à des outils automatisés est d'autant plus indispensable qu'un objectif à terme est d'automatiser les étapes de réalisation de la solution

conceptuelle retenue.

Les environnements logiciels d'aide au développement des systèmes d'information présenté actuellement sous le nom de Computer Aided Software Engineering (CASE) tendent à fournir un ensemble d'outils intégrés passifs (éditeurs de spécifications, base de spécifications, rapports documentaires, etc.) et actifs (maquettes, prototypes, outils de simulation, générateurs de plans de tests, générateurs de code gestionnaires de projets, etc.). Il faut cependant constater les limites profondes de nombreux outils CASE proposés comme support au développement des applications de gestion : certains sont d'un simplisme affligeant et laissent croire que la seule activité créatrice d'un concepteur consiste à dessiner ; d'autres au contraire sont d'une complexité extrême et mettent en avant les technologies sophistiquées à l'aide desquelles ils sont construits, au détriment des services actifs attendus par les responsables de l'informatique.

LA DEMARCHE : La démarche, qui constitue le troisième pôle de toute méthodologie de conception, doit être vue comme un ensemble de règles et de propositions générales qui précisent notamment comment mettre en œuvre les modèles et les outils automatisés, en vue de maîtriser les étapes de l'étude d'opportunité et de l'analyse conceptuelle.

I.3 Modèle conceptuel de données[4]

I.3.1 Définitions et objectifs

La méthode de structuration des données proposée repose sur le modèle Entité / Relation. Un modèle conceptuel de données (MCD) est une représentation simplifiée d'une réalité. Autrement dit un MCD n'est pas directement utilisable par une machine, mais c'est un mode de représentation intermédiaire entre la réalité observée et la machine avec son logiciel. Son objet est de mettre en lumière les caractéristiques essentielles du système d'information observé. Le modèle Entité/Relation est censé dégager des ensembles de données et les relations de sens qu'entretiennent ces ensembles. Une fois le modèle établi et validé par rapport à la réalité observée, il existe des règles permettant de le transformer en fichiers ou en base de données.

Le modèle conceptuel de base de données est un formalisme de description des informations relatives à un domaine d'application. Ce formalisme est indépendant des technologies de mise en œuvre.

Les défauts du modèle conceptuel de données sont : l'indéterminisme (un même domaine d'application peut être décrit selon une grande variété de

[4] Cette appellation est dite aussi Modèle Entité/Association (EA) rappelant les objets du modèle.

schémas Entité Association), manque de standardisation, manque de définition rigoureuse, manque d'intégration avec les autres aspects des systèmes d'information.

I.3.2 Concepts de base

L'analyse conceptuelle repose sur un modèle de structuration des informations de la mémoire du Système d'Information (le modèle ENTITE/ ASSOCIATION), et sur des modèles de structuration des traitements.

Le modèle entité/association met en évidence la notion d'ENTITE : abstraction d'un objet ou concept individualisé du réel (une interprétation proche serait de dire que cet objet réel est une entité). Les entités sont partitionnées en classes appelées TYPES d'ENTITES. Des entités peuvent être en association les unes avec les autres.

Les ASSOCIATIONS sont classées en TYPES D'ASSOCIATIONS ; chacun d'eux précise le nombre, le type et le rôle des entités participant à une association de ce type.

Chaque membre d'un type d'associations peut être caractérisé par sa CONNECTIVITE[5] (dans combien d'associations de ce type doit-on au minimum et peut-on au maximum trouver une même entité). Les propriétés des objets du réel seront représentées par des attributs que l'on adjoindra aux entités et aux associations.

On parlera d'ATTRIBUT d'un Type d'Entités ou d'un Type d'Associations ; tandis qu'on parlera de VALEURS D'ATTRIBUT attachées à une entité ou une association. Un attribut est SIMPLE ou REPETITIF, ELEMENTAIRE ou DECOMPOSABLE, OBLIGATOIRE ou FACULTATIF.

Outre ces objets de base, le modèle précise aussi la notion de CONTRAINTE D'INTEGRITE statique (définissant les états valides des données) et dynamique (définissant les changements d'états valides des données). Parmi les principales contraintes d'intégrité non encore citées figurent la notion d'*identifiant* pour les Types d'Entités et les Types d'Associations, les contraintes d'existence, d'exclusion et d'inclusion pour les Types d'Associations, les dépendances fonctionnelles pour les Types d'Entités et les Types d'Associations, ainsi que les domaines et contraintes de valeurs pour les attributs.

Enfin, la description du réel au moyen des concepts du modèle doit s'accompagner des EXPRESSIONS SEMANTIQUES précisant leur signification exacte.

---

[5] Dans quelque littérature, on dit cardinalité [Mat, 94], [Hai, 97].

I.3.3 Règles relatives au MCD

I.3.3.1 Normalisation des entités

Les entités du MCD doivent vérifier les règles suivantes dans le but d'éliminer les redondances et les anomalies de mise à jour :

PREMIERE FORME NORMALE (1FN) : dans une entité, toutes les propriétés sont simples et élémentaires et il existe au moins une clé caractérisant chaque occurrence de l'objet représenté. La clé, si elle est unique, sera prise comme identifiant. S'il y a plusieurs clés, on en choisira une comme identifiant. Toute entité doit donc avoir un identifiant.

DEUXIEME FORME NORMALE (2FN) : Toute propriété d'une entité doit dépendre de la clé par une dépendance fonctionnelle élémentaire. Autrement dit, toute propriété de l'entité doit dépendre de tout l'identifiant.

TROISIEME FORME NORMALE (3FN) : Dans une entité, toute propriété doit dépendre de la clé par une dépendance fonctionnelle élémentaire directe.

FORME NORMALE DE BOYCE-CODD (BCFN) : Si une entité a un identifiant concaténé, un des éléments composant cet identifiant ne doit pas dépendre d'une autre propriété.

I.3.3.2 Respect des contraintes d'intégrité

Le MCD doit respecter les règles de gestion exprimées dans les spécifications qui expriment ces contraintes d'intégrité.

I.3.3.3 Vérification

Dans toute occurrence d'entité ou de relation, il ne doit y avoir qu'une seule valeur de chaque propriété (non répétitivité). Pour les entités, cette règle résulte de la 1FN. Elle doit rester vraie pour les relations.

I.3.3.4 Normalisation des relations

Chaque propriété de la relation doit dépendre fonctionnellement de l'ensemble des identifiants des entités qui participent à la relation, mais d'aucun sous-ensemble de cet ensemble. Il doit y avoir une dépendance pleine des propriétés de la relation par rapport aux entités.

I.3.4 Construction du modèle conceptuel de données

Comment arrive-t-on à construire un MCD ? En voici une succession d'étapes :

- *Recueil des informations* : A la suite d'interviews des différents postes de travail du système d'information existant, on rassemble des exemplaires de tous les documents utilisés ainsi que les descriptions des divers fichiers en usage actuellement (si le système d'information est déjà automatisé). On établit la liste des propriétés à partir des documents et fichiers.
- *Constitution d'un dictionnaire de donnée* : On rassemble toutes les propriétés, leurs significations, leurs types, leurs longueurs... Si une propriété sert à des

utilisations différentes, il faudra momentanément considérer qu'il s'agit de propriétés distinctes ; quitte plus tard à réunir celles-ci en une seule, une fois repérées toutes les entités et relations.

- *Epuration du dictionnaire*: Des difficultés peuvent apparaître entre les signifiants et les signifiés. Un signifié est un objet abstrait ou concret que l'on veut qualifier. Un signifiant par contre, est un mot employé pour représenter ce signifié. Il faudra repérer les synonymes (deux signifiants pour un même signifié) et les polysèmes (un signifiant pour deux signifié).
- *Graphe des dépendances fonctionnelles*: On extrait du dictionnaire de données la liste des propriétés qui ne sont ni concaténées, ni calculées. On établit la liste des dépendances fonctiannelles dont le domaine de départ ne contient qu'une seule propriété non concaténée à partir de l'examen des documents et des identifiants évidents. Cette liste de dépendance fonctionnelle peut se visualiser sur un graphe.
- *Etablissement du MCD* : Ce graphe permettra de déterminer les entités (arcs terminaux dont les origines sont des identifiants) et les relations (arcs restants). Les règles de gestion permettront de trouver les connectivités. Il faut enfin s'assurer que les règles de vérification, normalisation et décomposition sont respectées.

I.3.5 Validation du schéma du modèle conceptuel de données

La validation d'un schéma du MCD se fait en deux étapes à savoir : les règles de complétude et celles de cohérence. Nous énumérons ces règles sans en faire une étude approfondie.

Règles de complétude : Elles sont rappelées pour chaque type primitif du MCD (Type d'entité, type d'association et attributs). Il s'agit d'identifier l'objet par son mon, sa définition, sa durée de vie, ses attributs, son identifiant...

Règles de cohérence : La cohérence d'un schéma Entité Association sera définie par l'absence de contradiction dans les spécifications et par la conformité de celles-ci à une forme canonique du schéma. Cette forme canonique est caractérisée par l'élimination ou le contrôle de la redondance et par l'élimination des ambiguïtés, elle a pour but la production d'un schéma conceptuel des informations aussi significatif et aussi stable que possible.

La section suivante se limitera à énumérer ces règles de vérification de la forme canonique d'un schéma conceptuel des informations.

- Elimination des contradictions,
- Unicité des noms et absence d'homonymes,
- Absence de synonymes,
- Tout identifiant doit être minimal,
- Contrôle des attributs dérivables,

- Elimination de structures redondantes,
- Désagrégation d'un type d'entité, d'association,
- Décomposition d'un type d'association.

I.4 Modèle Logique de données

Cette phase conduit, à partir de la solution conceptuelle, à la définition d'une solution qui soit «exécutable» par une machine abstraite, strictement indépendante des machines réelles. La solution logique est dotée des trois propriétés suivantes : correcte, efficace et indépendante de la machine réelle.
La solution logique est correcte : les algorithmes satisfont à la spécification de leur module et le schéma des données exprime toute la sémantique (y compris les contraintes d'intégrité) du schéma conceptuel.
La solution logique est efficace : l'accès aux données que réalisent les algorithmes est optimisé ; le schéma de données ne reprend que les mécanismes d'accès strictement nécessaires aux algorithmes des modules.
Un schéma logique décrit les structures d'une base de données telles qu'elles sont perçues par le programmeur ou l'utilisateur du système de gestion de base de données[6] (SGBD) ; mais en évitant les détails techniques propres à un SGBD particulier. Un schéma logique représente (implémente) un schéma conceptuel.

I.5 Modèle physique de données

Le schéma physique d'une base de données est constitué de son schéma logique, complété des constructions et paramètres techniques qui le rendent opérationnel et efficace selon les critères choisis. Les principaux critères sont liés aux performances attendues : temps de réponse, espace occupé, parallélisme maximum, simplicité de gestion, etc.
Dans les SGBD modernes (relationnels, Orientés Objets), ces constructions et paramètres sont inconnus des utilisateurs et des programmeurs (notion d'indépendance physique). Par exemple, le programmeur ignore l'existence des index lorsqu'il rédige des requêtes SQL.

Un schéma physique est fortement lié à un SGD particulier => difficulté de raisonner sur un modèle général comme pour les autres niveaux.
Cependant, certains concepts sont, dans une large mesure, communs à la plupart des SGBD :

- espaces de stockage *(fichier, tablespace, dataset,* etc.)
- clés d'accès *(index non dense, index dense, hashing,* etc.) [cfr. II.2.1.2].
- modes de stockage des enregistrements *(vrac, trié, cluster,* etc.).

[6] Est un logiciel permettant de décrire, manipuler et traiter les données d'une base.

I.6 Démarche de l'analyse fonctionnelle

Dans cette section, nous exposons un schéma [Fig. 1.6] reprenant les étapes conduisant du réel perçu (analyse d'opportunité) à l'analyse conceptuelle. Rappelons qu'une analyse d'opportunité a pour but d'élaborer un avant-projet de solution à partir d'un examen approfondi des besoins exprimés. Elle porte essentiellement sur un examen du « pourquoi » - c'est-à-dire sur les causes profondes des transformations à apporter au système d'information existant - et en réponse, sur la nature des transformations à apporter à celui-ci.

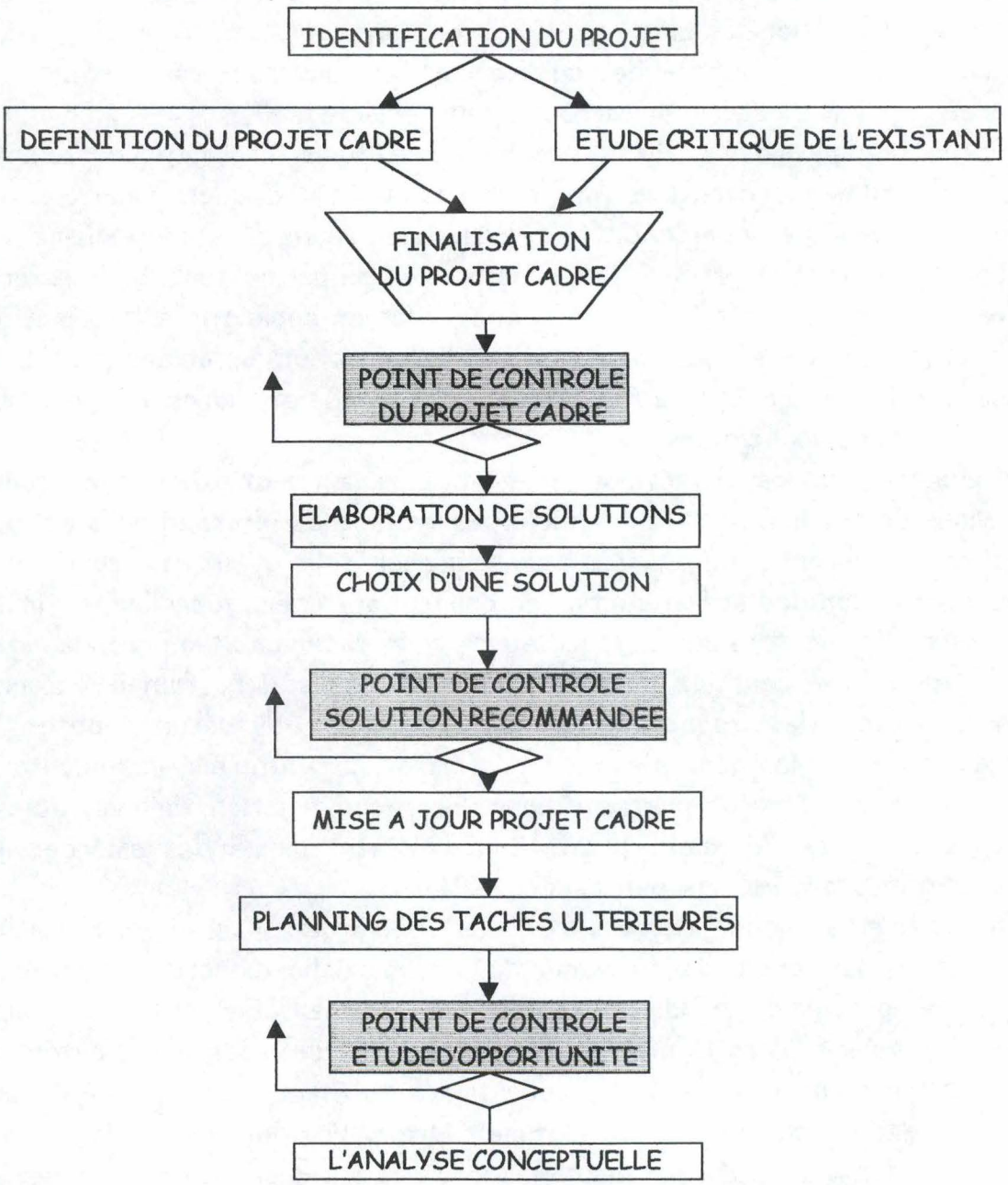


Fig. 1.6 Schéma de la démarche de l'analyse fonctionnelle.

I.7 Réalisation assistée des applications informatiques

Comme on vient de le voir, il serait dommage de ne pas profiter au maximum du soin apporté à la conception des spécifications fonctionnelles d'un système d'information dans la réalisation des applications informatiques correspondantes. En d'autres termes, il est très souhaitable que les applications informatiques soient le plus directement possible issues ou *dérivées des spécifications fonctionnelles*. Cette dérivation directe devrait contribuer de façon significative à la qualité des systèmes d'information opérationnels et à la productivité de leur développement, a fortiori de leur maintenance.

Cette dérivation directe et les avantages qui en découlent exigent notamment que la réalisation des applications informatiques soit systématique, mais sans rigidité excessive et correctement *documentée*.

Premièrement, pour garantir que les applications informatiques soient un reflet fidèle des spécifications fonctionnelles, il s'agit que leur réalisation soit la plus systématique possible. Cette systématique se traduit principalement par l'élaboration de règles générales de construction qui permettent de transformer progressivement les spécifications fonctionnelles en applications informatiques. Cette systématique et les règles de transformation associées permettent notamment d'envisager une *automatisation* plus grande et plus appropriée dans la construction des applications.

Deuxièmement, systématique et a fortiori automatisation sont souvent synonymes de rigidité dans le contexte des techniques informatiques actuelles. Celles-ci sont encore insuffisantes pour envisager, telle quelle et à court terme, la transformation des spécifications en applications opérationnelles, optimisées et exactement adaptées aux besoins. Aussi, pour éviter une trop grande rigidité et conserver une certaine souplesse dans la réalisation, faut-il encore et toujours prévoir des *adaptations manuelles*, ad hoc, du résultat obtenu par application des règles générales de transformation. Toutefois, ces adaptations ne devraient pas être un prétexte pour une reconstruction fondamentale des applications ; elles devraient plutôt être perçues comme des aménagements d'une solution standard, issue des spécifications.

Troisièmement, le recours à des techniques de transformations automatiques, laissant place à des adaptations manuelles ponctuelles, ne saurait évacuer le problème de la *documentation*, indispensable à la maintenance inévitable des systèmes d'information. Cette documentation est d'autant plus indispensable qu'une partie de la réalisation a pu être prise en charge par un automate et donc cachée à l'informaticien. Une réalisation systématique, même partiellement automatisée, ne dispense donc pas les organisations de disposer d'une documentation appropriée pour explicitement se souvenir des choix effectués, des solutions adoptées ou rejetées, des adaptations manuelles opérées.

Le schéma de la Fig. 1.7 illustre les trois composants qu'un environnement de réalisation devrait intégrer pour garantir les trois propriétés fondamentales évoquées ci-dessus.

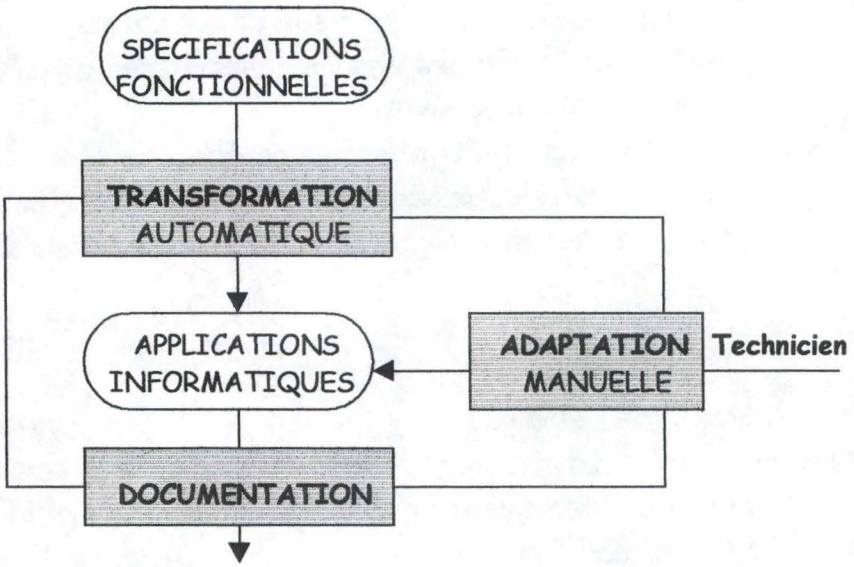


Fig. 1.7 Composants de réalisation des systèmes d'information.

I.8 Analyse fonctionnelle appliquée au cas pratique de SLLParc

Toute cette théorie a été le fondement pour l'analyse fonctionnelle effectuée dans le cadre du projet SLLParc que nous avons réalisé à Luxembourg. Le chapitre 5 de ce mémoire réserve une section de l'analyse concrète réalisée.

Nous basant sur le schéma d'enchaînement des étapes de l'étude d'opportunité décrit au point I.6 ci-haut, nous avons :

- Identifié le projet : le Responsable du service informatique à Swiss Life nous a expliqué le besoin du service et les objectifs à atteindre.
- Finalisé le projet cadre après avoir défini la frontière théorique du projet et après examen approfondi de l'existant. Ici, l'existant est l'ensemble des divers formulaires (voir annexes) servant au suivi du parc informatique. Divers entretiens avec l'utilisateur[7] nous ont permis de focaliser notre attention sur les fonctions prioritaires. Le projet se précisait de plus en plus.
- Mis à jour le projet cadre.
- Elaboré un planning des tâches ultérieures estimant que le développement de l'application respectera le temps imparti (ce qui est le cas).
- Commencé l'élaboration d'un schéma conceptuel des informations à l'aide d'un outil CASE : AMC*Designor.

[7] L'utilisateur est le Service Informatique de Swiss Life Luxembourg.

SI CELA ETAIT A REFAIRE :

- ✓ La mise à jour du projet cadre qui en fait, est la charte du projet tant vis-à-vis des utilisateurs que vis-à-vis des analystes et des informaticiens chargés de son développement, devra être un document écrit pour éviter des changements et des desiderata ultérieurs dans le cours du développement.
- ✓ Le schéma du MCD n'est qu'une ébauche à ce stade car il sera modifié au fil du développement mais conservant la sémantique. Il faut le savoir tout simplement.

I.9 Conclusion du chapitre

L'étape de l'analyse fonctionnelle dans un système d'information est essentielle. Elle est toujours faite d'une manière consciente en suivant ces principes ou tout simplement de manière empirique.

Son avantage est énorme quant à la suite de la réalisation du système d'information automatisé. Il est aussi bon de rappeler que l'on peut arriver à résoudre certaines difficultés dans une organisation par la mise en place d'une procédure d'organisation en étudiant le flux d'information par exemple, et pas par une réalisation technique nécessairement.

La fig. 1.6 est un bon résumé de la démarche de l'analyse fonctionnelle qui s'adapte à tout système d'information.

Chapitre 2 : Théorie sur les bases de données.

Sommaire

[1] SLLParc est l'application ou le logiciel sur mesure que nous avons développé à Swiss Life Luxembourg pour la gestion du parc informatique dans le cadre du stage de fin d'études.

Chapitre 2 : Théorie sur les bases de données.

II.1 Introduction

Concevoir une base de données revient toujours à concevoir un système d'information. Comment concevoir une base de données ? La meilleure solution consiste à commencer par représenter les données qui seront implémentées avec le modèle sémantique de type entité - association présenté au chapitre précédent.

Ce deuxième chapitre tentera de présenter un certain nombre de concepts essentiels sur les bases de données. Toutefois, il est difficile de distinguer ce qui est essentiel ou non dans une matière si vaste et centrale pour la conception des systèmes d'information. On comprendra déjà la difficulté qui est la nôtre de sélectionner quelques notions seulement.

II.2 Concepts de base de données.

II.2.1 Rappels sur la gestion des fichiers

II.2.1.1. Définitions

Fichier

Est un espace nommé, implanté dans un ordinateur, dans lequel des enregistrements peuvent être mémorisés, ainsi que les structures d'association et d'accès qui leur sont associées. Cette mémorisation est persistante.

Un enregistrement appartient à un et un seul fichier.
Un fichier contient 0, 1 ou plusieurs enregistrements. Ce nombre peut varier dans le temps (contrairement à un tableau).
Un fichier peut contenir des enregistrements d'un type (fichier PASCAL) ou de plusieurs types (fichier COBOL).
Ne pas confondre : fichier *(contenant)* ;
ensemble d'enregistrements *(contenu)* ;
type d'enregistrements *(format du contenu)*.
Un fichier a des attributs : nom ; date de création ; date de dernière mise à jour ; date de péremption ; créateur ; autorisations...
Les structures d'accès permettent l'insertion, la consultation, la modification et la suppression de données informatisées.

Enregistrement
Est un ensemble des champs concernant une même entité.

Champ
Est la plus petite unité d'information stockée. Chaque champ possède un type.

II.2.1.2. Organisation interne des fichiers
Nous reprenons les principaux types d'organisation interne des fichiers, essentiellement en choisissant ceux qui sont les plus utilisés dans les bases de données.

1. **Le fichier séquentiel :**

- Les enregistrements (de longueur fixe) sont ordonnés.
- Chaque enregistrement contient tous les champs dans un même ordre .

Structure et manipulation:
Pour accéder aux enregistrements, on définit une clé . Un attribut ou plus définiront l'objet décrit dans l'enregistrement. Le plus souvent, on fabrique une clé principale univoque (correspondant univoquement à un enregistrement) . Si cette clé principale n'est pas suffisante, on peut lui adjoindre des clés secondaires.

Le fichier séquentiel a été présenté sous forme de tableau (ou table) où les colonnes correspondent à des champs et les lignes à des enregistrements. Cette représentation sera très utile pour les bases de données relationnelles qui comportent en général des fichiers séquentiels.

Les fichiers séquentiels sont bien adaptés aux traitements qui s'effectuent en séquence. La structure du fichier est simple et donc facile à programmer.

Désavantage: La recherche est très lente car il faut parcourir tous les enregistrements précédant celui que l'on recherche.

2. **Fichier séquentiel indexé :**

Deux caractéristiques sont ajoutées à l'organisation précédente, pour remédier en fait aux désavantages des fichiers séquentiels:
- un index
- une zone de débordement.

Structure et manipulation: Un index est un fichier séquentiel (ou un tableau à deux entrées) qui contient deux types de champs :
le premier est une clé univoque permettant de définir un enregistrement dans le

fichier principal ;
le second est un pointeur qui désigne la place où se trouve cet enregistrement dans le fichier principal.

L'index est beaucoup plus petit que le fichier lui-même, donc plus rapide à manipuler. L'utilisation des pointeurs vers le fichier principal permet de ne plus trier celui-ci, mais de garder le fichier index trié (ceci se fait automatiquement). En pratique, on garde le fichier d'index en mémoire centrale, ce qui évite les opérations de lecture sur disque ou disquette pour accéder au fichier principal. L'index peut être constitué de plusieurs champs ; dans ce cas on parle *d'index à clé multiple.*

Lors de la manipulation de gros fichiers, la taille de l'index peut devenir elle-même trop importante. On est amené alors à créer des index d'index. Dans ce cas, l'index est dit *non-dense (* parce qu'il ne contient plus une entrée pour chaque enregistrement du fichier principal).

Une solution supplémentaire pour améliorer les performances des fichiers séquentiels indexés est d'utiliser des zones de débordement (overflow). Le principe est simple : au lieu d'insérer l'enregistrement à sa place, on l'insère en fin de fichier par exemple (overflow) et on fixe les pointeurs de façon à rétablir la continuité de la nouvelle séquence d'enregistrements (pour détails, voir [Hai,98]). Il est intéressant de prévoir des zones d'overflow aux limites des zones couvertes par les index lorsqu'ils sont non-denses.

Deux désavantages subsistent:
- la taille des enregistrements est plus grande puisqu'il existe un pointeur ;
- les index et les zones de débordement prennent de la place sur disque alors qu'ils ne contiennent pas d'informations supplémentaires.

Terminons en disant que la structure de fichier séquentiel indexé est la structure la plus utilisée dans les bases de données relationnelles car elle combine l'aspect séquentiel (avec la représentation sous forme de table) et l'aspect rapidité d'accès grâce aux index.

3. Fichier indexé :

Ce type de fichier est utilisé quand l'accès séquentiel n'est pas nécessaire. Les enregistrements sont accédés uniquement par l'index. Il n'y a pas de restriction quand à la place d'un enregistrement dans le fichier (des enregistrements de longueur variable sont donc possibles).
Un index sera associé à un attribut clé et des index peuvent exister pour tous les arguments sur lesquels une recherche se produit.

Structure et manipulation:

Il peut y avoir autant d'index que de types d'attributs dans le fichier. Un index consiste ici en un ensemble d'entrées correspondant aux champs du fichier et des pointeurs désignent la place de l'enregistrement dans le fichier principal. S'il y a plusieurs index, un répertoire (directory) d'index sera nécessaire pour savoir lequel employer Les index peuvent être *exhaustifs (* c'est-à-dire un index pointe vers tous les enregistrements) ou *non-exhaustifs* (*sélectifs,* c'est-à-dire qu'il pointe uniquement vers certains enregistrements).

Lorsqu'il y a plusieurs index, on peut passer de l'un à l'autre mais un seul est ouvert à la fois, c'est l'*index primaire* (ou maître). Les autres sont alors considérés comme des *index secondaires.*

Les fichiers indexés sont utilisés quand l'accès immédiat aux informations est important (réservation d'avions, banques d'emploi etc.). La succession des données n'est pas nécessaire et on obtient un gain de flexibilité.

Désavantage : redondance des données au niveau des index, car les mêmes données sont pointées par plusieurs index.

4. Fichier à accès direct.

La méthode à accès direct transforme une clé (univoque si possible) en une adresse dans le fichier physique, par une fonction dite de hachage (hashing), qui tient compte de la taille du fichier principal. Cette méthode est très rapide mais oblige à situer les données par rapport à une seule clé (dans les méthodes indexées, on peut changer d'index).

Structure et manipulation.

Il existe deux types d'algorithmes qui transforment la clé d'accès en une adresse.

- les algorithmes déterministes qui transforment la clé en une adresse unique (difficile à construire si le nombre d'entrées est grand) ;
- les algorithmes aléatoires qui donnent une distribution uniforme des accès. Le problème est que ces algorithmes génèrent des collisions (deux clés différentes donnent la même adresse). Plusieurs solutions sont possibles à ces collisions :
  - grouper les enregistrements par paquets,
  - changer la fonction de hashing,
  - utiliser une zone de débordement, qui est parcourue séquentiellement au moyen de pointeurs. Si une collision survient lors d'une insertion , on teste si l'enregistrement correspondant à l'adresse est occupé (champ de validité par exemple), on lit le pointeur qui indique en général la fin de fichier, et on suit un chaînage (avant ou arrière) jusqu'au premier enregistrement libre.

Le fichier à accès direct est utilisé dans des situations qui demandent des temps d'accès aussi courts que possible.
Désavantage : il y a des places vides dans le fichier ("gruyère"), si on choisit une taille du fichier principal assez grande. Par contre, si on choisit une taille trop petite , il y aura beaucoup de collisions.
La réorganisation du fichier nécessite en général, de recopier l'ancien fichier dans un fichier plus grand en recalculant toutes les adresses.
Dans la zone de débordement, les enregistrements sont accédés séquentiellement.

5. Fichiers multianneaux :

Cette méthode a pour but de trouver un groupe d'enregistrements qui contiennent des attributs communs (par exemple un groupe de clients de la même agence bancaire): dans ce cas, on parle d'anneau (ou de ring , ou encore de chaîne, car les enregistrements sont chaînés).

On peut imbriquer les anneaux (par exemple regrouper les différentes agences bancaires d'une même ville): dans ce cas, on parle de multianneaux (ou de multirings).

Structure et manipulation:
Pour expliquer la structure des fichiers à anneaux, nous allons employer directement la représentation sous forme de diagramme.

Structure de chaîne simple:
Les différents enregistrements sont chaînés grâce à des pointeurs. Le champ pointeur du dernier enregistrement pointe vers la tête de la chaîne . Un accès direct permet d'accéder à la tête. Pour simplifier la description des fichiers à anneaux, le symbole * dans le diagramme de type indique la présence d'une boucle même si un seul élément existe.
La structure de chaîne peut être unidirectionnelle (on définit seulement un pointeur suivant) ou bidirectionnelle (on définit un pointeur suivant et un pointeur précédent).
Désavantage : c'est le fait qu'il faut parcourir tous les enregistrements précédents pour atteindre un enregistrement précis. Pour remédier à ce problème, on a créé la structure de *chaîne-maître* qui consiste à ajouter un pointeur-maître qui permet de retourner à la tête de la chaîne, sans parcourir nécessairement tous les autres enregistrements. Chaque enregistrement est lié au maître, ce qui permet un gain de temps appréciable.

II.2.2 Définition d'une base de données

II.2.2.1. Définitions

Du point de vue **technique** : une base de données est une collection des données (et de leurs structures) d'un ensemble de fichiers. Est généralement gérée par un système de gestion de données (SGF[2] ou SGBD[3]). Une base de données contient un ou plusieurs fichiers.

Du point de vue **conceptuel** : une base de données est une collection des données qui décrivent un domaine d'application.

On admet qu'il existe 3 niveaux de description d'une base de données :

- **le schéma conceptuel** : description de la sémantique des structures de données; de nature abstraite, non technique, indépendante des systèmes de gestion de données (par exemple exprimée dans le modèle Entité-Association);
- **le schéma logique** : description des structures de données telles qu'elles sont perçues par le programmeur; de nature technique, liée au système de gestion des données (par exemple fichiers PASCAL, COBOL, Base de données relationnelle); le niveau de description est celui du Modèle d'Accès Généralisé (MAG[4]);
- **le schéma physique** : description des techniques d'implémentation des structures logiques; décrit des concepts tels que les pages (taille, taux de remplissage); paramètres des fichiers (localisation, taille, incrément d'allocation); tampon : taille, politique de gestion; pointeurs; index (taux de remplissage); etc.; en général ignoré du programmeur.

. L'utilisateur est intéressé par les données, c'est-à-dire par le contenu de la base de données.

. L'informaticien est intéressé par la structure des données, c'est-à-dire par les schémas de la base de données et leurs propriétés.

II.2.2.2. Propriétés

Une base de données doit posséder les caractéristiques suivantes :

- *Etre un ensemble organisé ou encore structuré :* il s'agit de stocker les données pour que leur exploitation soit efficace.
- *Etre un ensemble intégré :* il s'agit d'accéder aux mêmes données centralisées.
- *Correspondre fidèlement à la réalité :* une base de données qui n'est pas fidèle à la réalité est tout simplement inutilisable car les informations qu'on pourrait en retirer risquent d'être fausses. Pour forcer les données à rester

[2] Système de Gestion des Fichiers.
[3] Système de Gestion de Base de Données.
[4] Décrit les structures de données non seulement sous l'angle de la sémantique qu'expriment ces données, mais aussi sous celui des accès dont elles peuvent faire l'objet.

fidèles à la réalité, on définit, sur la base de données, **des contraintes d'intégrité** qui sont en fait la traduction informatique des règles de fonctionnement ou de gestion.

- *Contenir des données opérationnelles concernant un sujet donné :* dans la base de données, il faut stocker les entités et les associations. Ces deux types d'informations sont appelés données opérationnelles.
- *Etre utilisable en même temps par plusieurs utilisateurs :* le partage simultané des données implique l'existence des mécanismes de protections : confidentialité, gestions des accès concurrents, sauvegarde, reprise après pannes.
- *Etre non-redondante* : ceci est nécessaire pour assurer la **cohérence**. En effet, mise à part l'économie de la place, une donnée stockée une fois reste la même et ne peut être contredite.

II.2.3 Niveaux d'abstraction d'une base de données

Les trois niveaux d'abstraction d'une base de données qui représente son architecture sont : **niveau interne**, **niveau conceptuelle** et le **niveau externe**. Nous présentons ci-dessous [Fig. 2.1] un schéma plus détaillé de cette architecture.

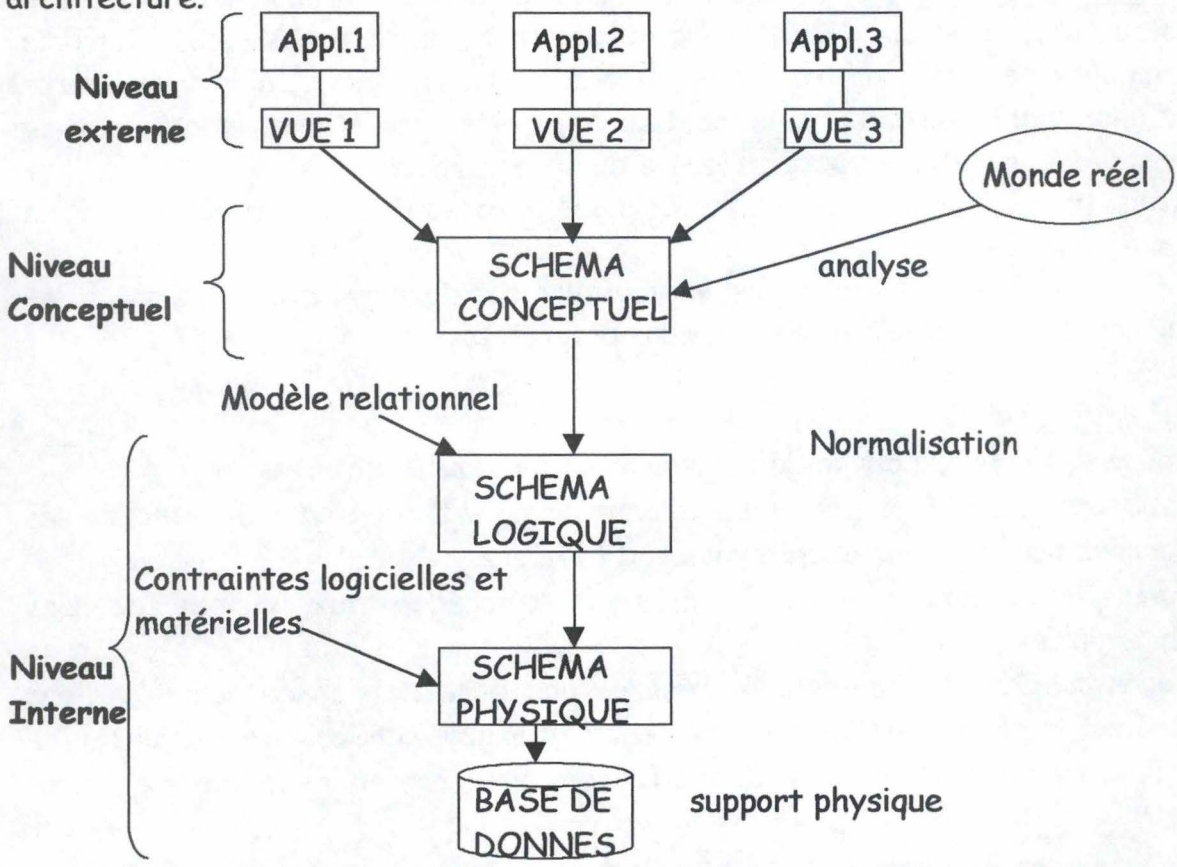


Fig. 2.1 Les différents niveaux de représentation d'une base de données.

Insistons sur le fait que seule la base de données physique possède une existence matérialisée par les octets enregistrés sur disques. Les schémas sont stockés dans la base de données : ils constituent le catalogue de la base[5]. Notons qu'en pratique, les différents schémas ne sont pas toujours aussi séparés que ne le suggère la figure ci-haut.

L'aspect fondamental de cette architecture est qu'elle rend les données et les programmes indépendants[6]. En effet, on constate que ces derniers s'adressent à la base en terme d'objets décrits au niveau externe indépendamment de spécifications physiques présentes dans le schéma du même nom. En corollaire, remarquons qu'elle autorise les conceptions séparées du schéma conceptuel qui dépend de l'univers informatisé et du schéma physique qui dépend de contraintes techniques.

II.2.4 Avantages des bases de données

Pourquoi une entreprise choisit-elle de stocker ses données dans une base de données ? La réponse la plus courte à cette question est qu'une base de données fournit à l'entreprise un moyen de **contrôle centralisé** de ses données opérationnelles. Cette propriété entraîne une série d'avantages que nous examinons ci-dessous.

- **La redondance peut être réduite**

Comme nous le savons déjà, dans les systèmes où les données ne sont pas centralisées, chaque application possède ses propres fichiers. Ceci conduit souvent à une redondance considérable pour les données stockées, avec pour résultat un vaste gaspillage de la mémoire. Mais, il y a bien plus grave encore, la redondance risque d'introduire de l'incohérence. Notons que la non-redondance n'est pas toujours facile à mettre en œuvre, et doit parfois faire l'objet d'un compromis.

Par exemple, considérons une base de données de fabrication contenant un fichier articles et un fichier des commandes. Lorsqu'un utilisateur interroge la base de données pour trouver la valeur totale des commandes d'un article donné, il peut accéder au fichier des commandes, lire la zone valeur de chaque ligne de commandes pour cet article, et additionner ces valeurs pour obtenir le total. Cette procédure peut prendre du temps, si l'article possède de nombreuses lignes de commandes. Si ce type de questions est fréquent, une meilleure approche est de gérer une zone "valeur totale" dans chaque enregistrement article. Répondre à la question est alors bien plus rapide, mais le nouveau champ est redondant.

---

[5] C'est la gestion de configuration de la base de données.
[6] Voir II.2.6

Une exception à la règle de non-redondance peut être faite pour satisfaire à des besoins de vitesse de traitements et gagner en temps de réponse.

De plus, certaines méthodes n'utilisent pas de pointeurs pour relier les données (modèle relationnel). Par exemple, placer un champ "numéro d'article" dans le fichier des articles et dans le fichier des commandes n'est pas une redondance : c'est un moyen d'implanter la liaison.

- **L'incohérence peut être évitée**

C'est en fait un corollaire du point précédent. En effet, si chaque donnée n'est enregistrée qu'une seule fois (la redondance est donc nulle), tous utilisateurs obtiendront des résultats cohérents lors de n'importe quelle interrogation.

- **Les données peuvent être partagées**

Il indique non seulement que des applications existantes se partagent les données mais aussi que de nouvelles applications peuvent être développées pour opérer sur les mêmes données. En d'autres mots, on peut créer de nouvelles applications sans devoir créer de nouveaux fichiers.

- **Des règles de sécurité peuvent être établies**

Ayant juridiction complète sur toutes les données opérationnelles, l'ADB[7] peut assurer que les accès à la base de données se font par des chemins bien définis. Ainsi, il peut définir des autorisations et des interdictions pour chaque type d'accès aux données.

Notons que sans ces mécanismes d'accès, un système de base de données est plus dangereux qu'un système de fichiers dispersés.

- **L'intégrité peut être maintenue**

Le problème de l'intégrité est le suivant : les données stockées dans la base doivent être exactes. L'incohérence entre deux enregistrements est un exemple de perte d'intégrité (ce qui ne peut survenir que s'il y a de la redondance). Mais, même si la redondance est éliminée, une base de données peut encore contenir des données fausses. Par exemple, sans prendre de précautions, on pourrait trouver qu'un employé a travaillé 200 heures sur une semaine.

Le contrôle centralisé de la base de données permet d'éviter de telles situations en donnant la possibilité au concepteur de la base de définir des contraintes d'intégrité ou des procédures de validation exécutées automatiquement chaque fois que des opérations de mise à jour sont effectuées.

---

[7] Administrateur de la base de données qui a la responsabilité de gérer les accès, de modifier la structure, d'entretenir la base…

II est important de noter que l'intégrité des données est un problème vital dans un système de base de données, justement parce que la base de données est partagée sans contrainte d'intégrité, il est alors possible pour un utilisateur d'introduire des données fausses qui iraient contaminer le travail des autres utilisateurs.

- **Les conflits d'accès peuvent être équilibrés**

Connaissant les demandes globales de l'entreprise plutôt que les demandes individuelles des utilisateurs, l'ADB peut structurer la base de données de manière à assurer le meilleur service global pour l'entreprise. Par exemple, l'organisation des données en mémoire peut être choisie pour promettre un accès rapide aux applications les plus coûteuses.

Remarquons que puisque les opérations peuvent évoluer en cours d'exploitation de la base de données, le système doit fournir à l'ADB la possibilité de modifier l'organisation de la mémoire pour conserver un bon rendement global de la base de données. Ce problème est connu sous le terme de *"tunabilify"* de la base de données.

II.2.5 Fonctionnement d'un système de base de données

Du point de vue externe, les choses peuvent être décrites simplement. Regardons par exemple, ce qui se passe dans notre système de gestion du parc informatique lorsqu'une personne désire ajouter un hardware.

L'utilisateur du SGBD est un employé qui est chargé d'encoder le nouveau matériel. II a à sa disposition un terminal qui lui permet de communiquer avec la base de données. L'employé sélectionne dans un ensemble de menus afin de faire apparaître "l'écran" qui lui permet de faire l'ajout du hardware. Sur cet écran, diverses rubriques apparaissent code du hardware, numéro de série, type, marque, nom du fournisseur, date d'achat, etc. L'employé doit alors compléter ces rubriques. Parmi celles-ci, certaines sont obligatoires (code du hardware, nom du fournisseur) et d'autres sont optionnelles (date d'achat). Lorsqu'il a terminé de saisir les informations requises, l'employé appuie sur la touche prédéfinie [valider] et le nouveau hardware est enregistré dans la base de données. Après ceci, le système est prêt à effectuer une nouvelle opération.

Le déroulement du dialogue entre l'employé et le SGBD paraît simple mais comment est-ce possible ? Quels éléments a-t-il fallu communiquer au SGBD pour que ces tâches se déroulent de façon satisfaisante ?

Lorsqu'un programmeur écrit un programme d'application, il le fait uniquement à partir de la connaissance; (de la vue) qu'il a de la base de données c'est-à-dire au

travers d'un schéma externe. Par contre, le SGBD fait des accès à des données dont l'organisation sur les disques est décrite dans un schéma interne. Un des rôles du SGBD est précisément de cacher les opérations de conversion au programmeur.
La figure [Fig. 2.2] suivante illustre cette suite des transformations.

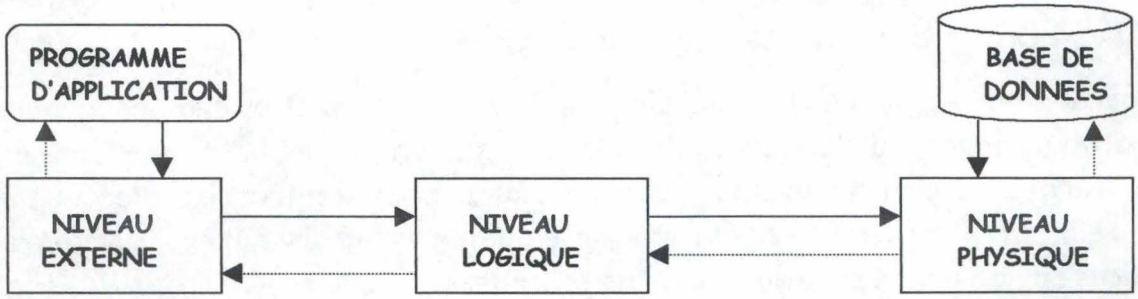


Fig. 2.2 Accès à une base de données par un programme.

II.2.6 Indépendance des données et des programmes

Dès l'exploitation des premiers systèmes de gestion de base de données, on s'est vite rendu compte qu'il fallait renforcer la notion d'indépendance des données et des programmes. Cette constatation émanait de deux phénomènes.
En premier lieu, au fur et à mesure de l'exploitation d'une base, on a constaté que sa structure logique globale évoluait toujours dans le sens d'une complexité croissante. Par exemple, on ajoute des nouvelles entités ou des associations ou encore, on décompose une entité en sous-entités. De plus, les règles de gestion du monde informatisé peuvent évoluer.

Ensuite, nous savons également qu'une base de données doit pouvoir évoluer physiquement afin d'équilibrer au mieux ("tuning") les temps d'accès des différentes applications.

Dès lors, on comprend aisément que pour assurer une pérennité maximale aux applications, et ainsi réduire leur prix de revient, il est important de pouvoir changer la structure logique ou physique d'une base sans devoir changer les programmes d'application qui l'utilisent. Dans ce contexte, les bases de données sont plus qu'une nouvelle technique de stockage et de manipulation des données. Elles impliquent une nouvelle approche de la conception et de l'utilisation des systèmes d'information.

II.3 Système de gestion de bases de données (SGBD)

II.3.1 Définition et description

Le logiciel qui permet à un utilisateur d'exploiter une base de données[8], est le Système de Gestion de Base de Données (SGBD). Un des rôles fondamentaux d'un SGBD est de permettre à un utilisateur d'exploiter les données en lui cachant les détails de l'organisation et de la localisation des données sur disques.

Un SGBD est constitué d'un ensemble de fonctions. Parmi ces fonctions, certaines gèrent l'organisation des données sur disques, alors que d'autres permettent d'effectuer des recherches et des mises à jour dans la base. Un SGBD possède également des fonctions permettant de définir des règles d'accès aux données et des contraintes d'intégrité.

II.3.2 Fonctions d'un SGBD

Citons quelques fonctions importantes d'un SGBD.

• Description et définition

Un SGBD doit offrir au concepteur de la base, encore appelé administrateur, la possibilité de créer la base, de définir ses paramètres physiques et les objets qu'elle contient. Les objets que l'administrateur définit peuvent se situer à différents niveaux. C'est ainsi que l'administrateur est amené à définir tant la taille initiale d'un fichier que la structure d'objets abstraits, comme par exemple, les vues dans une base relationnelle.

• Manipulation

Les utilisateurs doivent pouvoir manipuler les données de la base. Manipulation est, ici, un terme générique pour désigner tant la recherche d'informations que l'ajout, la suppression ou la modification de données. Dans les bases de données relationnelles, le mode de manipulation des données est interactif. Ce mode est réservé aux utilisateurs informaticiens qui expriment leurs requêtes grâce à un langage spécialisé (Par exemple SQL[9]). Ce mode interactif ne permet pas à lui seul, de développer des applications. Mais, il est utilisé pour réaliser la mise au point des requêtes d'accès à la base. Ces requêtes seront ensuite incluses dans un programme écrit en langage de 3e ou 4e génération pour former une application.

Les utilisateurs finals, quant à eux, s'expriment au moyen de menus d'écrans de

[8] C'est-à-dire décrire manipuler et traiter les données d'une base.
[9] Pour SQL, voir II.5

saisies prédéfinis gérés par les applications ci-dessus.

• Intégrité

Nous avons écrit que les données contenues dans une base devaient représenter la réalité. Pour ce faire, le SGBD doit posséder deux fonctions. Il doit d'abord permettre de définir des contraintes d'intégrité représentant les règles de gestion du système informatisé. Ceci est réalisé grâce à la fonction DEFINITION décrite plus haut. Il doit ensuite assurer qu'à tout moment, les valeurs présentes dans la base, ou celles qu'un utilisateur tente d'introduire respectent les contraintes.

• Confidentialité

La nécessité d'avoir une fonction permettant de définir et de faire respecter des règles de confidentialité est une conséquence directe du fait que dans une base, les données sont partagées par plusieurs utilisateurs. A ce stade, on peut dire simplement que la fonction confidentialité permet d'assurer que chaque utilisateur n'effectue que des opérations autorisées sur certaines données.

• Concurrence d'accès

II arrive fréquemment que des utilisateurs différents tentent d'accéder en même temps aux mêmes données. Dans ce cas, on parle d'accès concurrents aux données. Si les accès concurrents ne sont pas traités correctement, ils peuvent introduire des incohérences dans la base de données. C'est par exemple le cas, lorsque pour une base portant sur la gestion des billets d'avion, deux demandes de réservation de places pour un même vol ne sont pas traitées séquentiellement. Les fonctions que nous venons de décrire ne sont certainement pas les seules que doit posséder un SGBD. Nous pensons notamment à la fonction qui assure qu'après une panne, les utilisateurs pourront reprendre l'exploitation à partir d'une base dont les données ne sont pas perdues et sont cohérentes.

D'autre part, un SGBD offre également un ensemble de fonctions permettant d'effectuer des sauvegardes et des recouvrements de données ainsi que des fonctions d'analyse des accès à la base pour pouvoir adapter les paramètres de celle-ci afin qu'elle offre le meilleur service global pour toutes les applications. A ce titre, on comprend aisément que ces fonctions sont indispensables à l'administrateur de la base. Il est intéressant de noter, que dans ce domaine, il existe des outils complémentaires[10] au SGBD proprement dit.

[10] Comme le moniteur transactionnel qui gère la file d'attente des accès concurrents pour une base de données.

II.3.3 Architecture d'un SGBD

Il n'existe pas d'architecture standard pour les SGBD. Cependant, si l'on se focalise sur les SGBD relationnels qui représentent actuellement 90% du marché, on constate qu'ils possèdent tous une architecture proche de celle présentée à la figure ci-dessous. En première analyse, on remarque que les SGBD sont constitués de deux parties.

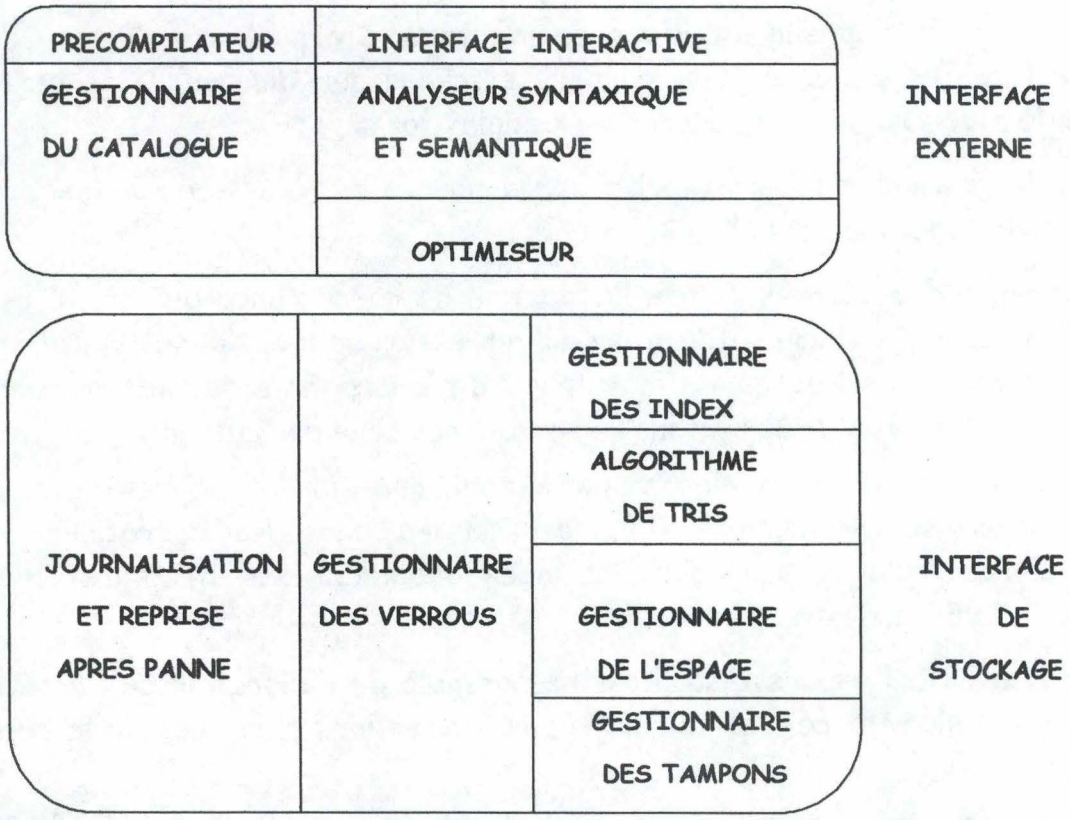


Fig. 2.3 Architecture d'un SGBD.

La première partie, que non appelons **interface externe**, reçoit les requêtes venant d'une application ou d'une interface interactive, analyse la requête, génère du code constitué d'appels à des primitives et optimise ce code. Inversement, elle joue aussi le rôle de couche de présentation pour formater le résultat des requêtes soumises au SGBD. Cette partie du SGBD assure les fonctionnalités de description, utilisation, intégrité et confidentialité.

Le module gestion du **catalogue** est très important. En effet, dans le jargon des bases de données, un catalogue est une "mini" base de données contenant des informations sur une base de données. C'est pour cette raison qu'on l'appelle également **métabase**. Généralement, dans les systèmes relationnels actuels, le catalogue d'une base est stocké dans la base elle-même et peut être interrogé au même titre que les données de la base. Sans entrer dans les détails, disons

que le catalogue d'une base contient la description de tous les objets présents dans la base, mais aussi les contraintes d'intégrité, les autorisations d'accès (confidentialité) et les valeurs courantes des paramètres physiques de la base. Dans la majorité des cas, c'est l'administrateur de la base qui extrait des informations du catalogue. Dans les systèmes relationnels, ceci peut se faire directement en interactif en utilisant SQL ou au moyen d'outils spécialisés offrant une plus grande convivialité (menu, fenêtres, ...).

La deuxième partie s'appelle **interface de stockage**. C'est elle qui s'occupe de tout ce qui concerne l'accès aux données stockées sur disques. Dans cette interface de stockage nous retrouvons les modules suivants :

Le rôle de la **journalisation et reprise après pannes** est d'assurer la fonction de sécurité de fonctionnement.

Le gestionnaire des verrous assure la fonction de concurrence d'accès. C'est grâce à ce module que chaque utilisateur a l'impression qu'il est le seul à utiliser la base de données. Ceci est loin d'être trivial à réaliser. En effet, il faut faire en sorte que le travail d'un utilisateur ne trouble pas celui des autres.

Le gestionnaire des index et algorithmes de tris opèrent dans le même but. Il s'agit de minimiser le nombre d'entrées/sorties dans les opérations de recherches. Pour cela, on a imaginé des index possédant des structures très élaborées (B-TREE, index hachés, etc.).

Le gestionnaire de l'espace disque est responsable de l'allocation de l'espace disque. C'est également ce module qui gère les extensions physiques de la base de données.

Le gestionnaire des tampons gère l'utilisation des tampons *(buffers)* en mémoire centrale. C'est ce module qui gère les entrées/sorties physiques entre les disques et les tampons en mémoire. Par exemple, il transfère le contenu des tampons vers les disques ou recopie une partie de la base dans les tampons. Ce gestionnaire utilise des techniques très sophistiquées comme celles des lectures anticipées pour minimiser le nombre d'entrées/sorties physiques.

II.3.4 Déroulement d'une recherche dans un SGBD

Pour expliquer, étape par étape, comment une recherche a lieu, considérons un exemple. Supposons qu'un membre souhaite la liste des utilisateurs actifs du parc informatique. Pour ce faire, après avoir parcouru un ensemble de menus, il va activer un programme.

La figure suivante montre toutes les étapes qui doivent être exécutées afin d'obtenir la liste souhaitée. Nous supposons que nous travaillons avec SGBD relationnel.

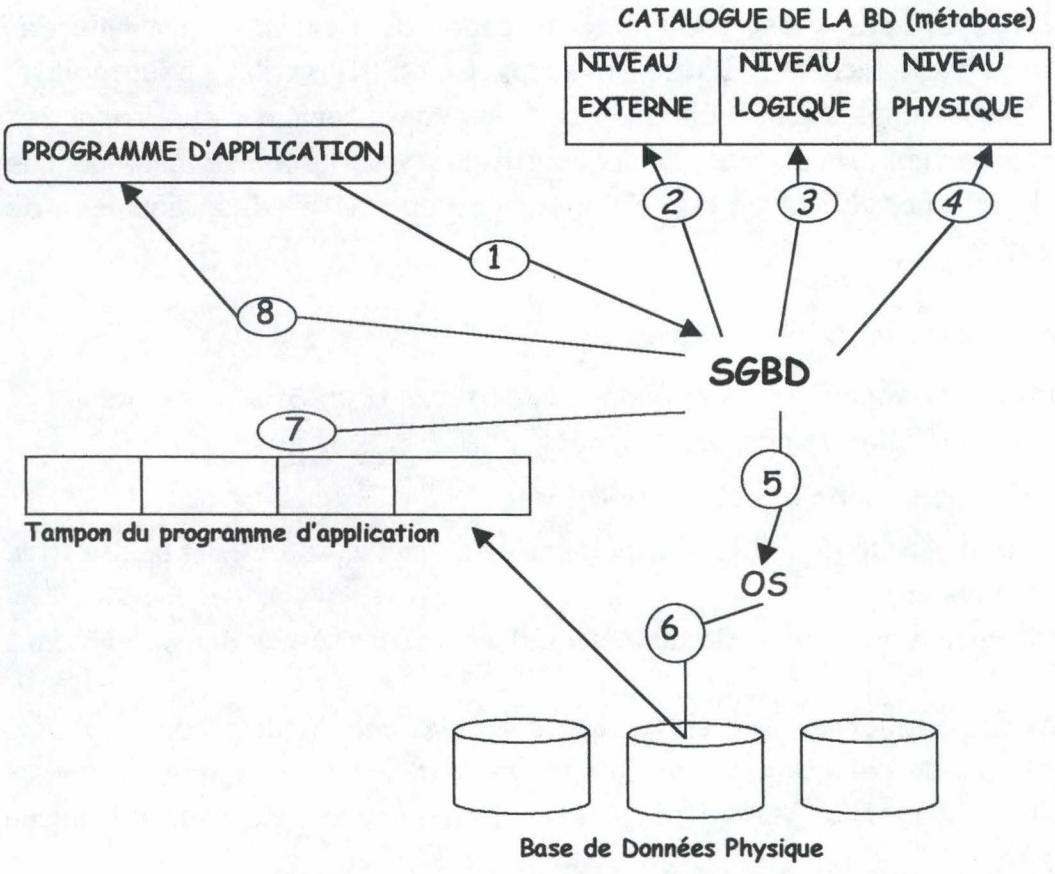


Fig. 2.4 Déroulement d'une recherche.

La figure Fig. 2.4 fait apparaître les principaux éléments suivants :

- Le SGBD qui assure le dialogue entre le programme d'application et la base de données.
- Le système d'exploitation de l'ordinateur sur lequel le SGBD est en exploitation.
- Les différents schémas : externe, logique et physique qui sont stockés dans le catalogue de la base de données.
- Les tampons du programme d'application dans lesquels vont être amenées les pages constituant la base physique.

II.4 Modèle relationnel (SGBDR)

1. Introduction

La naissance de l'approche relationnelle remonte à l'année 1970 lorqu'un mathématicien, E.F. Codd, membre de l'IBM San José Research Laboratory, publie un article intitulé « A Relational Model of Data for Large Shared Data Banks ».

L'attrait pour les systèmes relationnels ne relève pas d'une simple mode passagère. Ces systèmes s'inscrivent dans le cadre de l'évolution normale des techniques informatiques : facilité et souplesse d'utilisation, adaptabilité, concepts de bases simples à appréhender pour les non-informaticiens, capacités à valider l'information enregistrée, homogénéité du produit sur une large gamme de matériel et possibilité d'accroître la productivité des équipes de développement.

2. Les données dans un SGBD relationnel

Nous présenterons dans un premier temps les critères d'un SGBD relationnel et ensuite dans un deuxième temps, les concepts.

2.1. Critères définissant un SGBD relationnel

Le Relational Task Group (R.T.G.) a défini 5 critères pour qu'un SGBD puisse être qualifié de relationnel :

a) Toute information de la base de données est représentée par une valeur dans une table.

b) il n'y a pas de pointeurs ou lien entre tables visibles par l'utilisateur.

c) Les opérateurs de sélection, projection, équi-joint utilisant le même attribut dans les deux relations, doivent être réalisés sans aucune limitation de niveau interne.

d) Tous les autres opérateurs de base de l'algèbre relationnelle sont réalisés sans aucune limitation de niveau interne.

e) La contrainte de la clé et la contrainte référentielle sont gérées par le système.

Si les trois premiers critères sont vérifiés, le système est dit "minimalement relationnel". Si tous les critères sont vérifiés, il est alors appelé "complètement relationnel".

2.2. Les concepts d'un SGBD relationnel concernant les données

Domaine, attribut, relation, dépendance fonctionnelle

Un *attribut* A_i *est* une variable qui prend ses valeurs dans un ensemble D_i appelé *domaine.*

Une *relation* R définie sur $U = \{A_1, \ldots A_n\}$ est un sous-ensemble du produit cartésien $D_1 \times \ldots\ldots \times D_n$ (les domaines ne sont pas nécessairement distincts).

L'attribut B est dit fonctionnellement dépendant de l'attribut A si, étant donné deux n-uplets $< a_1, b_1, C_1 >$ et $< a_2, b_2, c_2 >$ de R, $a_1 = a_2$ implique $b_1 = b_2$. C'est toujours la même valeur de b qui est associé à une valeur de a. La dépendance fonctionnelle est notée : $A \rightarrow B$.

Extension et intension d'une relation

L'intension correspond au schéma de relation et l'extension à une occurrence du schéma de relation.

Clé primaire, clé candidate, clé étrangère

Une clé candidate est un sous-ensemble minimal d'attributs tel qu'il existe une dépendance fonctionnelle de ce sous-ensemble vers tout autre attribut de la relation n'appartenant pas à ce sous-ensemble.

Une clé primaire est une des clés candidates.

Une clé étrangère est un groupe d'attributs qui peut apparaître comme une clé primaire dans une autre relation.

Contrainte d'unicité de la clé

Il ne peut exister deux n-uplets possédant la même valeur pour une occurrence de relation.

Contrainte d'intégrité référentielle

Toute valeur d'une clé étrangère dans une relation doit exister dans une autre relation comme valeur de clé primaire.

Contrainte d'entité

Tout attribut participant à une clé primaire ne doit pas avoir de valeur nulle. Mais une clé étrangère peut avoir des valeurs nulles. Toutefois, il est possible de spécifier le contraire dans les SGBD.

3. Définition du modèle

Il repose sur la notion de *domaines de valeurs* : un *domaine* est un ensemble de valeurs définies en extension ou en compréhension : par exemple le domaine des entiers positifs (compréhension) ou celui des couleurs principales : violet, indigo, bleu, vert, jaune, orange, rouge (extension).

Une *relation* est construite à partir d'un ensemble de domaines D_1, D_2, ... D_n (non nécessairement distincts) et se définit comme un sous ensemble du produit cartésien $D_1 \times D_2 \times \ldots D_n$. Elle correspond donc à un tableau de valeurs auquel un nom est attribué. Les lignes de ce tableau s'appellent des *n-uplets* et les colonnes représentent des valeurs prises dans chacun des domaines. Pour pouvoir distinguer une colonne d'une autre (notamment dans le cas de domaines identiques), des noms sont donnés aux colonnes : ces noms constituent les *attributs* de la relation.

Pour chaque relation, il faut définir l'ensemble d'attributs dont les valeurs permettent de distinguer un n-uplet d'un autre : cet attribut ou ensemble d'attributs s'appelle la *clé de la relation* (on peut être amené à choisir une clé parmi plusieurs ensembles potentiels).

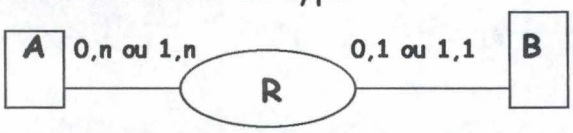
Notons que les attributs d'une relation sont *simples* c'est à dire qu'ils ne peuvent être eux mêmes des relations (on est dans le contexte de relations *normalisées).*

4. Passage du MCD Entités/Relations au MLD[11] relationnel

Les entités deviennent des relations au sens relationnel.

Leurs propriétés deviennent des constituants (l'identifiant devenant une clé).

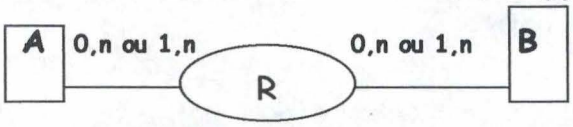
- Une relation R du MCD de type :



disparaît dans le MLD relationnel, l'identifiant de A étant incorporé à la relation B (au sens relationnel).

Si R est porteuse de propriétés, celles-ci deviennent des constituants de B.

- Une relation R au sens entités/relations de type :



devient une relation au sens relationnel, sa clé étant obtenue en concaténant les identifiants des entités qui participent à cette relation.

Si R est porteuse de propriétés, celles-ci deviennent des constituants de la relation relationnelle R.

II.5 Manipulation des données : Le langage SQL

II.5.1 Introduction et généralités

Le langage de requête le plus répandu aujourd'hui est le SQL, acronyme de Structured Query Language. La plupart des SGBD (Systèmes de Gestion de Bases de Données) qui présentent les données sous la forme de tables proposent un langage de requête dénommé SQL. Présenté pour la première fois en 1973 par une équipe de chercheurs du laboratoire d'IBM à San José (USA), il a rapidement été adopté comme standard. SQL est disponible sur tous les types de matériel, depuis les micro-ordinateurs jusqu'aux plus gros ordinateurs. On a recensé en 1992 plus de 150 logiciels offrant le langage SQL[12].

L'exposé que nous présentons ici n'est pas complet, le lecteur étant renvoyé à la littérature spécifique, largement disponible dans les librairies.

---

[11] Modèle Logique de Données.

[12] Le Monde Informatique du 15/02/1993 et qui écrit : « Au travers des normes SQL2 et SQL3, l'une approuvée par l'Ansi, l'autre encore en chantier, le langage de requête SQL se dote peu à peu de fonctions propres à un L4G ».

II.5.2 Manipulation des données

SQL possède trois composantes :
DDL: Data Definition Language - utilisé pour définir les objets (tables, vues, index etc.) lors de la conception de la BD ou de sa modification (par le DBA[13]) ;
DML: Data Manipulation Language - utilisé pour manipuler les données (consulter, insérer, supprimer..) (par les programmeurs et utilisateurs) ;
DCL: Data Control Language - utilisé pour définir les accès, mots de passe etc.. (par le DBA).

Sous SQL, chaque table de la base de données est considérée comme un ensemble de lignes et de colonnes. L'intersection de chacune d'elles contient une valeur. Les commandes SQL spécifient les données à extraire des tables (langage déclaratif) et non la procédure à employer pour y accéder.

La commande d'interrogation est SELECT. La modification de la base de données revêt un triple aspect : ajout, mise à jour et suppression. Dans SQL, il s'agit respectivement des commandes INSERT, UPDATE, DELETE.

- Création d'une table : CREATE TABLE *nom_table* (*liste_attribut_type*) ;
- Recherche de base : SELECT [ALL | DISTINCT] *clause_de_selection*
  FROM *nom_table* [WHERE *condition*] ;

*clause_de_selection : := * | nom_table.* | liste_colonne*

condition := [NOT]
condition_de_base
| condition_between
| condition_in
| condition_like
| condition_null
| condition AND | OR condition
| (condition)

condition_de_base : := colonne = |<> |< | > | =< | >= constante

[13] DBA : Data Base Administrator.

- Recherche de base avec jointure : SELECT [ALL | DISTINCT]
  clause_de_selection FROM *liste_table_ref*
  [WHERE *condition*] ;

  clause_de_selection ne change pas.

  table_ref : := *nom_table [synonyme]*

  condition reste la même mais devons enrichir la notion *condition_de_base* pour autoriser la comparaison entre attributs compatibles.
  condition_de_base : := colonne oper_comp constante / colonne
  oper_comp : := = <> / < / =< / > / >=

Nous recommandons vivement au lecteur d'étendre cette étude des commandes SQL dans les livres spécialisés suggérés dans la bibliographie. Cette introduction n'a montré que le premier pas.

II.5.3 Les vues

Une vue correspond à une table *virtuelle* dont seule la définition est stockée et non le contenu. Une requête SQL peut extraire des données de tables réelles (stockées dans la base) ou virtuelles (vues). Il est même possible sous certaines conditions de modifier les données d'une vue (c'est-à-dire d'ajouter, supprimer, modifier des lignes). Dans ce cas, le SGBD répercute sur les données réelles les modifications demandées. D'une manière générale, on considérera qu'une vue peut être utilisée comme une table réelle.

Le but d'une vue est d'offrir à un utilisateur de la base de données une présentation des données qui soit adaptée à ses besoins (adéquation aux besoins particuliers). Il peut aussi être possible de restreindre l'accès aux données présentées par une vue, et auxquelles il est autorisé à accéder. Il peut enfin protéger un utilisateur de l'effet de modifications de la structure de la base de données.

En SQL, une vue est définie au moyen d'une *expression_de_selection* de la manière suivante :

Créer_vue : := CREATE VIEUW *nom_vue* [(*liste_colonne*)]
AS *expression_de_selection*
[WITH CHECK OPTION] ;

Au lieu d'évaluer l'*expression_de_selection,* et matérialiser ainsi son contenu, on peut considérer qu'une telle commande définit une table virtuelle à partir des tables de la clause FROM de l'*expression_de_selection.* Les tables présentes dans la clause FROM sont appelées tables sources. Il peut s'agir des tables de la base ou des vues.

II.5.4 Programmer avec SQL

Un langage autonome comme SQL est insuffisant à lui seul pour écrire des applications accédant à une base de données.

En effet, on voit mal comment on pourrait, par exemple en SQL, gérer des menus, intercepter une donnée introduite par un utilisateur ou venant d'un autre ordinateur, ou encore générer des rapports. Bref, il est impossible de construire une application "base de données" uniquement au moyen de SQL.

Dans cette section, nous voulons simplement dire qu'on peut écrire des programmes associant ces commandes. Dans ce genre de programmes, les commandes SQL sont utilisées pour les accès aux données de la base.

La norme SQL2 exige que chaque implémentation supporte au moins trois modes d'utilisation de SQL à savoir : le mode interactif (direct SQL), le mode SQL intégré *(embedded SQL)* et le mode module *(SQL modular).*

II.6 Algèbre relationnelle

Les opérateurs ensemblistes et relationnels forment à eux deux ce que l'on appelle **l'algèbre relationnelle ou le langage algébrique de CODD** que nous rappelons dans ce paragraphe et le suivant. Ce langage, a été inventé par CODD. Il peut être considéré comme une collection d'opérateurs portant sur des relations. Il est caractérisé par les propriétés suivantes :

- **Fermeture** l'application d'un opérateur relationnel sur une ou des relations génère toujours une relation qui peut à son tour être utilisée comme argument de nouveaux opérateurs.
- **Ensembliste** *("set at a time")* : il n'y a pas de variables représentant un tuple d'une relation. Au lieu de cela, le résultat d'une requête est toujours un sous-ensemble d'une ou plusieurs relations.
- **Non-procédural** : l'utilisateur qualifie le résultat qui L'INTERESSE (LE QUOI) ; le système détermine la meilleure procédure d'accès aux données recherchées (LE COMMENT).
- Universel : le langage algébrique de CODD possède un caractère universel. Son étude constitue un réel tremplin pour l'étude des langages supportés par n'importe quel SGBD relationnel commercialisé.
- Indépendance : les opérateurs sont basés sur des valeurs d'attributs ce qui constitue le seul moyen d'accès. Tous les accès multi-relations sont effectués par des comparaisons entre valeurs d'attributs (définis sur des domaines compatibles) ce qui permet de très grandes potentialités d'accès totalement

indépendantes de l'implantation. Il n'y a pas de chemins d'accès explicites comme dans les autres schémas logiques (hiérarchiques et réseaux).

Les opérateurs ensemblistes de base sont des opérateurs binaires, c'est-à-dire qui, à partir de deux relations, en génèrent une troisième. Les opérateurs que nous allons présenter ici, sont l'union, la différence, l'intersection et le produit cartésien. Ils ne sont pas tous indépendants. Par exemple, il est bien connu que l'intersection peut être définie à partir de la différence.

II.6.1 Les opérateurs ensemblistes

Union : $X = R_1 \cup R_2$

- L'union de deux relations R_1 , R_2 union-compatibles[14] est une relation X contenant l'ensemble des tuples appartenant à R_1 ou à R_2 ou aux deux.
- Requête SQL correspondante : select * from R_1 union select * from R_2

  On peut aussi sélectionner une colonne (projection et union) :

  select * from R_1 union select * from R_2

Différence : $X = R_1 - R_2$

- La différence de deux relations R_1 , R_2 union-compatibles et dans l'ordre ci-dessus, est une relation X contenant les tuples appartenant à R_1 et n'appartenant pas à R_2 .
- Requête SQL correspondante : select * from R_1 intersec[15] select * from R_2

Produit Cartésien : $X = R_1 \times R_2$

- Le produit cartésien de deux relations R_1 , R_2 (de schéma quelconque) est une relation X ayant pour attributs tous les attributs de R_1 et de R_2 et dont les tuples sont constitués de toutes les concaténations possibles d'un tuple de R_1 à un tuple de R_2.
- Requête SQL correspondante[16] : select * from R_1, R_2

II.6.2 Les opérateurs relationnels

Deux opérateurs unaires selection et projection combinés avec les opérations ensemblistes ci-dessus permettent de définir toutes les expressions correctes de l'algèbre relationnelle.

Projection :

- La projection d'une relation sur certains constituants consiste à ne retenir que ces constituants, tout en éliminant les tuples dupliqués.

---

[14] Relations de même arité (même nombre d'attribut).

[15] L'intersection peut s'exprimer à partir de la différence : $R_1 \cap R_2 = R_1 - (R_1 - R_2)$

[16] Cette requête ne correspond pas souvent au résultat recherché d'où l'usage de la jointure.

- Notation : PROJ.nom_de_la_relation (liste des constituants retenus[17])
- En SQL : **select distinct** liste des constituants retenus **from** nom_de_la_relation

Sélection :

- La sélection ou la restriction consiste à construire une relation en sélectionnant les tuples de la relation manipulée qui vérifient une certaine propriété.
- Notation : R1 = SELECT.R (P)[18]
- En SQL : **select** liste_des_constituants_retenus **from** nom_de_la_relation **where** condition.

II.6.3 Les opérateurs additionnels

Intersection : $X = R_1 \cap R_2$

- L'intersection de deux relations R_1 , R_2 union-compatibles est une relation X contenant les tuples appartenant à R_1 et à R_2 .

Jointure : $X = R_1 \times R_2$

- La jointure de deux relations R_1 , R_2 selon un critère généralisé c est l'ensemble des tuples du produit cartésien $R_1 \times R_2$ satisfaisant le critère c .
- En SQL : **select** liste_des_constituants_retenus **from** noms_de_2_relations **where** condition2[19].

II.7 Outils CASE

II.7.1 Définition d'un outil CASE

- CASE est l'acronyme de **C**omputer **A**ided **S**oftware **E**ngineering.
- Un outil CASE assiste l'informaticien dans sa démarche durant le cycle de vie d'une application. Il nous permettra d'implémenter le schéma conceptuel de données à partir duquel l'on générera le schéma logique puis physique. Aussi, le code SQL pour créer la base de données et les divers rapports.
- Il existe plusieurs outils CASE parmi lesquels: Rational Rosé, DB-MAIN[20], AD/Cycle, AMC *designor[21], designer 2000, Paradigm+, Together, ...

---

[17] Noms des colonnes.
[18] Où R1 est la relation résultat, R la relation manipulée et P une proposition logique faisant intervenir des constituants de R, des opérateurs relationnels et voire des opérateurs booléens.
[19] Cette condition2 met dans une même proposition logique les 2 colonnes liées de deux relations.
[20] Data Base Maintenance développé à l'Institut d'Informatique des FUNDP à Namur.
[21] Nous l'avions utilisé à Swiss Life Luxembourg lors du développement de SLLParc.

II.7.2 Classification d'un outil CASE

1 ° Critère

Horizontal

- un CASE tool horizontal supporte une activité qui s'étale sur plusieurs périodes méthodologiques,
- traçabilité, configuration, documentation, process modeling, ...

Vertical

- Un CASE tool vertical est spécifique à une étape méthodologique,
- reverse engineering, outil de design, analyse, simulation, validation.

2 ° Critère

UpperCASE

- destiné aux phases d 'analyse en amont du cycle de développement,
- analyse, requirements, transformation, validation, spécification, modélisation de données ;

LowerCASE

- destiné aux phases d 'implémentation en aval du cycle de développement,
- éditeur, gestion de version, debugger, performance, RAD, ...

II.8 Base de données de SLLParc.

Après avoir obtenu le schéma du modèle conceptuel de données par l'outil CASE AMC*Designor, nous avions généré le code de la base de données. Ce code sql crée les tables et les index de la base de données.
Il faudra rappeler que ce code peut être modifié au niveau de type et longueur de champs, des index... afin de l'adapter aux besoins de l'utilisateur. Par exemple certains seront ajoutés pour l'accès rapide suivant les besoins.

Si c'était à refaire :

- ✓ Les index primaires qui sont uniques rassemblant deux ou trois attributs constituent une contrainte impossible d'obtenir pratiquement. Ils seront scindés en index secondaires (pas unique) pour chaque attribut concerné.
- ✓ Se rappeler que le code de la base de donnée généré par l'outil CASE peut faire l'objet de multiples et diverses modifications suivant les besoins de l'utilisateur et du programmeur (qui connaît le langage d'implémentation).

II.9 Conclusion du chapitre.

Nous avons consacré ce chapitre à rappeler les concepts sur les bases de données. Nous recommandons au lecteur d'approfondir ses connaissances dans cette vaste matière de lire des volumes spécialisés indiqués dans la bibliographie.

Les notions sur les bases de données telles que les transformations des schémas, la technologie des bases de données, la rétro-ingénierie,... n'ont pas fait l'objet de nos rappels.

Le schéma suivant conclut ce chapitre en rappelant succinctement la démarche de conception d'une base de données. En pratique, c'est le schéma conceptuel de données édité grâce à un outil CASE qui constitue le résultat de cette étape. De ce schéma est généré le code Sql, modifiable, de la base de données à créer.

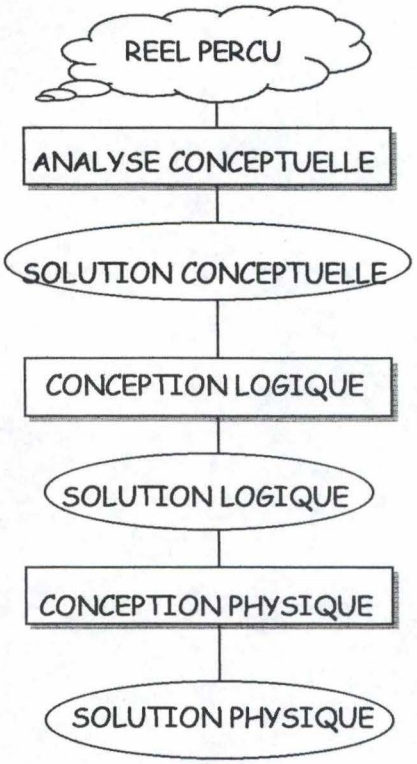


Fig. 2.5 Schéma de la démarche de conception d'une base de données.

Chapitre 3 : Théorie sur la conception des Interfaces Homme-Machine (IHM).

Sommaire

[1] SLLParc est l'application ou le logiciel sur mesure que nous avons développé à Swiss Life Luxembourg pour la gestion du parc informatique dans le cadre du stage de fin d'études.

Chapitre 3 : Théorie sur la conception des Interfaces Homme-Machine (IHM).

III.1 Introduction

Ce chapitre rappelle quelques notions de la conception de l'interface utilisateur. Après avoir proposé certains principes généraux de conception, applicables à n'importe quelle interface utilisateur, ce chapitre sera essentiellement consacré à la conception d'interfaces graphiques.

La conception de systèmes informatiques couvre un large champ d'activités : de la conception du matériel à la conception de l'interface utilisateur. S'il a toujours été de la responsabilité des ingénieurs électroniciens de concevoir le matériel, c'est souvent aux informaticiens que revient la responsabilité de la conception de l'interface utilisateur, en plus de la conception du logiciel. Bien que les spécialistes des facteurs humains soient de plus en plus impliqués dans la conception des interfaces utilisateur, leur rôle se limite à donner des conseils plutôt qu'à concevoir l'interface.

L'interface utilisateur est souvent l'aune à laquelle on mesure un système. Si une interface utilisateur est difficile à utiliser, dans le meilleur des cas, elle provoquera beaucoup de fautes d'utilisation, et dans le pire des cas, le système sera rejeté quelles que soient ses fonctionnalités.

Une interface mal conçue peut être à l'origine d'erreurs d'utilisation potentiellement catastrophiques. Si l'information est présentée de manière confuse ou trompeuse, l'utilisateur peut se méprendre sur le sens d'une information et lancer une série d'actions dangereuses.

Bien que ce chapitre ne puisse pas couvrir l'interface utilisateur dans son intégralité, mon but est de présenter le sujet aux développeurs du logiciel en espérant qu'ils éviteront ainsi de dangereuses erreurs de conception. Pour un traitement plus complet de la conception d'interfaces utilisateur, le lecteur est invité à se reporter à des ouvrages spécialisés référencés dans la bibliographie.

Dans ce chapitre, nous mettrons l'accent sur les interfaces graphiques plutôt que sur les interfaces textuelles. Les terminaux sont de plus en plus puissants et ont des moniteurs à haute résolution capables d'afficher plusieurs polices de caractères et de mélanger textes et graphiques.

III.2 Problématique de conception des IHM

III.2.1 Notions

III.2.1.1 *Définition*

Une interface est un système informatique utilisé par une personne pour réaliser une tâche accomplie à l'aide d'un ensemble de moyens informatiques par l'intermédiaire d'actions exercées sur des objets interactifs.

Une interface est l'intermédiaire entre des actions voulues par l'utilisateur (représentations mentales) pour réaliser sa tâche et la technologie mise à sa disposition. C'est la représentation des actions offertes par les moyens.

III.2.1.2 *But de l'interface*

Le but de l'interface est de permettre à l'utilisateur de réaliser sa tâche efficacement. Ce qui signifie que l'interface doit être **utile et utilisable**.

Une interface sera dite **utile** si elle fournit les fonctions nécessaires à l'utilisateur pour mener à bien les tâches qui lui sont assignées, c'est-à-dire pour produire les résultats attendus dans les conditions requises.

Une interface sera dite **utilisable** si les moyens qu'elle fournit pour réaliser la tâche sont compatibles avec le profil cognitif de l'utilisateur et n'implique pas, de façon contraignante pour celui-ci, des actions étrangères à la nature de la tâche.

Remarquons que la prise en compte des perspectives d'utilité et d'utilisabilité est récente en informatique (1985) : elle traduit le souci d'une démarche de conception des systèmes d'information « **centrés sur l'utilisateur** » alors que depuis leurs débuts, les développements de l'informatique étaient « centrés sur la technique ». Ces deux perspectives dépassent largement le domaine des IHM au sens étroit : elles concernent tous les aspects de l'informatique en liaison avec les personnes tels que les langages de programmation, les modèles de spécification, les méthodes de développement, les outils de software engineering, etc.

III.2.1.3 *Enjeux économiques*

- Permettre à des personnes étrangères au monde des ordinateurs de les utiliser effectivement dans leur domaine : enfants, handicapés, gestionnaires, médecins, etc.
- Accroître la productivité des utilisateurs spécialisés.

III.2.2 Principes de conception

La conception d'une interface utilisateur doit prendre en compte les besoins, l'expérience et la capacités de l'utilisateur. Il faut **impliquer les utilisateurs potentiels dans le processus de conception**, et utiliser le **prototypage** rapide pour développer une interface utilisateur. Les utilisateurs doivent pouvoir disposer du prototype de manière à pouvoir incorporer leurs réactions dans la conception.

Du fait du grand nombre d'utilisateurs et d'applications, nous donnerons un guide de construction d'interface plus général que spécifique. Certaines sociétés ont leur propre style d'interfaçage et leur propres guides (comme chez Swiss Life Luxembourg), mais il y a certains principes généraux que l'on peut appliquer à toutes les conceptions d'interface utilisateur. Bien sûr, on peut formuler beaucoup de principes, mais je pense que ceux énumérés dans l'encart suivant sont les plus importants.

1. L'interface doit utiliser des termes et des concepts qui soient familiers à l'utilisateur.
2. L'interface doit être consistante.
3. Le système ne doit pas surprendre l'utilisateur.
4. L'interface doit comporter des mécanismes qui permettent à l'utilisateur de corriger ses erreurs.
5. L'interface doit comporter une forme de guidage de l'utilisateur.

Fig 3.1 Principes de conception d'interface utilisateur.

Détaillons ces principes :

1. On ne doit pas forcer les utilisateur à s'adapter à une interface parce qu'elle est plus facile à implémenter. L'interface doit utiliser des termes familiers à l'utilisateur et manipuler des objets qui ont des analogies directes avec l'environnement de l'utilisateur.

Par exemple, si un système est conçu pour être utilisé par une équipe de secrétaires, il doit manipuler des objets comme des lettres, des documents, des agendas, des dossiers, etc. Il doit offrir des opérations comme «classer», «trouver», «indexer», «jeter», etc. En pratique, on implémentera ces objets à l'aide de fichiers ou d'entités dans des bases de données, mais on ne doit pas forcer la (le) secrétaire à assimiler des concepts informatiques comme les fichiers de travail, les répertoires, les identificateurs de fichiers, etc.

2. Pour qu'une interface soit consistante, il faut que les commandes système et les menus aient le même format ; que le passage des paramètres soit effectué de la même manière pour toutes les commandes ; et que la ponctuation des commandes soit homogène, etc. La consistance d'une interface

permet de réduire le temps d'apprentissage, car on peut appliquer les connaissances apprises sur une commande à d'autres parties du système. Par exemple, mettons qu'une commande du système accepte différents types de paramètres des noms de fichiers et des options. Si on veut différencier les noms de fichiers des options par un moyen quelconque (par exemple en faisant précéder les options du caractère «-»), on devra respecter cette notation pour toutes les commandes du système.

3. L'utilisateur a tendance à s'irriter lorsque le système informatique ne se comporte pas de la manière attendue. Lorsqu'il utilise un système, l'utilisateur en construit un modèle mental et il s'attend à ce que le système agisse conformément à ce modèle. Si une action, dans un contexte, cause un type de changement particulier, on s'attend à ce que la même action, dans un autre contexte cause un changement comparable. S'il arrive quelque chose de différent, l'utilisateur est à la fois surpris et dérouté. Les concepteurs doivent donc s'assurer que des actions comparables ont des effets comparables.

4. Les utilisateurs font inévitablement des fautes lorsqu'ils utilisent un système. La conception de l'interface peut minimiser ces erreurs (par exemple en utilisant des menus pour éliminer les fautes de frappe) mais on ne peut jamais éliminer toutes les erreurs. C'est pourquoi l'interface devra comporter des moyens permettant à l'utilisateur de corriger ses fautes. Il y en a de deux types :

(1) La confirmation des actions destructives. Si un utilisateur lance une action potentiellement destructive, on lui demandera de confirmer que c'est ce qu'il veut réellement faire avant de détruire quoi que ce soit.

(2) La possibilité d'annuler, qui permet de restaurer le système dans son état d'avant l'action. Plusieurs niveaux d'annulation peuvent s'avérer nécessaires car quelquefois, les utilisateurs ne se rendent pas compte immédiatement de leurs fautes. D'un point de vue pratique, c'est assez difficile à réaliser, si bien que la plupart des systèmes ne permettent d'annuler que la dernière commande.

5. Enfin, les interfaces doivent intégrer des fonctionnalités d'aide à l'utilisateur. On doit pouvoir accéder à ces derniers depuis la console d'utilisation et obtenir différents niveaux d'aides et de conseils. Cela devrait aller de l'information de base sur la manière de démarrer le système jusqu'à une description détaillée des fonctionnalités du système et de leur utilisation. Ces possibilités d'aides doivent être structurées de manière à ce que l'utilisateur ne soit pas submergé d'information lorsqu'il demande de l'aide.

En résumé, ces principes mettent l'accent sur le fait que le processus de conception de l'interface doit être **centré sur l'utilisateur**. Les utilisateurs de systèmes informatiques essaient de résoudre leurs problèmes à l'aide d'ordinateurs, mais il y a encore beaucoup de systèmes qui ne tiennent pas compte des besoins et des limitations de ces utilisateurs. Le concepteur devra toujours garder à l'esprit le fait que les utilisateurs ont une tâche à accomplir, et que l'interface doit être orientée vers cette tâche.

III.2.3 Problèmes dans la conception : Tâche et Interface.

La nature des problèmes d'interfaçage est liée à la réduction du fossé qui sépare **l'univers psychologique** dans lequel une personne pense la tâche à accomplir (objectif, intention) et **l'univers physique** de l'interface dans lequel elle accomplit les actions nécessaires à la réalisation de la tâche.

Une personne exprime ses objectifs et ses intentions en des termes significatifs pour elle, en des termes liés à la représentation mentale qu'elle se fait de son problème, c'est-à-dire des termes psychologiques.

Elle doit traduire ses objectifs dans des actions à exécuter à l'aide des mécanismes du système interface exprimés en des termes physiques.
De même, elle doit évaluer la réalisation de ses objectifs à partir des variables d'état du système interface qui sont des variables physiques.

Il apparaît donc que le problème central d'une interface réside dans la transformation de variables psychologiques en variables physiques et dans la transformation inverse. Moins de différences qu'il y aura entre ces deux types de représentation des connaissances, plus aisées et plus directes seront ces deux transformations et plus faciles et plus commodes seront l'apprentissage et l'utilisation de l'interface.

Une interface sera d'autant meilleure qu'elle requiert peu d'efforts cognitifs (mentaux) de la part de l'utilisateur aux plans de l'exécution et de l'évaluation.

- Au plan de l'exécution, une bonne interface devrait fournir des objets, des commandes et des mécanismes aussi proches que possible des représentations mentales des utilisateurs (golfe d'exécution) .
- Au plan de l'évaluation, l'interface devrait fournir des dispositifs de sortie (d'affichage) dont le modèle conceptuel est facilement perçu, interprété et évalué par l'utilisateur (golfe d'évaluation).

III.2.4 Efforts cognitifs et leurs réductions.

III.2.4.1 *Efforts cognitifs*

L'effort cognitif requis par un utilisateur peut être caractérisé par la distance qui sépare les variables psychologiques des variables physiques de l'interface. C'est-à-dire la distance entre la tâche que l'utilisateur pense accomplir et la façon dont la tâche peut être accomplie, via l'interface. Plus cette distance sera réduite et moins dur sera l'effort requis pour établir un pont entre les représentations de l'utilisateur et la façon de les réaliser dans le système.

III.2.4.2 *Moyens de réduire ces efforts cognitifs*

On analyse la réduction de l'effort cognitif requis des utilisateurs en fonction de deux paramètres :

- la distance contenue dans le langage de l'interface (distances sémantique et articulatoire) [cfr. Fig.3.2],
- l'implication directe de l'utilisateur.

III.2.4.2.1 Distances contenues dans le langage de l'interface

On établit une distinction entre la **distance sémantique et la distance articulatoire** attachées au langage de l'interface.

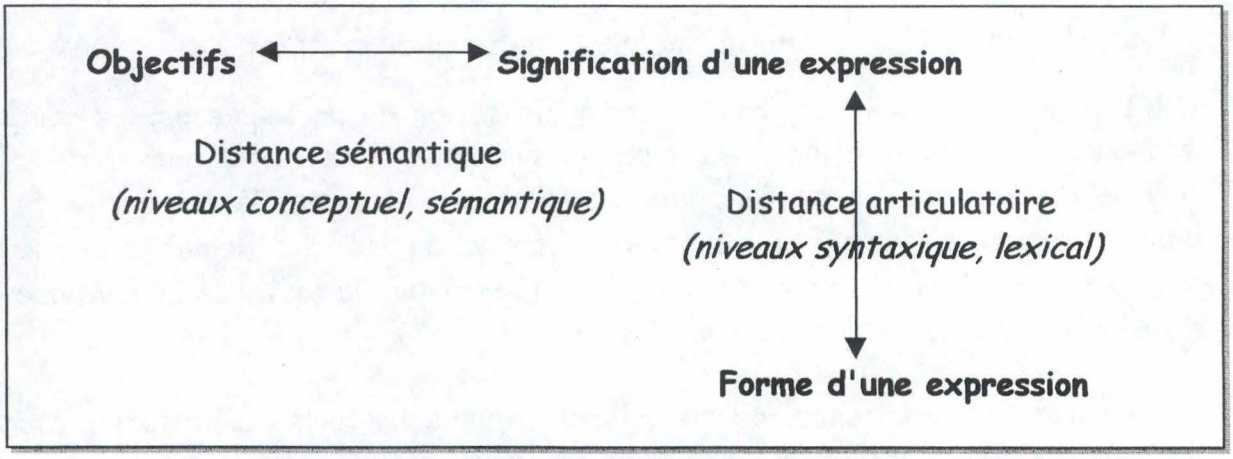


Fig. 3.2 Langage de l'interface.

On appelle langage de l'interface, le langage utilisé pour indiquer aux dispositifs de l'interface la nature des actions que l'on veut exécuter :

- langage de commande ;
- sélection d'une action dans un menu ;
- cliquer sur une icône, un élément dans une image ;
- remplir un formulaire ;
- introduire une requête textuelle, vocale,... ;
- déclencher un dispositif dans un environnement de « réalité virtuelle » ou de manipulation directe à distance ; etc.

1. Distance sémantique : Elle répond à la question principale : « peut-on exprimer dans le langage de l'interface ce que l'on veut réaliser ? ».

La distance sémantique résulte des choix de conception effectués à deux niveaux d'abstraction : le niveau conceptuel et le niveau sémantique.
La distance sémantique sera d'autant meilleure (courte) qu'il y a peu d'informations étrangères à la tâche pensée par l'utilisateur.

Comment réduire la distance sémantique? Pour la réduire, on peut considérer :

- l'utilisation des langages de haut niveau permettant de décrire la tâche au niveau de l'interface dans un langage très proche de celui utilisé dans le domaine de la tâche elle-même. Plus le langage est de haut niveau, plus il est proche du langage du domaine de la tâche elle-même et plus il y a de possibilités qu'il soit spécialisé.

- de faire en sorte que l'output exprime les concepts sémantiques directement.

- l'adaptation de l'utilisateur au modèle conceptuel du système.

L'utilisateur modifie sa propre conceptualisation du problème pour s'adapter au mode de représentation fournit par le système : il se met à penser dans les termes du système et en fonction des concepts offerts. Ce qui comporte un danger d'appauvrissement de l'univers conceptuel de la tâche. Remarquons que les automatismes dans le comportement ne réduisent pas la distance sémantique de même que la mémorisation ou la virtuosité.

- la qualité de l'assistance fournie à l'utilisateur sous forme de tutorial, d'aide en ligne passive ou active de gestion des erreurs,...

2. Distance articulatoire : Elle est l'objet de ces questions : « peut-on déduire de la forme d'une expression sa signification ? Quelle relation existe-t-il entre la forme physique des expressions du langage et leur désignation ? ».

Ces formes physiques concernent les éléments en input et les éléments en output.

La distance articulatoire résulte des choix de conception effectués à deux niveaux d'abstraction : le niveau syntaxique et le niveau lexical.

Le niveau lexical concerne la dépendance avec les dispositifs matériels (clavier, souris,...) et les mécanismes par lesquels l'utilisateur spécifie la syntaxe (symboles, codage, format,...).

Le niveau syntaxique définit comment les unités lexicales (mots) sont assemblés en « phrases » qui indiquent à l'ordinateur d'exécuter des actions. A chaque style d'interaction (langage de commande, menus, remplissage de formulaires,...) couplé avec un dispositif d'interaction (boîte de dialogue, menus déroulants, commande vocale,...) est associé une syntaxe particulière. Le niveau syntaxique définit comment faire pour réaliser une opération donnée.

Exemples :

- Déplacer un curseur à l'aide d'une souris engendre une distance articulatoire minimale car l'expression physique mimétise l'intention ; mais déplacer un curseur à l'aide de touches de commande engendre une distance importante car il n'y a pas de mimétisme.
- Exprimer le changement de valeur d'une variable sous la forme d'un graphique ou d'un tableau de nombres correspond à des distances articulatoires différentes alors que la distance sémantique est la même.

III.2.4.2.2 Degré d'implication de l'utilisateur.

L'effort cognitif devrait également être réduit en fonction du sentiment d'implication directe éprouvé par l'utilisateur.

On distingue deux situations opposées sous l'angle de l'implication directe suivant que l'interface est construite :

- sur la métaphore de la conversation ;
- sur la métaphore du mini-monde.

1. Interface construite sur la métaphore de la conversation

Le véhicule de l'interface est un langage dans lequel l'utilisateur et le système ont une conversation au sujet d'un monde supposé mais non explicitement représenté.

L'utilisateur est en relation directe avec des structures linguistiques, structures qui peuvent être interprétées comme se référant à des objets d'intérêt mais qui ne sont pas ces objets eux-mêmes.

L'interface n'insère pas l'utilisateur dans une situation donnée : L'interface est entre un opérateur et un ordinateur considéré comme un outil. L'interface joue le rôle d'intermédiaire entre l'utilisateur et l'ordinateur. L'utilisateur agit comme un programmeur qui donne des instructions à un ordinateur.

Exemples : Langage de commande de UNIX, langage de requête SQL.

2. Interface construite sur la métaphore du mini-monde

L'interface devient elle-même un monde dans lequel l'utilisateur peut agir ; ce monde change d'état en réponse aux actions de l'utilisateur.

L'interface en tant que tel disparaît, elle devient invisible. Les opérations sont quasiment faites dans le domaine de la tâche.

Il n'y a plus d'intermédiaire entre l'utilisateur et le monde dans lequel il veut agir.

L'interface est envisagée dans un contexte mimétique : elle devient un objet mimétique. L'utilisateur est un acteur qui assure un rôle dans un scénario qui est une représentation mimétique d'une situation du monde réel.
L'utilisateur agit à la première personne et de ce fait éprouve un sentiment d'implication directe.

La métaphore du mini-monde a donné lieu au style d'interaction appelé manipulation directe qui est appliquée dans de nombreux domaines tels que le traitement de texte, les tableurs, les jeux-vidéo, les systèmes d'information géographiques, les hypermédias, la conception assistée par ordinateur dans les métiers de l'ingénieur, de l'architecte, de l'imprimeur, de l'informaticien, du neurochirurgien, dans le diagnostic médical, les services de sécurité, la maintenance à distance,...

3. Conditions requises d'une interface pour fournir ce sentiment d'implication directe :

- Les langages d'input et d'output doivent être inter-référentiels : le même objet doit pouvoir être utilisé comme entité d'input et entité d'output. C'est parce qu'une expression d'input peut contenir une expression d'output préalable que l'utilisateur a le sentiment que l'expression d'output est la chose elle-même

et que l'opération est appliquée directement à la chose elle-même (traitement de texte, tableurs, systèmes graphiques, ...).

- Les objets de l'interface et les actions à effectuer sur ceux-ci doivent correspondre à des métaphores visuelles ou spatiales significatives pour l'utilisateur.

- Les actions doivent pouvoir être effectuées à l'aide d'opérations physiques au lieu de l'emploi d'une syntaxe complexe.

- Le système doit procurer une réponse immédiate (réponse en temps réel) : il ne peut exister de délai entre l'exécution et les résultats sauf si ce délai est intrinsèque au domaine. La rapidité du feed-back et la continuité dans la représentation de l'état du système contribuent à éliminer la perception de l'ordinateur comme intermédiaire et sont, dès lors, des facteurs essentiels pour provoquer l'illusion d'une implication directe.

- L'interface ne doit pas interférer avec les actions mimétiques de l'utilisateur, sous peine de détruire l'implication à la première personne. Interférence sous la forme de messages d'erreurs, de help, d'interruption pour procéder à des sauvetages de données, etc.

En résumé, ces conditions conduisent à minimiser les distances sémantiques et articulatoires. L'examen des conditions requises pour créer une IHM basée sur la métaphore du mini-monde en montre les avantages indéniables qui expliquent son succès. On peut également en extraire les difficultés et les limites : choix de métaphores adéquates, risque d'ambiguïté dans l'interprétation des symboles visuels ou auditifs, surcharge éventuelle des représentations visuelles ou spatiales, lenteur éventuelle pour un utilisateur expérimenté due à l'emploi des dispositifs de pointage par rapport à l'usage du clavier, etc.

III.3 Présentation des matériaux de construction d'une IHM

Ce paragraphe est consacré à une simple présentation des principaux matériaux qui interviennent dans la construction d'une IHM.

Les principaux matériaux de construction d'une IHM sont :

Un Objet Interactif Concret (OIC) tel qu'un menu, une fenêtre physique, une boîte de dialogue, un bouton-radio, un objet 3D,... est manipulé à l'aide de Moyens ou Dispositifs d'Interaction (MPI) tels qu'un clavier, un écran tactile, une souris, un outil de reconnaissance de la parole (speech récognition), un générateur vocal,...

Un OIC est produit par un outil de présentation (tel que Open Look ou Graphical Device Interface) exploité à l'aide d'un outil graphique (tel que SUN WINDOWS ou MS WINDOWS).

Un OIC concrétise la représentation abstraite d'un Objet Interactif Abstrait (OIA). Ces derniers sont indépendants des environnements physiques d'exploitation que sont les outils graphiques et les outils de présentation. Les OIA sont aux OIC ce que les modèles logiques de données sont aux SGBD.

Un style de dialogue ou SDI (langage de commande, langue naturelle, manipulation directe,...) est piloté par l'organisation du dialogue. Celle-ci est caractérisée par les attributs suivants : contrôle du dialogue (externe, interne ou mixte), le séquencement du dialogue (monofil, multifil), le mode de dialogue (synchrone, asynchrone, mixte) et le type de déclenchement des services (fonctions) fournis par l'application (déclenchement automatique à l'initiative de l'utilisateur).

Le contrôle exercé sur un OIA résulte d'une action en provenance d'un MDI agissant dans le contexte d'un SDI ou d'une combinaison de SDI.

III.4 Ergonomie logicielle des IHM.

III.4.1 Différentes formes d'ergonomie

"L'ergonomie est une discipline dont l'objet est l'étude du travail de l'homme. Son objectif est l'adaptation du travail à l'homme" [2].

Liées à l'évolution du travail, on distingue différentes formes d'ergonomie dont :

Ergonomie physique

Elle porte sur les aspects physiques du travail. L'homme a un corps et une force dont il faut tenir compte pour développer les postes de travail. La mécanisation permet d'alléger le travail physique, elle essaie donc d'être ergonomique. L'ergonomie physique prend en compte la diversité des aptitudes physiques telles que, par exemple :

- la position du corps :
  - hauteur du siège, de la table de travail,
  - inclinaison du buste, etc.

[2] Selon Karnas, 1987.

- la vision :
  - vision périphérique,
  - vision des couleurs,
  - angle de vue, distance de vue,
  - sensibilité aux contrastes,
  - capacité d'identifier un objet dans un contexte,
  - fatigue visuelle, etc.

Ergonomie de l'information

Lorsque la technologie se substitue à la force physique, l'homme doit commander et contrôler le dispositif mis en œuvre. Par conséquent, des problèmes de communication d'information apparaissent entre l'homme et le dispositif qu'il commande.

L'opérateur humain réagit selon des stéréotypes. En conséquence, par exemple, pour accroître l'intensité lumineuse, statistiquement la tendance de réponse consiste à tourner le bouton dans le sens des aiguilles d'une montre.

Dans la conception d'un poste de travail, il est donc nécessaire de respecter la compatibilité entre les dispositifs d'information et les dispositifs d'action. Par exemple, entre la représentation de fonctions à l'écran (déplacement du curseur, sélection dans un menu) et la commande de ces fonctions à l'aide de moyens d'interaction (MDI).

Ergonomie des systèmes

II s'agit de la prise en charge réelle des interactions entre une cellule de production constituée du couple (homme-machine) et les autres cellules de production par des dispositifs de communication homme-machine (communication verbale, signaux, tableaux d'affichage,...). Il s'agit d'une extension de l'ergonomie de l'information.

Cette forme d'ergonomie est liée au développement des processus intégrés dans l'organisation du travail (exemples : fabrication, organisation administrative). Elle joue un rôle de plus en plus important avec l'expansion des technologies telles que les systèmes de Workflow, l'EDI (Electronic Data Interchange). Il importe, en particulier, que la mise en place de systèmes de communications formelles n'engendrent pas une altération de la communication interpersonnelle causant, de ce fait, un risque d'appauvrissement de la tâche et de diminution de la motivation.

Ergonomie cognitive ou ergonomie logicielle

Elle tient compte des caractéristiques de l'homme au niveau cognitif pour améliorer les performances du couple homme-machine.

Rappelons que ces caractéristiques concernent la capacité de traitement de l'information de l'individu : perception (temps de perception, sélection), mémorisation, apprentissage, intelligence (aptitude à choisir une action appropriée à une situation). Précisons que différents facteurs influenceront les performances cognitives d'un individu, notamment :

- la charge mentale,
- la connaissance des résultats,
- l'isolement,
- l'anxiété, la crainte.

L'ergonomie cognitive joue un rôle majeur dans les différents aspects de la conception d'une interface Homme-Machine : la présentation des objets du dialogue (textes, graphiques, messages d'erreur, aides), la conversation entre l'homme et la machine, la guidance ou l'assistance.

La psychologie cognitive fournit des fondements à l'ergonomie logicielle. Ces fondements sont cependant trop limités pour procurer une base globale à la conception ergonomique des IHM. L'approche retenue par les concepteurs d'IHM est de recourir à des règles ergonomiques dont certaines trouvent leur fondement dans la psychologie cognitive ; mais la majorité sont des règles empiriques dégagées par la pratique.
Préalablement à l'analyse de règles ergonomiques, on examinera successivement dans les sections suivantes : les critères d'évaluation de l'ergonomie d'une interface et les critères de conception.

III.4.2 Critères d'évaluation de la qualité d'une interface

Nous avons vu que le but d'une interface est de permettre à un agent de réaliser efficacement la tâche qu'il veut accomplir. A cette fin, une interface doit être utile et utilisable.

Différents critères ont été proposés pour évaluer ces caractéristiques de la qualité d'une interface. Nous en retenons cinq :

- **Temps d'apprentissage** des dispositifs nécessaires à l'exécution d'une tâche.

- **Rapidité d'exécution** d'une tâche.

- **Taux d'erreurs** effectuées par l'utilisateur. On évaluera :
  1) *la fréquence des erreurs* en distinguant deux types d'erreurs :
     - les erreurs d'exécution (erreurs syntaxiques) généralement liées à des défauts de manipulation ;
     - les erreurs d'intention lorsque l'utilisateur sélectionne une commande inappropriée (interprétation incorrecte).
  2) *le temps de correction* (diagnostic, recommandation, correction).

- **Période de rémanence** durant laquelle un utilisateur conserve la connaissance acquise : heures, jours, semaines,... C'est une fonction directe de l'effort cognitif requis pour planifier une séquence d'actions qui réalisent une intention donnée. C'est une fonction indirecte du temps d'apprentissage et de la fréquence d'utilisation.

- **Satisfaction subjective à utiliser le système**. Elle peut se traduire par un sentiment de confort, d'enrichissement.

A ces cinq critères, nous pouvons aussi ajouter celui du :

- Degré de couverture des dispositifs de l'interface par rapport aux actions qui font partie de la tâche de l'utilisateur.

Ces critères d'évaluation n'ont pas une portée absolue : ils doivent être pondérés en fonction du contexte d'exécution de la tâche. A titre illustratif, considérons quatre classes de situations (contextes) :

- **Les situations vitales** telles que celles de contrôle du trafic aérien, de surveillance d'une réaction nucléaire, de monitoring dans un service de soins intensifs, etc. Dans ces contextes, l'apprentissage peut être très long, afin d'arriver à un taux d'erreur voisin de zéro, à une bonne rétention et à une rapidité d'exécution adéquate ; il est indispensable que l'interface procure une couverture maximale.

- **Les applications industrielles et commerciales** qui exigent un temps d'apprentissage limité, une grande rapidité d'exécution et une couverture adéquate.

- **Les applications domestiques et l'emploi de jeux** où l'on doit privilégier la satisfaction individuelle et la rapidité d'apprentissage.

- **Les applications créatives** qu'il s'agisse des systèmes d'information d'aide à la décision des différents types d'applications dont la réalisation est assistée par ordinateur (.AO) ou d'activités à caractère artistique. Dans ce contexte, la satisfaction individuelle devient le critère majeur à vérifier.

III.4.3 Critères de conception d'une interface

Les règles ergonomiques ont pour mission de respecter un ou plusieurs critères ergonomiques lors de la conception d'une interface homme-machine.

Un *critère ergonomique* constitue une dimension reconnue comme menant à une interface utile et utilisable. Par conséquent, ces critères forment des fondements des choix en matière d'interface homme-machine.

Cette approche est :

. **empirique**, car elle est basée sur une classification de critères progressivement établie à partir de recommandation ;

. **non exhaustive**, car elle est incomplète et non entièrement validée sur le terrain;

. **réductrice**, car chaque critère général est décliné suivant les niveaux linguistiques.

Les critères ergonomiques généraux identifiés et retenus sont au nombre de huit :

- la compatibilité,
- la cohérence,
- la charge de travail,
- l'adaptabilité,
- le contrôle du dialogue,
- la représentativité,
- le guidage,
- la gestion des erreurs.

Nous donnons dans la suite une définition et un objectif à atteindre pour chaque critère ergonomique.

La compatibilité

Définition : une interface homme-machine est qualifiée de *compatible* si le (re)codage d'informations et de tâches du monde réel en données et actions du système est réduit. En effet, une interface est d'autant meilleure qu'elle est compatible car le transfert d'informations est d'autant plus rapide que le (re)codage en données est réduit, l'accomplissement de tâche est d'autant plus rapide que son interprétation en action est courte. La compatibilité est ici interprétée comme une cohérence avec l'environnement extérieur à l'application (par exemple les attentes de l'utilisateur, ses habitudes comportementales, les

procédures de gestion, les documents sources).

Objectif : le but est de réduire le besoin de traduire, de transposer, d'interpréter l'information en données du système, de raccourcir l'interprétation de la tâche en actions du système, de minimiser les références à la documentation lors de l'évaluation.

La cohérence

Définition : une interface homme-machine est qualifiée *de cohérent* si les données et les actions sont facilement identifiables, reconnaissables et utilisables. En effet, les données sont d'autant mieux perçues et les actions d'autant mieux accomplies qu'elles sont présentées de manière stable et uniformisée. Ce critère concerne la cohérence d'une application avec d'autres applications ou elle-même.
Objectif : le but est de recourir aux mêmes moyens pour arriver aux mêmes résultats dans des contextes similaires. En standardisant l'interface et les procédures, on favorise l'instauration d'une interface prédictible dans laquelle l'utilisateur sait à l'avance le résultat.

La charge de travail

Définition : une interface homme-machine est qualifiée *d'efficace en charge de travail* si le volume de données à manipuler et d'actions à accomplir par unité de tâche est réduit. En effet, l'interaction est d'autant plus rapide que les actions de l'utilisateur portant sur un nombre limité de données sont courtes ; l'utilisateur est d'autant plus efficace lors de l'accomplissement de sa tâche qu'il est moins distrait par des informations étrangères à la tâche.

Objectif : ce critère remplit un rôle double : garder la charge de travail dans les limites de capacité des facultés humaines (particulièrement, la mémoire à court terme, la vision) et garantir une performance.

L'adaptabilité

Définition : une interface homme-machine est qualifiée *d'adaptable* (flexible) si elle possède la faculté de mimétisme comportemental vis-à-vis de son utilisateur. En effet, l'utilisateur est d'autant moins dérouté et acquerra d'autant plus d'expérience que l'interface peut s'adapter aux différents contextes de travail.

Objectif : le but est de fournir à l'utilisateur différentes voies pour accomplir sa tâche qui peuvent varier en fonction de différents paramètres.

Le contrôle de dialogue

Définition : une interface homme-machine est qualifiée d'interface *à contrôle explicite* si elle peut fournir à l'utilisateur l'apparence, l'illusion d'être placée sous contrôle de l'utilisateur en exécutant des actions suite aux demandes explicites de ce dernier.

Objectif: le but est de laisser l'utilisateur contrôler le déroulement du dialogue autant que possible, de compléter une action uniquement lorsque l'utilisateur en spécifie le but.

La représentativité

Définition : une interface homme-machine est qualifiée de *représentative* si les codes utilisés, les items de menu, les libellés facilitent l'encodage, la rétention.
Objectif : le but est de répandre l'usage de dénominations significatives au sein du dialogue.

Le guidage

Définition : une interface homme-machine est qualifiée d'*efficace en guidage* (ou en *feed-back)* si elle informe de manière constante l'utilisateur sur l'issue de ses actions et sur sa position dans l'accomplissement de sa tâche. En effet, l'utilisateur réalise sa tâche d'autant mieux qu'il est guidé à travers toutes les étapes nécessaires pour la mener à bien.

Objectif: le but est de fournir à l'utilisateur une aide sur ce qu'il peut entreprendre, sur la situation dans laquelle il se trouve et sur les résultats des actions effectuées ; on demande, en outre, de faire attention à la lisibilité.

La gestion des erreurs

Définition : une interface homme-machine est qualifiée d'*efficace en gestion des erreurs* si elle s'avère robuste aux erreurs commises par l'utilisateur et se montre conviviale dans la manière de les corriger. En effet, la performance de réalisation d'une tâche est d'autant meilleure que les occasions d'erreurs sont réduites.
Objectif : le but est d'éviter les erreurs autant que possible.

III.5 Quelques règles ergonomiques[3]

III.5.1 Règles à la sélection des menus.

- *La conception des menus peut être basée sur le modèle du mini-monde, c'est-à-dire en reflétant les options réelles de la tâche.*
- *La complexité des menus doit refléter le niveau de l'utilisateur, les fonctionnalités doivent refléter les besoins de la tâche ; aucune valeur par défaut ne peut être placée dans un menu.*
- *A chaque item de menu attaché à une action doit correspondre une unité de la tâche que l'utilisateur doit accomplir.*
- *Pour les menus affichés séparément des données sur l'écran, lorsque la sélection de menu s'effectue par pointage, le système doit placer le curseur automatiquement sur le premier item du menu.*
- *En toute généralité, l'attachement de chaque item de menu doit être approprié à la tâche*

  - soit un sous-menu,
  - soit un écran de saisie/affichage,
  - soit un écran secondaire,
  - soit le branchement ou le débranchement du paramètre associé à l'option ("Toggle item"),
  - soit une action déclenchable de l'application.

- *Les dénominations des items doivent de préférence être choisies par les utilisateurs.*
- *La présentation des menus (implicite ou explicite) doit être appropriée à la tâche.*
- *La présentation des items et des explications dans les menus doit être appropriée au niveau d'expérience de l'utilisateur.*
- *Les dénominations des items doivent correspondre univoquement à des unités de la tâche que l'utilisateur doit accomplir.*
- *La dénomination de l'item associé à l'option d'abandon du menu courant doit être appropriée, compréhensible, cohérente et suffisamment distincte de celle de l'option de clôture de la session interactive.*

III.5.2 Règles aux dénominations et abréviations.

- *Les dénominations des données affichées et de leurs libellés identificatifs doivent incorporer les termes familiers, clairs ou propres à la tâche de l'utilisateur et éviter les termes techniques appartenant à l'informatique.*

Lorsqu'un doute persiste, les dénominations affichées doivent être pré-testées avec des utilisateurs finals.

---

[3] Il s'agit d'en donner une ébauche. Nous n'avons pas l'intention de citer le maximum de ces innombrables règles [Van., 93].

- *Les dénominations des données affichées et de leurs libellés identificatifs doivent être sélectionnées avec soin et utilisées de manière cohérente.*
- *Les dénominations des données affichées doivent être choisies par les utilisateurs.*
- *Les dénominations des données affichées doivent être empiriquement optimisées pour les utilisateurs.*
- *Les abréviations et les acronymes ne devraient être utilisés que s'ils sont plus courts que l'item à abréger et qu'ils sont bien compris par l'utilisateur.*
- *L'option de saisie d'une abréviation à la place d'une donnée longue peut être prévue.*

On peut utiliser plusieurs techniques d'abréviations. Citons :

- l'approximation phonétique,
- la production naturelle,
- l'affectation à une touche spéciale suivie d'un mnémonique,
- l'abréviation par suppression de toutes les lettres peu importantes,
- la troncature normale, à deux lettres non ambiguës ou maximale,
- la contraction (suppression des voyelles).

- *La règle d'abréviation doit être simple :*

la troncature simple reste probablement la meilleure règle qui puisse être appliquée sans ambiguïté. En effet, elle est facilement expliquée à des utilisateurs de niveau débutant et novices ; elle fonctionne bien mieux que la contraction de mots ou l'omission de voyelles.

- *La règle d'abréviation doit être cohérente.*
- *La règle d'abréviation doit être familière à l'utilisateur.*

III.5.3 Règles à la sélection d'OIC.

- *Pour toute donnée, l'objet interactif qui est le plus facile à manipuler par l'utilisateur pour représenter cette donnée doit être sélectionné.*
- *La sélection des oia de saisie doit être intra-cohérente.*
- *Un choix simple ne peut pas se concrétiser par un ensemble de bouton-radios dans une liste de sélection.*
- *Les menus peuvent être présentés comme des objets de contrôle.*
- *Utiliser des boutons de commande ou des boutons graphiques pour déclencher des actions.*
- *A tout bouton de commande ou graphique doit être associée une et une seule action que l'utilisateur peut invoquer.*
- *Utiliser des boîtes à cocher pour activer/désactiver des options.*
- *Utiliser des bouton-radios pour des choix mutuellement exclusifs.*
- *Utiliser des cadrans pour saisir des valeurs numériques oscillant dans un intervalle bien défini.*

- *Utiliser des cadrans, des curseurs de défilement ou d'autres oia de même comportement pour contrôler la direction, la position, l'amplitude.*

III.5.4 Règles à la saisie, affichage et placement d'OIC.

- *La forme doit être compatible en saisie comme en affichage et avec le document source, papier.*
- *L'utilisateur doit explicitement exécuter la tabulation pour se déplacer de champ en champ.*
- *Lorsqu'une donnée calendrier doit être saisie, elle doit suivre les conventions de l'utilisateur.*
- *La position des oic doit être cohérente avec les conventions de l'utilisateur.*
- *La position des oic peut suivre la définition d'un standard d'affichage.*
- *La position des données affichées doit être cohérente.*
- *La position des données affichées doit être telle que les données soient facilement utilisables.*
- *La position de l'objet interactif concret relatif à la première donnée à saisir/afficher dans un micro-dialogue doit être située le plus en haut à gauche.*
- *Les oic associés à des données importantes, à retenir doivent être positionnés en premier lieu.*
- *Toutes les données intervenant dans une saisie doivent être justifiées automatiquement.*
- *Les oic doivent être positionnés en utilisant l'écran tout entier et non pas seulement sa partie gauche.*
- *Les oic d'un même oic composé devraient être justifiés inférieurement par groupes afin d'éviter l'empilement des libellés ou d'oic liés.*

- *Les titres d'écran, les en-têtes et les données d'un même oic composé devraient autant que possible être justifiés à gauche, équilibrés verticalement, équilibrés proportionnellement.*

III.6 Interface utilisateur de SLLParc.

Après avoir conçu et obtenu le schéma du modèle conceptuel de données, la tentation est grande de commencer directement la programmation.

Nous pensons qu'après cette étape, il convient de construire la maquette ou le prototype de l'application afin de simuler son fonctionnement. Cette maquette peut bien être sur du papier. Le plus grand avantage de cette pratique est de refaire parler l'utilisateur pour préciser certains besoins.

En utilisant cette technique, nous avons corrigé, ajusté et replacé plusieurs concepts expliqués partiellement par l'utilisateur dans l'étape d'analyse des besoins.

En ce qui concerne SLLParc, notre démarche a abouti par une séance « publique » de validation de l'interface que nous présentons en annexe de ce travail.

Si la maquette, à valider par l'utilisateur, se construit à l'aide du logiciel comme c'était le cas avec SLLParc, cela présuppose le choix du logiciel de développement. Nous y reviendrons.

Si c'était à refaire :

Je reprendrai la même démarche de recherche de validation de l'interface utilisateur par l'utilisateur. Le fait d'utiliser le logiciel de développement m'a permis de me familiariser au langage de programmation choisi. Les remarques de l'utilisateur peuvent faire l'objet de modification de la maquette voire du schéma conceptuel (Voir les remarques de la page 11).

III.7 Conclusion du chapitre.

Ce Chapitre nous a permis de survoler les concepts des IHM. L'ergonomie du logiciel est le centre de tout. En effet, un logiciel est écrit pour l'utilisateur et non pour soi-même.

Chapitre 4 : Théorie sur la méthodologie de développement du logiciel.

Sommaire

[1] SLLParc est le nom du logiciel que nous avons développé chez Swiss Life à Luxembourg.

Chapitre 4 : Théorie sur la méthodologie de développement du logiciel.

IV.1 Introduction.

Un logiciel est produit en suivant un processus comme tout produit manufacturé, mais le domaine du logiciel a des particularités. Lorsqu'un logiciel est réalisé, sa production se fait par copie et non une fabrication en série.

Un processus de développement du logiciel fait une large place à l'analyse, à la conception et à la validation.

Une autre particularité du développement du logiciel est l'invisibilité de celui-ci : il est difficile d'observer un logiciel en cours de développement et de se persuader que le processus se déroule correctement. D'où l'importance de la gestion de configuration[2].

Une fois développé, un logiciel est mis en exploitation. Se posent alors les problèmes de la maintenance, qu'il s'agisse de corriger des défauts, d'améliorer certaines caractéristiques, ou de suivre les évolutions des besoins ou de l'environnement.

La maintenance peut remettre en cause d'une manière significative les fonctions du système et implique de ce fait un processus de redéveloppement. C'est pourquoi, on parle du cycle de vie du logiciel quant on considère l'exploitation et la maintenance en plus du développement [Cfr. Fig. 1.3].

IV.2 Etapes et Modèles de développement du logiciel.

IV.2.1 Les étapes de développement du logiciel.

Quelque soit le modèle de développement (voir le point suivant IV.2.2) retenu pour développer un logiciel donné, les étapes ou activités suivantes sont plus au moins importantes. Dans certains cas, elles peuvent même être inutiles, par exemple, lorsque le logiciel à produire est une évolution d'un produit existant.

1. **Analyse des besoins**

Le but de cette activité est d'éviter de développer un logiciel non adéquat. On va donc étudier le domaine d'application, ainsi que l'état actuel de l'environnement du futur système afin d'en déterminer les frontières, le rôle, les ressources disponibles et requises, les contraintes d'utilisation et de performance, etc.

La particularité de cette activité est que ses données sont fournies par des experts du domaine d'application et par les futurs utilisateurs. L'équipe de

[2] La gestion de configuration est la gestion de document, de programme, de code,... d'un produit logiciel.

développement doit établir un dialogue avec ces spécialistes du domaine d'application qui ne sont pas forcément des informaticiens.

De ce fait les méthodes utilisées ne relèvent pas directement des techniques informatiques, mais plutôt des sciences cognitives : entretiens, questionnaires, observations de l'existant, études de situations similaires.

Le résultat de cette activité est un ensemble de documents décrivant les aspects pertinents de l'environnement du futur système, son rôle et sa future utilisation. Un manuel d'utilisation préliminaire est parfois produit.

Le partage entre logiciel et matériel, quand il ne va pas de soi, est défini, en liaison avec les études de faisabilité.

L'analyse des besoins est l'activité essentielle au début du processus de développement. Elle est alors menée en liaison avec les études de faisabilité et la planification. Cependant, elle se poursuit durant tout le processus de développement (car des questions relatives aux besoins et à l'environnement peuvent apparaître), et durant tout le cycle de vie du logiciel (maintenance évolutive).

2. Spécification globale

Le but de cette activité est d'établir une première description du futur système. Ses données sont les résultats de l'analyse des besoins ainsi que des considérations de technique et de faisabilité informatiques. Son résultat est une description de ce que doit faire le logiciel en évitant des décisions prématurées de réalisation (on dit *quoi,* on ne dit pas *comment).*

Cette description[3] va servir de point de départ au développement. Une première version du manuel de référence est parfois produite, ainsi que des compléments au manuel d'utilisation.

Cette activité est fortement corrélée avec l'analyse des besoins avec laquelle des échanges importants sont nécessaires : dans les modèles de développement présentés aux paragraphes suivants, ces activités sont souvent regroupées dans la même étape. Elle est aussi corrélée avec l'activité de validation.

Il est important de ne pas faire des choix d'implémentation prématurés lors de cette activité : il est trop difficile d'anticiper leurs conséquences sur la réalisation finale en termes de performances, ressources, ou même de faisabilité.

3. Conceptions architecturale et détaillée

L'activité de conception consiste à enrichir la description du logiciel de détails d'implémentation afin d'aboutir à une description très proche d'un programme. Elle se déroule souvent pendant deux étapes : l'étape de *conception architecturale* et l'étape de *conception détaillée.*

[3] C'est ce qu'on appelle spécification technique de besoins.

L'étape de conception architecturale a pour but de décomposer le logiciel en composants plus simples. On précise les interfaces et les fonctions de chaque composant. À l'issue de cette étape, on obtient une description de l'architecture du logiciel et un ensemble de spécifications de ses divers composants.
L'étape de conception détaillée fournit pour chaque composant une description de la manière dont les fonctions du composant sont réalisées : algorithmes, représentation des données.

La frontière entre l'activité de spécification et les activités de conception est souvent floue. En effet, il ne serait pas raisonnable de spécifier un système indépendamment de toute considération de faisabilité.
Le fait que la conception soit possible démontre la faisabilité : l'activité de conception commence souvent pendant la spécification, et peut la remettre en cause.
De plus, dans de nombreux contextes, il existe des contraintes de réalisation qui amènent à anticiper sur la conception au moment de la spécification.

4. Programmation

Cette activité consiste à passer du résultat de la conception détaillée à un ensemble de programmes ou de composants de programmes. Elle est la mieux maîtrisée et la mieux "outillée" : dans certains cas, elle est même automatisée.

Actuellement, l'activité de programmation ne représente que de 15 à 20% de l'effort de développement d'un logiciel, voire seulement 10% selon certains chiffres. La spécification et la conception représentent environ 40% de l'effort dans un projet bien conduit. La validation et la vérification représentent de l'ordre de 40% de l'effort total de développement.

5. Gestion de configuration et intégration

La *gestion de configuration* a pour but de permettre la gestion des composants du logiciel, d'en maîtriser l'évolution et les mises à jour tout au long du processus de développement.
Jusque récemment, cette activité n'était informatisée que pour les textes de programmes et les autres documents étaient gérés à part. Avec l'apparition d'environnements intégrés de développement[4], tous les documents relatifs au développement d'un logiciel peuvent être traités de manière homogène.

L'intégration consiste à assembler tout ou une partie des composants d'un logiciel pour obtenir un système exécutable. Cette activité utilise la gestion de configuration pour assembler des versions cohérentes de chaque composant.
Souvent il existe plusieurs choix possibles pour certains composants, cela correspond à des variantes du logiciel, en particulier pour des systèmes d'exploitation différents. Les activités de gestion de configuration et

[4] Nous citerons MS Visual Basic, Centura Builder,...

d'intégration permettent de gérer ces variantes.

6. Validation et vérification

Le problème de l'adéquation des résultats de l'analyse des besoins et de la spécification globale est délicat.

La question à poser est : « a-t-on décrit le "***bon***" système, c'est-à-dire un système qui répond à l'attente des utilisateurs et aux contraintes de leur environnement ? ».

L'activité qui a pour but de s'assurer que la réponse à cette question est satisfaisante s'appelle la ***validation.***

Par opposition, l'activité qui consiste à s'assurer que les descriptions successives du logiciel, et, *in fine,* le logiciel lui-même, satisfont la spécification globale s'appelle la *vérification.*

La question à laquelle on répond est le développement est-il *correct* par rapport à la spécification de départ ?

L'analyse des besoins et la spécification globale sont donc intrinsèquement liées à l'activité de validation. De même, les activités de conception et de programmation impliquent de la vérification.

La validation consiste essentiellement en des revues et inspections de spécifications ou de manuels, et du prototypage rapide.

La vérification inclut également des inspections de spécifications ou de programme, ainsi que de la preuve et du test.

Une preuve porte sur une spécification détaillée ou un programme et permet de prouver que celle-ci ou celui-ci satisfait bien la spécification de départ.

Le test consiste à rechercher des erreurs dans une spécification ou un programme. Cette recherche peut se faire par examen ou analyse du texte dans ce cas on parle de *test statique* , ou alors par des exécutions sur un sous-ensemble fini des données possibles, on parle alors de *test dynamique*[5].

7. Rôle de la maquette

La principale difficulté de l'activité de validation est due à l'imprécision des besoins et des caractéristiques du système à développer.

Ces besoins sont exprimés dans les termes du domaine d'application, et souvent ils ne sont même pas complètement exprimés.

La *maquette* ou *prototype rapide* peut aider à résoudre ce problème. Cette activité consiste à développer très rapidement un programme qui est une ébauche du futur système : il n'en a pas les performances, ni toutes les fonctionnalités, et ne répond pas aux exigences de qualité d'un produit fini.

Ce programme est ensuite soumis à des scénarios en liaison avec les futurs

[5] Nous parlerons du test au paragraphe IV.4.

utilisateurs afin de préciser leurs besoins ou leurs souhaits. On parle alors de *maquette exploratoire.*

Cette activité peut également intervenir lors d'une étape de conception des choix différents qui peuvent être expérimentés et comparés au moyen de maquettes. On parle alors de *maquette expérimentale.*
Il est important de bien définir les objectifs d'une opération de prototypage rapide et d'en tenir compte pour la conception de la maquette.

IV.2.2 Les modèles de développement du logiciel.

IV.2.2.1 Le modèle de la cascade.

Le principe du modèle de la cascade (ou modèle de la chute d'eau) est très simple : on convient d'avoir un certain nombre d'étapes (ou phases).

Une étape doit se terminer à une certaine date, par la production de certains documents ou logiciels.

Les résultats de l'étape sont soumis à une revue approfondie, et on ne passe à l'étape suivante que quand ils sont jugés satisfaisants.

Sur le schéma suivant du modèle en cascade, certaines étapes portent le nom d'une activité ; cela signifie que l'activité est essentielle pour cette étape, mais n'impose pas qu'elle n'ait lieu que dans cette étape.

De plus d'autres activités interviennent. Par exemple, le contrôle technique ou la gestion de configurations sont présents tout au long du processus.

L'interaction entre étapes et activités sert de base pour définir les résultats requis de chaque étape.

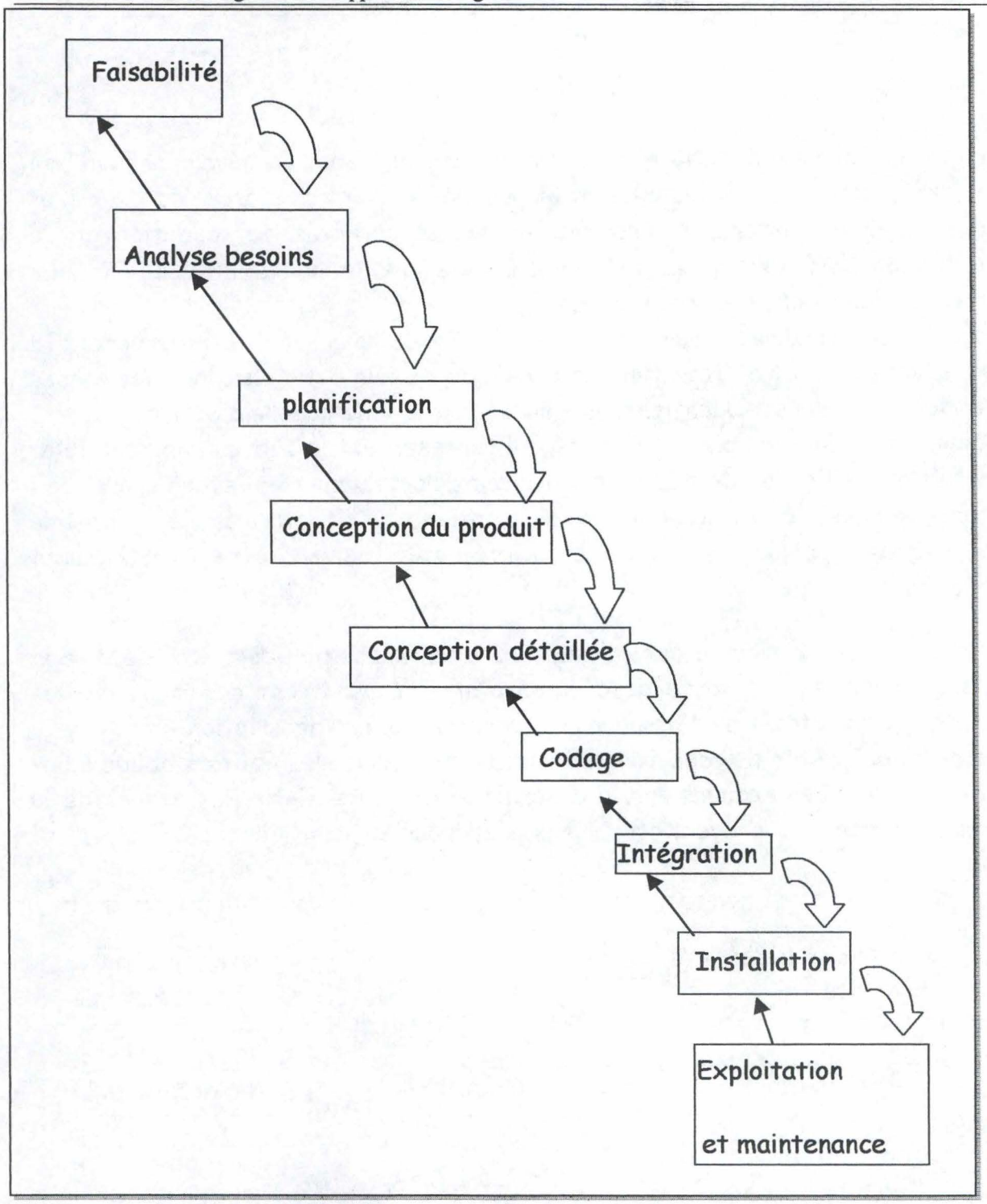


Fig. 4.1 *Modèle de la cascade*

Cependant ce modèle, séduisant par sa simplicité, est souvent abandonné au profit du modèle en V, plus récent, qui présente une approche plus réaliste de l'articulation entre les activités de réalisation et celles de validation - vérification.

IV.2.2.2 Le modèle en V.

Ce modèle rend explicite le fait que les premières étapes du développement, qui ont trait surtout à la construction du logiciel, doivent préparer les dernières étapes, qui font intervenir essentiellement les activités de validation et de vérification. La figure ci-dessous donne un exemple de modèle en V. Il y a deux sortes de dépendances entre étapes :

— celles matérialisées par des traits gras obliques, qui correspondent à l'enchaînement et à l'itération éventuelle du modèle de la cascade , les étapes se déroulent donc séquentiellement en suivant le V de gauche à droite ,

— celles matérialisées par les flèches, qui représentent le fait qu'une partie des résultats de l'étape de départ est utilisée directement par l'étape d'arrivée , par exemple, à l'issue de la conception architecturale, le protocole d'intégration et les jeux de test d'intégration doivent être complètement décrits.

Le principe de ce modèle est qu'avec toute décomposition doit être décrite la recomposition, et que toute description d'un composant est accompagnée des tests qui permettront de s'assurer qu'il correspond à sa description.

De plus l'obligation de concevoir les jeux de test et leurs résultats oblige à une réflexion et à des retours sur la description en cours. Enfin, les étapes de la branche droite du V peuvent être mieux préparées et planifiées.

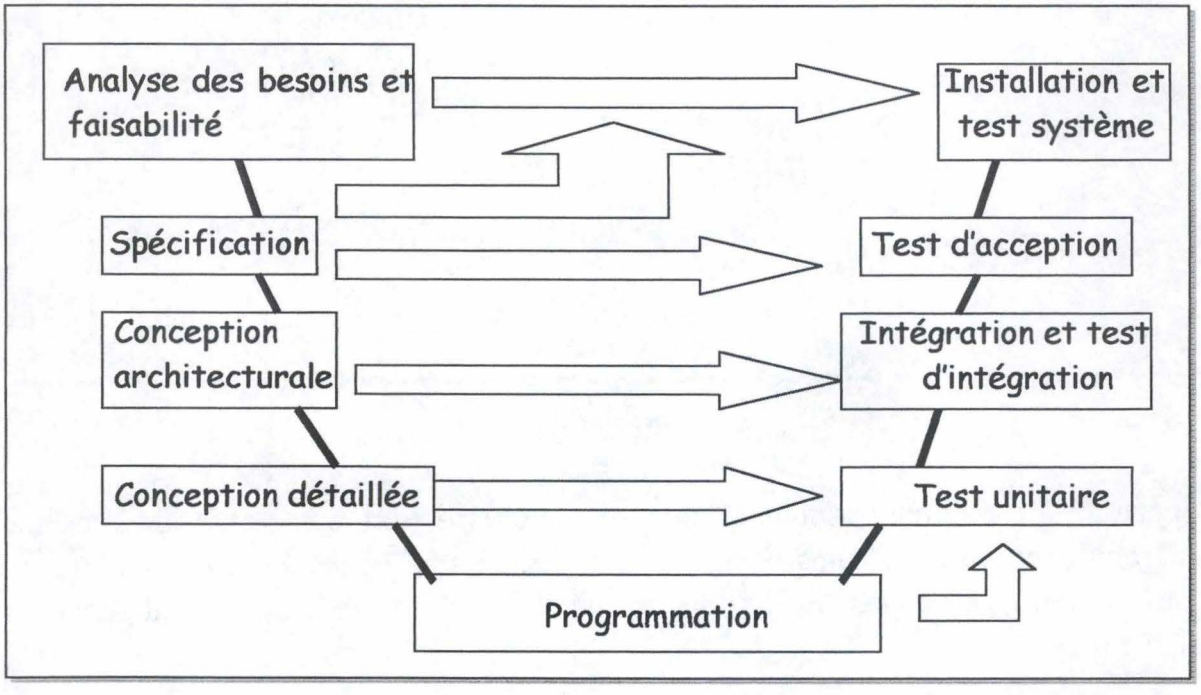


Fig. 4.2 *Modèle en V.*

IV.2.2.3 Le modèle en spirale.

Ce modèle, proposé par B. Boehm en 1988, est beaucoup plus général que les précédents et peut les inclure.

Il met l'accent sur une activité particulière, l'analyse de risques : chaque cycle de la spirale, se déroule en quatre phases représentées par des quadrants :

1. détermination des objectifs du cycle, des alternatives pour les atteindre, des contraintes, à partir des résultats des cycles précédents, ou, si il n'y en a pas, d'une analyse préliminaire des besoins.

2. analyse des risques, évaluation des alternatives, éventuellement maquette ;

3. développement et vérification de la solution retenue ;

4. revue des résultats et planification du cycle suivant.

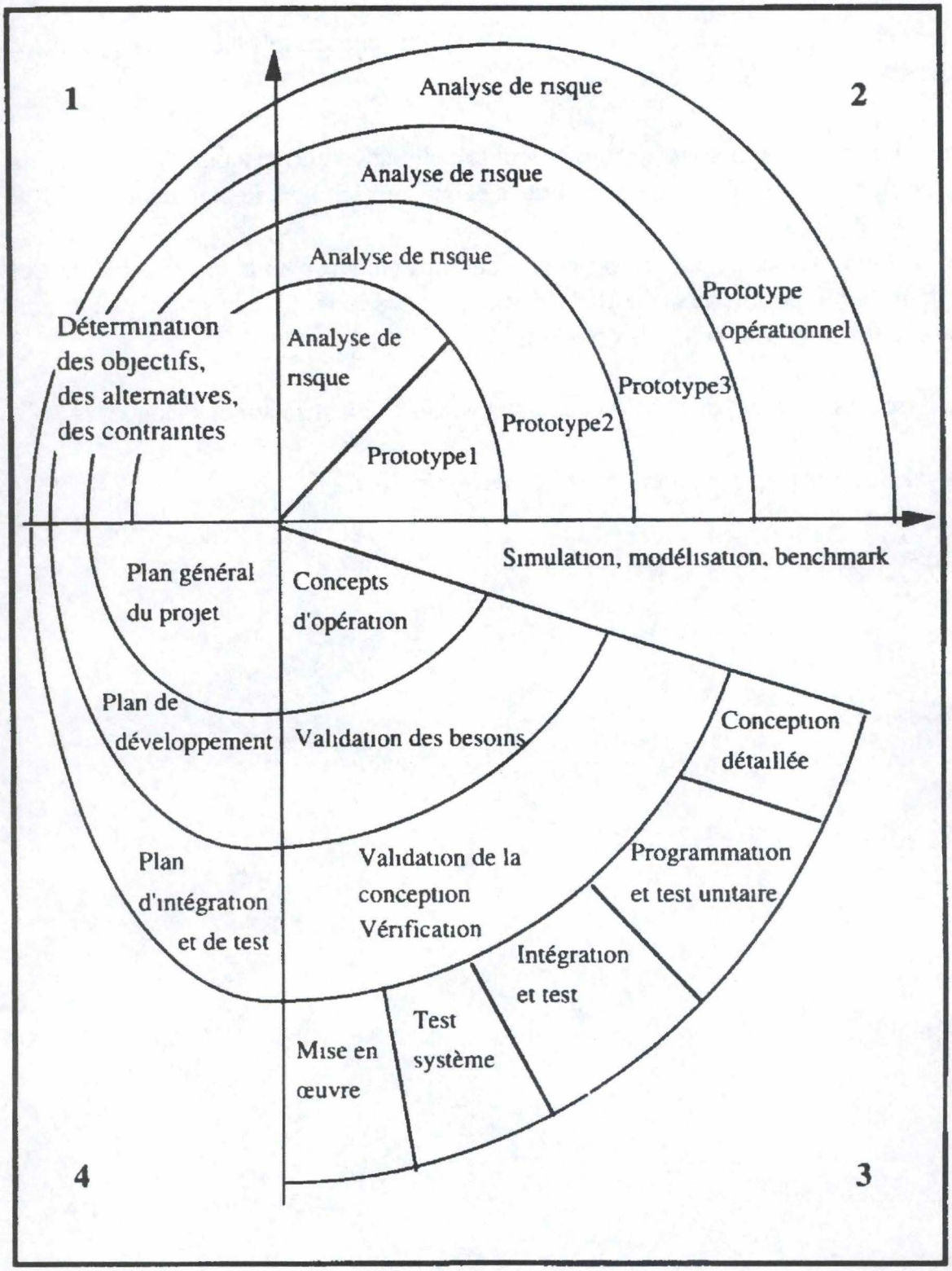


Fig. 4.3 Modèle en spirale

L'originalité de ce "super" modèle est d'encadrer le développement proprement dit par des phases consacrées à la détermination des objectifs et à l'analyse de risque.

Les risques majeurs du développement de logiciel

Un des intérêts du modèle en spirale est de fournir des listes des risques encourus lors d'un développement de logiciel et de suggérer des solutions.

Voici la liste des dix risques majeurs et, pour chacun, des mots-clés indiquent des solutions à envisager. Cette liste est donnée à titre d'exemple et de point de départ d'une analyse de risque : elle doit être adaptée, bien sûr, à chaque contexte.

- *Défaillance de personnel :* embauche de personnel de haut niveau ; adéquation entre profil et fonction ; esprit d'équipe , formation mutuelle ; personnes clés.
- *Calendrier et budget irréalistes* : estimation détaillée des coûts et calendriers ; développement incrémental ; réutilisation ; élagage des besoins.
- *Développement de fonctions inappropriées :* analyse de l'organisation ; analyse de la mission ; formulation des concepts opérationnels ; revues d'utilisateurs : manuel d'utilisation précoce.
- *Développement d'interfaces utilisateurs inappropriées :* maquette ; scénarios et revues d'utilisateurs ; analyse des tâches.
- *Produit "plaqué or" :* élagage des besoins ; maquette ; analyse des coûts-bénéfices ; conception prenant en compte les coûts.
- *Volatilité des besoins :* seuil élevé de modification ; masquage d'information ; développement incrémental où les derniers incréments sont les plus changeants.
- *Composants externes manquants :* inspections : essais et mesures ; analyse de compatibilité.
- *Tâches externes défaillantes :* audit avant attribution de sous-traitance ; contrats avec bonus ; revues.
- *Problèmes de performances :* simulations ; modélisations ; essais et mesures ; maquette.
- *Exigences démesurées par rapport à la technologie :* analyses techniques de faisabilité ; maquette.

IV.2.2.4 Le modèle par incrément.

Dans les modèles présentés jusqu'ici, il est implicite qu'après une décomposition en composants (qui résulte de la conception architecturale), ces composants sont développés indépendamment les uns des autres, en parallèle ou en séquence, selon les ressources disponibles.

Dans les modèles par incréments, un seul sous-ensemble des composants est developpé à la fois: un *logiciel noyau* est tout d'abord développé puis des *incréments* sont successivement développés et intégrés.

Le processus de développement de chaque incrément est un des processus classiques.

Les avantages de ce type de modèle sont nombreux :

— chaque développement est moins complexe ;
— les intégrations sont progressives ;
— il peut y avoir des livraisons et des mises en service après chaque intégration d'incrément.

Cette approche est souvent utilisée pour de grands projets, fonctionnant par appels d'offres et sous-traitances.

Ce modèle permet également de mieux lisser dans le temps l'effort de développement et les effectifs. En effet, dans les modèles de la cascade ou en V, cet effort et ces effectifs augmentent beaucoup au moment de la spécification détaillée pour diminuer significativement au moment du test d'intégration. De plus, les compétences nécessaires varient beaucoup selon les étapes.

Dans la plupart des modèles par incréments, les développements se recouvrent comme dans la figure (Fig. 4.4) ci-après, ce qui permet d'atténuer ce problème.

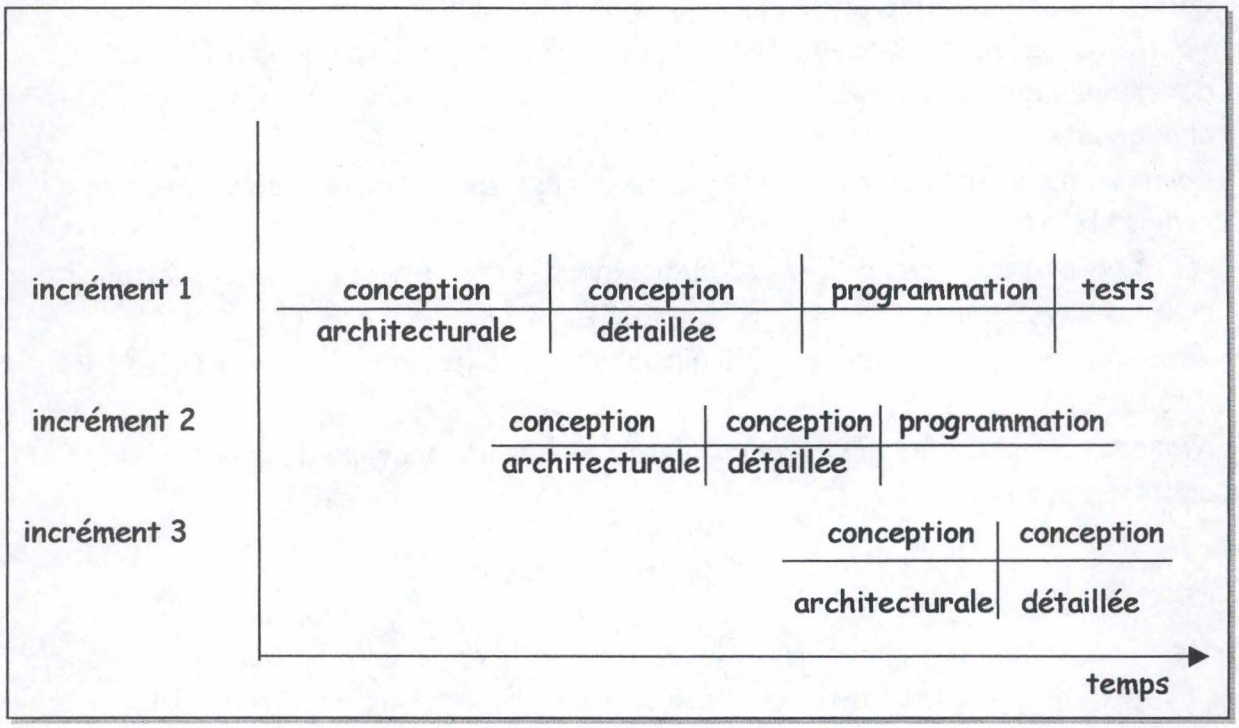


Fig. 4.4 *Modèle par incrément.*

IV.3 Techniques de spécification et méthodes d'analyse et de conception.

Les méthodes associées aux activités d'analyse et de conception des logiciels se sont développées pour répondre à l'évolution des matériels, des systèmes, des langages de programmation, et surtout à la complexité toujours croissante des logiciels. Elles reposent sur un ensemble de techniques qui sont souvent communes à ces activités.

La première partie de ce paragraphe présente les différentes techniques de spécification, et la deuxième partie a pour but de citer les méthodes d'analyse et de conception les plus répandues actuellement.

IV.3.1 Différentes techniques de spécification

IV.3.1.1 Les énoncés informels

Les spécifications écrites uniquement en langage naturel, même si elles respectent des plans types[6] ou des plans propres à une entreprise ou à un projet, posent en général des problèmes de non cohérence, d'ambiguïté, de non complétude, de difficulté d'organisation avec risque de redondance des informations[7] surtout dans le cas des systèmes de logiciels complexes.

C'est la raison pour laquelle, différents outils ont été développés permettant de résoudre tout ou partie des problèmes mentionnés ci-dessus.

IV.3.1.2 Les présentations formatées

- **Le dictionnaire des données ou le glossaire**

Il est constitué de l'ensemble des spécifications des données utilisées aux différents niveaux d'analyse et de conception , il contient en général les définitions des termes utilisés classées par ordre alphabétique ; il présente des sigles, des codes ou des symboles employés dans les documents ; précise les synonymes, les alias,... Il peut permettre aussi de définir la structure d'une donnée composée avec une notation syntaxique stricte de forme Backus-Naur.

- **La table de décision**

Est une représentation tabulaire de tous les cas des valeurs d'entrée d'un processus et des valeurs de sortie correspondant à chacune de ces combinaisons.

- **La table état-transition**

Est particulièrement adapté à la spécification des systèmes dont les sorties sont déterminées par les entrées et l'historique des états antérieures.

---

[6] Comme le standard STD 830, DoD 2167 – A.

[7] Cfr. les sept péchés capitaux du spécificateur dans [Bod, 94].

IV.3.1.3 Les techniques graphiques ou semi-formelles

- **Le modèle entité-association**

Ce modèle le plus répandu a été traité dans les deux premiers chapitres de notre présent travail.

- **Les diagrammes de flots (flux) de données**

Intégré dans diverses méthodes, les diagrammes de flots de données sont utilisées pour modéliser les traitements. Ils permettent de montrer comment chaque processus transforme ses entrés successives en sorties correspondantes.

- **Les diagrammes de structures**

Ils permettent de décrire l'architecture d'un système, comme une hiérachie des fonctions et la présentent sous forme d'une structure arborescente, à lire de gauche à droite.

- **Le diagrammes états – transitions**

Ils permettent de matérialiser l'incidence des événements sur les différents états du système en indiquant les actions à effectuer.

- **Les réseaux de Pétri**

Est un outil mathématique très général permettant de modéliser le comportement d'un système dynamique à événements discrets.

IV.3.1.4 Les spécifications algébriques

Les spécifications algébriques sont des spécifications formelles, c'est-à-dire elles sont écrites suivant une syntaxe bien définie comme celle d'un langage de programmation. Et la syntaxe est accompagnée d'une sémantique rigoureuse qui définit des modèles mathématiques représentants les réalisations acceptables de chaque spécification syntaxiquement correcte.

IV.3.2 Méthodes d'analyse et de conception

IV.3.2.1 Les méthodes fonctionnelles :

- **La méthode SADT**[8] : sert à communiquer les problèmes.
- **L'Analyse structurée** : raffinement successifs des traitements ou processus.
- **La conception structurée** : décomposition fonctionnelle de l'analyse structurée en précisant les liens et paramètres entre différents modules.
- **L'analyse et la conception « temps réel »**.

[8] Structured Analysis and Design Technique.

IV.3.2.2 Les méthodes orientées objets.

IV.3.2.3 La méthode Merise : Les deux premiers chapitres de ce mémoire s'y sont consacrés avec la coloration spécifique de l'Institut d'Informatique de Namur.

IV.4 Test et Preuve de logiciel.

IV.4.1 Test du logiciel.

1. Définition du test de logiciel

L'objectif du test est de détecter des fautes ou des inadéquations d'un logiciel. Il est nécessaire de préciser par rapport à quelles références on tente de dégager ces inadéquations : les spécifications du logiciel, les normes ou règles portant sur son code ou les documents le concernant. Le test de logiciel est donc défini ici avec un sens assez large : la recherche d'inadéquations d'un logiciel par rapport à des références établies.

Le test ne peut établir que la *présence* d'erreurs ; détecte les symptômes et pas les causes. Le test de logiciel est la démarche la plus répandue.

2. Classification de méthodes

Il existe deux grandes classes de méthodes de test :

— les méthodes de *test statique* qui traitent le texte du logiciel sans exécuter ce logiciel sur des données réelles , elles peuvent être appliquées à des textes de spécifications ou de programme ;

— les méthodes de *test dynamique* qui reposent sur l'exécution effective du logiciel sur un sous-ensemble bien choisi du domaine de ses entrées possibles.

Les principales méthodes de test statique sont :

— les *lectures croisées* et *l'inspection,* c'est-à-dire la vérification "collégiale" d'un document (programme ou spécification du logiciel) ;

— l'*analyse d'anomalies,* telles que typage impropre, incohérence des interfaces de modules, portions de code isolées, mauvais usages de variables, etc., qui sont assistées par des outils ,

— *dévaluation symbolique* qui simule l'exécution du programme sur des données symboliques ; on obtient ainsi des expressions symboliques correspondant au texte des programmes.

Ces méthodes de test statique ne doivent jamais être négligées , ceci est particulièrement important pour le test de logiciels mettant en jeu du parallélisme, où le test dynamique est parfois peu applicable en raison du non-déterminisme des résultats obtenus par le test dynamique.

Le test dynamique repose sur quatre activités :

— la *sélection* qui consiste à choisir judicieusement un sous-ensemble des entrées possibles du logiciel, appelé *jeu de tests ;*

— la *soumission* du jeu de tests afin d'exécuter les tests retenus lors de l'étape précédente ,

— le dépouillement des résultats qui consiste à décider du succès ou de l'échec du jeu de tests ; se pose alors le problème dit de *l'oracle* car la décision n'est pas toujours simple, ni même possible ;

— l'évaluation de la qualité des tests effectués, qui est un des éléments pour la décision *d'arrêt du test.*

C'est sur la façon de mener la première étape que l'on classifie les diverses méthodes de test dynamique selon trois approches. Un testeur dispose principalement de trois types d'informations pertinentes à propos du (morceau de) logiciel à tester les spécifications, les textes de programmes et les caractéristiques du domaine d'entrée :

— s'il sélectionne un jeu de tests en se fondant sur les textes et la structure des programmes, il suit une méthode dite de *test structurel* (parfois aussi appelé test boîte transparente, en anglais *white or glass-box testing)*

— s'il sélectionne un jeu de tests en se fondant sur les spécifications, il suit une méthode dite de *test fonctionnel* (souvent appelé test boîte noire , en anglais *black-box testing :* on ne regarde pas l'intérieur de la boite, comme les textes des programmes du logiciel),

— il est enfin aussi possible d'appliquer des méthodes de test *aléatoire* ou *statistique,* pour lesquelles les documents relatifs au logiciel en test servent à établir le domaine des entrées possibles, voire à en dégager plusieurs sous-domaines, on utilise ensuite les fonctions de répartitions statistiques de ces entrées pour sélectionner un jeu de tests.

Autres types de tests :

- Test de performance ;
- Test de sécurité ;
- Test de convivialité - de documentation ;
- β-testing ;
- Test back to back : comparer deux versions ;
- Stress testing : pousser le système au delà de ces limites pour détecter des erreurs cachées, pour vérifier un atterrissage contrôlé (fail soft).

IV.4.2 Preuve du logiciel.

Le test n'est pas la seule méthode utilisable pour les activités de validation et de vérification. On peut aussi faire des preuves : celles-ci peuvent servir à démontrer sur une spécification qu'une certaine situation se produira toujours ou ne se produira jamais, on fait alors de la *validation.*

Plus généralement, on peut vouloir prouver qu'un programme, ou une spécification détaillée, satisfait une propriété apparaissant dans sa spécification de départ, on fait alors de la *vérification.*

La preuve est la seule démarche pour établir l'absence d'erreurs. C'est une démarche peu répandue vu son coût et des problèmes de communication.

1. Rappel sur les preuves

Une approche plus formelle de la notion de preuve repose sur la notion de *système formel.* Un système formel est la donnée :

— d'un langage de *formules,*
— d'un sous-ensemble de ces formules, les *axiomes,* qui sont vrais par définition,
— d'un ensemble de règles de déductions, appelées *règles d'inférence,* qui formalisent les étapes élémentaires licites de raisonnement.

2. Que peut-on prouver ?

Dans le cadre du développement d'un logiciel, la preuve formelle peut être utilisée à différentes fins. Elle ne peut être utilisée que si le développement est basé sur une méthode formelle car on a vu qu'il faut, au départ, un système formel.

On peut vouloir prouver la correction d'une étape de développement ou on peut vouloir prouver une propriété locale à un document de spécification ou de développement.

La figure suivante montre où peuvent intervenir des preuves de correction d'une étape de développement dans le cas d'un développement formel. Pour simplifier la figure, on a supposé qu'il y a une seule étape de conception. Les boîtes représentent des documents, et les flèches des activités.

Les formes des boîtes sont significatives : les besoins des utilisateurs, qui résultent de l'activité d'analyse des besoins, sont souvent exprimés en langage naturel, avec les risques d'imprécision que l'on sait ; les documents de spécification sont écrits dans un langage de spécification formelle , le programme est écrit dans un langage de programmation qui doit avoir été défini formellement.

Les flèches du bas correspondent aux activités de validation et de vérification on confronte le $n^{ième}$ document au $n\text{-}1^{ième}$. Une preuve d'une étape de développement n'est possible que si les documents de départ et d'arrivée sont

formels. Si ceux-ci sont écrits dans le même langage, on parle d'une preuve mono-langage , si par contre ils sont écrits dans deux langages différents, on parle de preuve bi-langage.

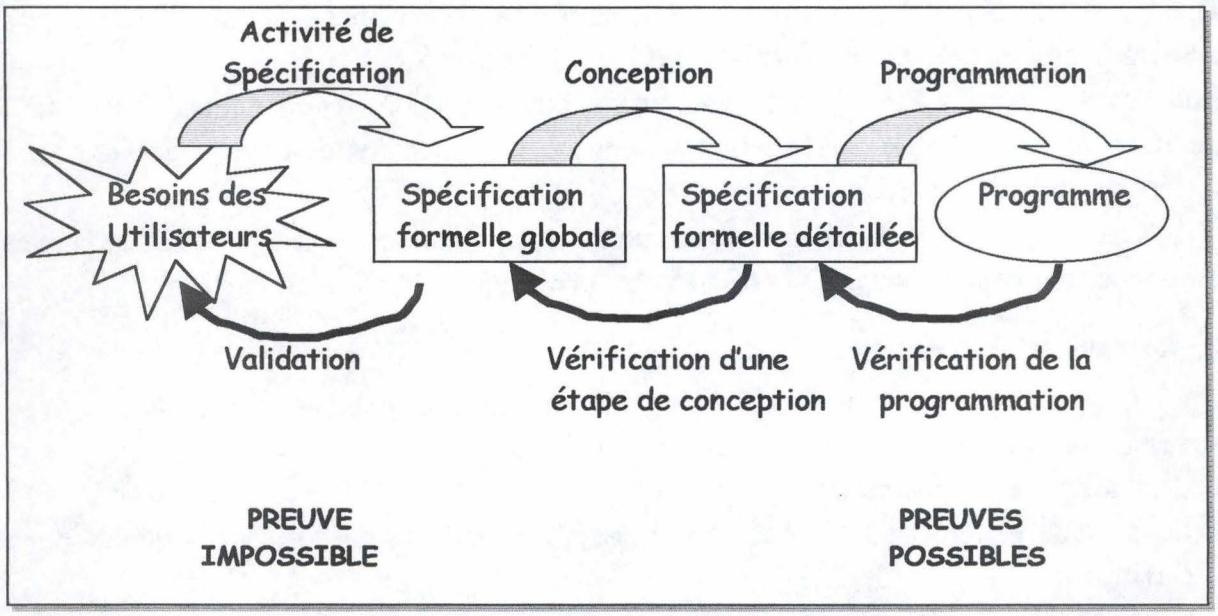


Fig. 4.5 Preuves de conception et preuves de programme.

On voit sur cette figure trois cas possibles :

— les besoins des utilisateurs étant exprimés informellement, on ne peut prouver la correction de la spécification formelle globale par rapport à eux,

— la preuve de l'étape de conception revient à prouver la correction de la spécification détaillée par rapport à la spécification globale , si le même langage est utilisé pour la spécification globale et la spécification détaillée, la preuve sera mono-langage : il s'agit alors d'une preuve de raffinement,

— la preuve de l'étape de programmation doit démontrer la correction du programme par rapport à la spécification détaillée, c'est généralement une preuve bi-langage.

En plus des preuves de vérification d'une étape de développement, on peut faire des preuves sur une spécification ou un programme, indépendamment du document qui le précède dans le développement.

IV.5 Méthode générale de développement d'une application.

Maintenant, nous pouvons résumer la méthode générale de conception et de développement d'une application. Supposons que sont déjà effectués :

- l'étude d'opportunité (cahier des charges) ;
- la conception du système d'information:
  - lister les résultats souhaités,

- relever les données élémentaires (dictionnaire des données),
- noter les contraintes sur les données.

Nous allons alors considérer trois étapes principales :

1. Analyse sémantique: (modélisation des données)

- repérer les Entités et les Associations.
- attribuer à chaque Entité un identifiant.
- placer les propriétés dans les Entités (attributs).
- faire un diagramme Entité/Association, avec les cardinalités[9].
- décrire des contraintes d'intégrité.
- valider: vérifier si le modèle est capable de répondre convenablement aux principales requêtes. Sinon, vérifier s'il ne manque pas une Entité ou une Association.

2. Analyse structurelle: (passage au logique)

- choisir un SGBD adapté au problème (en général c'est le SGBDR).
- implémenter le diagramme E/A pour le SGBD choisi (par exemple en relationnel, transformer le modèle en fichiers et choisir les champs communs en réseau, choisir les pointeurs).
- choisir les différents champs des fichiers et leur type (en appliquant par exemple la méthode des formes normales).
- choisir l'organisation des fichiers (index etc.).
- choisir une bonne codification (clés significatives, etc.).

3. Réalisation de l'application:

- choisir un langage de programmation.
- réaliser les fonctions demandées (en programmation , en mode direct, en SQL, etc.) en découpant le plus possible en unités modulaires.
- définir éventuellement des menus pour rendre l'application plus conviviale. Valider la maquette.
- faire les états de sortie : formulaires, étiquettes, factures, rapports...
- choisir les périphériques : écran, imprimante, etc.
- Vérification et validation de l'application.

[9] Ou connectivité.

IV.6 Cas de SLLParc.

Dans notre rapport de stage [Mif. 00], nous avons planifié notre travail en cinq grandes étapes que voici :

1. Etude de l'existant (différentes fiches des inventaires). Cette source d'information nous permettra une analyse de besoins dans un contexte précis.

2. Entretien avec les membres du Service Informatique afin d'en dégager les besoins précis.

3. Réalisation du modèle conceptuel de données. Cette modélisation sera complétée et modifiée aux besoins de l'utilisateur. Génération du modèle physique de données. Constitution d'un petit dossier d'analyse.

4. Etude et familiarisation au langage de programmation choisi pour implémenter cette application.

5. La programmation ou le développement de l'application (les interfaces, les tables, les fonctionnalités prévues et attendues et les tests).

De ce planning ressortent les étapes du développement du logiciel citées ci-haut. Nous les clarifieront dans le chapitre suivant.

IV.7 Conclusion du chapitre.

Nous avons compris qu'un logiciel est produit en suivant un processus. Le point IV.5 nous a donné les grandes lignes du développement du logiciel. Nous allons appliquer ce canevas pour un cas spécifique : gestion d'un parc informatique d'une entreprise. C'est l'objet du chapitre suivant.

Partie II

LA CONCEPTION ET LA REALISATION D'UN LOGICIEL

Dans cette partie, nous parlons de la réalisation du développement d'un logiciel. Le logiciel s'appelle SLLParc et sert à la gestion du parc informatique à Swiss Life à Luxembourg.

Elle est composée d'un chapitre :

Chapitre 5 : Conception et réalisation de SLLParc de Swiss Life Luxembourg.

Chapitre 5 : Conception et Réalisation de SLLParc[1] de Swiss Life Luxembourg.

Sommaire

[1] Nom donné à l'application développée pour gérer le parc informatique de Swiss Life Luxembourg.

Chapitre 5 : Conception et réalisation de SLLParc de Swiss Life Luxembourg.

V.1 Introduction.

Ce cas pratique de la conception et la réalisation de cette application est soutenu par la longue théorie que nous avons rappelée dans les chapitres précédents de la première partie. Nous allons donc, dans ce chapitre, suivre plus au moins les points que nous avons expliqués dans les chapitres théoriques.

Ce chapitre est le reflet de notre rapport de stage effectué chez Swiss Life de Septembre à Décembre 1999 [Mif,00]. Plusieurs autres documents s'y rapportant sont repris en annexe de ce mémoire.

Enfin, ce chapitre est la présentation du travail que nous avons effectué lors du stage. Travail pour lequel nous avons obtenu des résultats escomptés. La pratique tient surtout compte de tout le contexte et l'environnement dans lesquels le travail est fait.

V.2 Analyse fonctionnelle dans la conception et la réalisation de SLLParc.

V.2.1 Conception du système d'information

V.2.1.1 Besoins du service informatique de Swiss Life Luxembourg.

Swiss Life est une multinationale suisse d'assurance-vie. Sa branche luxembourgeoise est une organisation d'une cinquantaine de travailleurs. Son service Informatique est dirigé par Monsieur Benno Leguina secondé par quatre informaticiens. Le parc informatique possède des ordinateurs personnels (fixes et portable), des serveurs, des imprimantes, etc.

Voici dans l'encart suivant, les besoins exprimés par le service Informatique désirant avoir une application pour gérer son parc informatique.
Le contenu de ce fax exprime les grandes lignes du réel perçu dans ce domaine.

Veuillez trouver ci-joint un petit descriptif de la proposition de stage disponible actuellement chez Swiss Life :
Durée : 3 ou 4 mois selon disponibilités de l'étudiant.
Début : 1 Août 1999 ou 1 Septembre 1999.
Conditions : Participation aux frais avec contrat de durée déterminé sous le statut de «stagiaire ».
Localisation : L'ensemble de la prestation sera fait à Luxembourg dans nos locaux.

Titre: «Participation à la réalisation d'un programme de suivi et maintenance d'un parc informatique ».
Contenu : L'application devra gérer un inventaire de tout le matériel informatique de la société notamment :

- Les software installés avec suivi de la version, paramétrage d'installation, nombre de licences et droits de l'utilisateur.
- Les fournitures : stock, disponibilité, lieu de rangement et consommation par durée.
- Les machines et autres hardware : numéro de série, durée d'amortissement, caractéristiques techniques de l'équipement, version bios, historique d'affectation du matériel, documentation du software installé, notices d'installation, etc.
- Les utilisateurs : droits d'accès, matériel mise à disposition, formations (informatiques) suivis depuis l'arrivée à la société, adresses e-mail, etc.
- Les fichiers de données : affectation, droits d'accès par service et par individu, structuration du "file server".

Le programme devra être réalisé dans un langage client - serveur disponible dans la société (Centura - Sql Windows), et devra modéliser les données à l'aide de AMCD.

But :

On considérera le stage réussi si un exécutable est produit avec une gestion minimale de ces actifs cette version devrait au moins comprendre les trois ou quatre écrans nécessaires à la gestion des composants hardware.

Mémoire :

Un sujet de mémoire pourrait être développé entre autres au tour de l'organisation et gestion d'un parc informatique, des outils de

développement dans un environnement client-serveur ou encore dans la modélisation des standards de documentation.

Dans l'attente de vos nouvelles, je vous prie d'agréer, Monsieur Ramaekers, à l'assurance de ma considération distinguée.

Swiss Life (Luxembourg) S.A.

L.BERG B.LEGUINA

Directeur Technique Responsable Service Informatique

Fig. 5.1 Descriptif de la proposition de stage à Swiss Life Luxembourg.

V.2.1.2 Premier entretien.

Le Service Informatique de Swiss Life Luxembourg désire donc informatiser la gestion et faire le suivi de son parc informatique.

Il s'agit de gérer les PC (postes fixes et portables), les écrans, les serveurs, les programmes et les applications installés sur ces machines, les utilisateurs, leurs formations informatiques, leurs droits d'accès et enfin les consommables.

Cette gestion permettra essentiellement l'inventaire, le suivi et l'historique des équipements informatiques de la société.

En dehors des fonctionnalités de base de l'application (ajouts, consultations, mise à jour des données), quelques autres résultats spécifiques sont attendus. Entre autres:

- Signalétique des utilisateurs, leurs matériels et logiciels installés ;
- Suivi de la maintenance d'un équipement ;
- Historique d'un équipement ;
- Gestion des consommables, etc.

Nous disposons comme seule source physique d'information des diverses et multiples fiches des inventaires du matériel informatique de la société que nous présentons en annexe 1.

De ce premier contact avec l'utilisateur[2], nous avons défini le projet

[2] Par utilisateur ici, nous désignons le service informatique de Swiss Life Luxembourg.

cadre et avons établi un planning pour canaliser notre travail connaissant dès lors ses besoins immédiats.

Pour pouvoir atteindre l'objectif dans le délai, de la réalisation de ce programme, nous avons proposé de suivre le planning suivant :

1. Etude de l'existant (différentes fiches des inventaires). Cette source d'information nous permettra une analyse de besoins dans un contexte précis.
2. Entretien avec les membres du Service Informatique afin d'en dégager les besoins précis.
3. Réalisation du modèle conceptuel de données. Cette modélisation sera complétée et modifiée aux besoins de l'utilisateur. Génération du modèle physique de données.
4. Etude et familiarisation au langage de programmation choisi pour implémenter cette application.
5. La programmation ou le développement de l'application (l'interface utilisateur, la base de données, les fonctionnalités prévues et attendues et les tests).

V.2.1.3 Etude de l'existant.

La gestion du parc informatique de Swiss Life Luxembourg se faisait jusque-là à l'aide des formulaires. A l'annexe 1 de ce mémoire, le lecteur peut consulter quelques uns de ces documents.

Il va sans dire que le suivi, la mise à jour de ces formulaires n'ont pas été aisés d'autant plus que le parc informatique s'accroît et que le service informatique doit répondre aux besoins urgents et s'occuper des affaires courantes.

L'informatisation de cette gestion s'imposait donc. Ainsi, dans ce projet de développement, il nous est demandé de remplacer cette gestion manuelle par un outil informatique de gestion.

Outre ces formulaires, les entretiens fréquents avec l'utilisateur ont été sources incontournables d'information pour le besoin de notre analyse.

Après cela, nous pouvons passer à l'étape de la modélisation des données. Le schéma du modèle conceptuel des données en est la finalité.

V.2.2 Modélisation des données.

V.2.2.1 Modèle Conceptuel de Données de base.

Comme nous le savons, plusieurs schémas peuvent être construits pour une seule application. Mais peu d'entr'eux répondront exactement aux besoins de l'utilisateur et du programmeur. Pour plusieurs raisons, le schéma conceptuel de base sera modifié et adapté conservant principalement la sémantique de l'application.

En ce qui concerne SLLParc, le schéma initial présentait l'héritage pour le type d'entité Hardware. Des modifications et choix ont permis d'adopter le schéma suivant comme schéma de base.

VEUILLEZ CONSULTER LE PREMIER SCHEMA DE L'ANNEXE 2[3].

V.2.2.2 Adaptation du Modèle Conceptuel de Données.

Du modèle conceptuel de données de base ci-haut, les modifications ci-après ont été apportées :

1. L'utilisateur sera identifié par son Login.
2. Le software sera identifié par son nom.
3. Le consommable sera identifié par sa catégorie et sa référence.
4. Le leasing et la licence seront gérés indépendamment.
5. Le code d'un emplacement possède déjà son site.

Nous repensons ce qu'est une affectation dans ce cas précis.

En effet, cette opération se fait, il est vrai entre le software, le hardware, l'utilisateur et le lieu de travail. L'index qui rend ces quatre champs obligatoires est aboli car ce n'est pas ce que je souhaite en pratique. En effet, je peux bien affecter un lieu de travail à un utilisateur ou seulement un hardware à un utilisateur.

D'où l'idée de dire que ces champs peuvent être nuls (en affectant deux entités, les deux autres restent vides). Cette idée permettra des champs vides dans un fichier qui par conception gonfle exponentiellement.

Une façon plus pratique de modéliser l'affectation est de scinder ce fichier en plus petits reliant ces tables deux à deux. Nous augmentons le nombre d'entité quasi identique mais nous « gagnons » pratiquement. Cette modification quoique majeure ne change pas le besoin de l'utilisateur.

En dehors de ceci, quelques autres modifications mineures ont été

[3] Ceci pour ne pas faire double emploi.

faites. Voici au paragraphe ci-dessous, le modèle conceptuel validé qui nous a servi au développement.

V.2.2.3 Modèle Conceptuel de Données validé.

Après que l'utilisateur ait validé le schéma (règles de complétude et de cohérence), le schéma du modèle conceptuel de données suivant servira au développement de SLLParc.

VEUILLEZ CONSULTER LE DEUXIEME SCHEMA DE L'ANNEXE 2.

V.2.3 Modèle Logique de données

VEUILLEZ CONSULTER LE TROISIEME SCHEMA DE L'ANNEXE 2.

V.2.4 Modèle physique de données.

Le schéma physique d'une base de données est constitué de son schéma logique complété des constructions et paramètres techniques qui le rendent opérationnel. Du schéma logique ci-haut, on ajouterait des espaces de stockage, des clés d'accès, des modes de stockage des enregistrements.

V.3 Base de données dans la conception et la réalisation de SLLParc.

V.3.1 Conception de la base de données

V.3.1.1 Considérations théoriques.

Par rapport aux concepts de base de données évoqués au point II.2, la base de données de SLLParc s'inscrit dans cette voie. A savoir la définition, les avantages, le fonctionnement et l'indépendance des données et de programmes.

V.3.1.2 Création de la base de données.

L'outil CASE (AMC*Designor) nous a permis de générer le script SQL de la création de la base de donnés. Il faudra tout de suite dire que ce code a été modifié au niveau des index afin d'avoir une base de données pratique.

Pour éviter le double emploi, nous convions le lecteur à consulter ce code à l'annexe 4 de ce mémoire.

Ce script exécuté par la composante SQL Talk du développeur Centura Builder crée la base de données vide stockée sur un support physique (serveur de base de données, dans notre cas).

V.3.2 Système de Gestion de Base de Données.

Le Système de Gestion de Base de Données (SGBD) est le logiciel qui permet à un utilisateur d'exploiter une base de données[4]. Un des rôles fondamentaux d'un SGBD est de permettre à un utilisateur d'exploiter les données en lui cachant les détails de l'organisation et de la localisation des données sur disques.

Le SGBD de SLLParc est Oracle 7.0. Alors que le langage de développement d'application est le Centura Builder (Voir le point V.4.1 sur son choix).

V.3.3 Système de Gestion de Base de Données Relationnelle.

Le Système de Gestion de Base de Données est Relationnel. Non seulement qu'il est le plus utilisé, mais il offre des nombreux avantages[5] par rapport aux Systèmes de Gestion de Base de Données réseau (CODASYL) ou hiérarchique.

V.3.4 Le langage SQL.

Centura Builder offre la possibilité de déclencher les événements en exécutant des requêtes SQL programmées pour les composants. Les exemples sont donnés à l'annexe 5.

V.3.5 L'outil CASE utilisé.

L'outil CASE disponible à Swiss Life Luxembourg est l'AMC*Designor. C'est donc avec cet outil que nous avons modélisé les données de cette application.

V.4 Conception de l'interface homme-machine dans la réalisation de SLLParc.

V.4.1 Choix du langage de programmation.

Le développement des applications informatiques à Swiss Life Luxembourg se réalise jusqu'à présent en Centura Builder. Centura est un langage de développement de 4° génération. C'est la programmation événementielle, génération des écrans... Il possède de nombreuses fonctions. La programmation se base sur l'identation du code et les requêtes SQL.

Centura Builder, comme d'autre langage de développement, présente

[4] C'est-à-dire décrire, manipuler et traiter les données d'une base.
[5] Voir chapitre deuxième.

quatre modules en son sein :

1° Centura Builder : pour développer des applications.
2° Team Object Manager : gestionnaire des projets,
3° SQL Talk : manipulation du langage SQL,
4° SQLBase Server : gestionnaire des bases de données.

Il y a aussi à Swiss Life Luxembourg, le langage de développement de Microsoft, Visual Basic. Le manque de recul dans ce dernier langage par rapport à la grande expérience et à l'abondante librairie en Centura Builder, a facilité le choix par l'utilisateur.

V.4.2 Construction de la maquette de SLLParc.

A l'aide de Centura Builder, le langage de programmation choisi, nous construisons la maquette de l'application. Elle nous a permis de spécifier davantage les besoins de l'utilisateur et surtout la façon qu'il veut voir son application se présenter et fonctionner.

Nous vous présentons cette maquette validée par son utilisateur à l'annexe 3 de la partie Annexes du présent mémoire. Le mode d'emploi de SLLParc, englobant encore cette maquette, se trouve à l'annexe 6.

V.5 **Méthodologie de développement dans la réalisation de SLLParc.**

V.5.1 Considérations théoriques au développement de SLLParc.

V.5.1.1 Les étapes du développement de SLLParc

Nous avons eu à analyser les besoins, à les spécifier, à concevoir une architecture pour cette application (§ V.5.2), et à programmer. Pendant le développement, la gestion de la configuration a été aussi réalisée (rapport de stage, mode d'emploi de SLLParc). Et, afin de mettre en production l'application, des tests ont été réalisés. Le rôle de la maquette a été important et s'est vérifié lors de la séance de sa validation.

V.5.1.2 Le modèle du développement de SLLParc

SLLParc a été développé suivant le modèle en V.

V.5.1.3 Les méthodes d'analyse et de conception, technique de spécification de SLLParc

La méthode d'analyse et de conception est la méthode Merise avec les éléments à connotation de notre Institut d'Informatique de Namur, rassemblés dans les deux premiers chapitres de ce mémoire.

Parmi les techniques de spécification : les énoncés informels et le modèle entité-association ont été utilisés.

V.5.1.4 SLLParc : test ou preuve ?

Nous avons réalisé des tests de SLLParc en vue d'établir la présence d'erreurs. Elles ont été corrigées en grande partie pour le bon fonctionnement de l'application. Ces tests ont été dynamiques et statiques.

V.5.1.5 Méthode générale du développement de SLLParc

En règle générale (§ IV.5), la méthode du développement a été respectée : étude d'opportunité, la conception du système d'information, la modélisation des données, passage au logique et la réalisation de l'application.

V.5.2 Architecture de SLLParc.

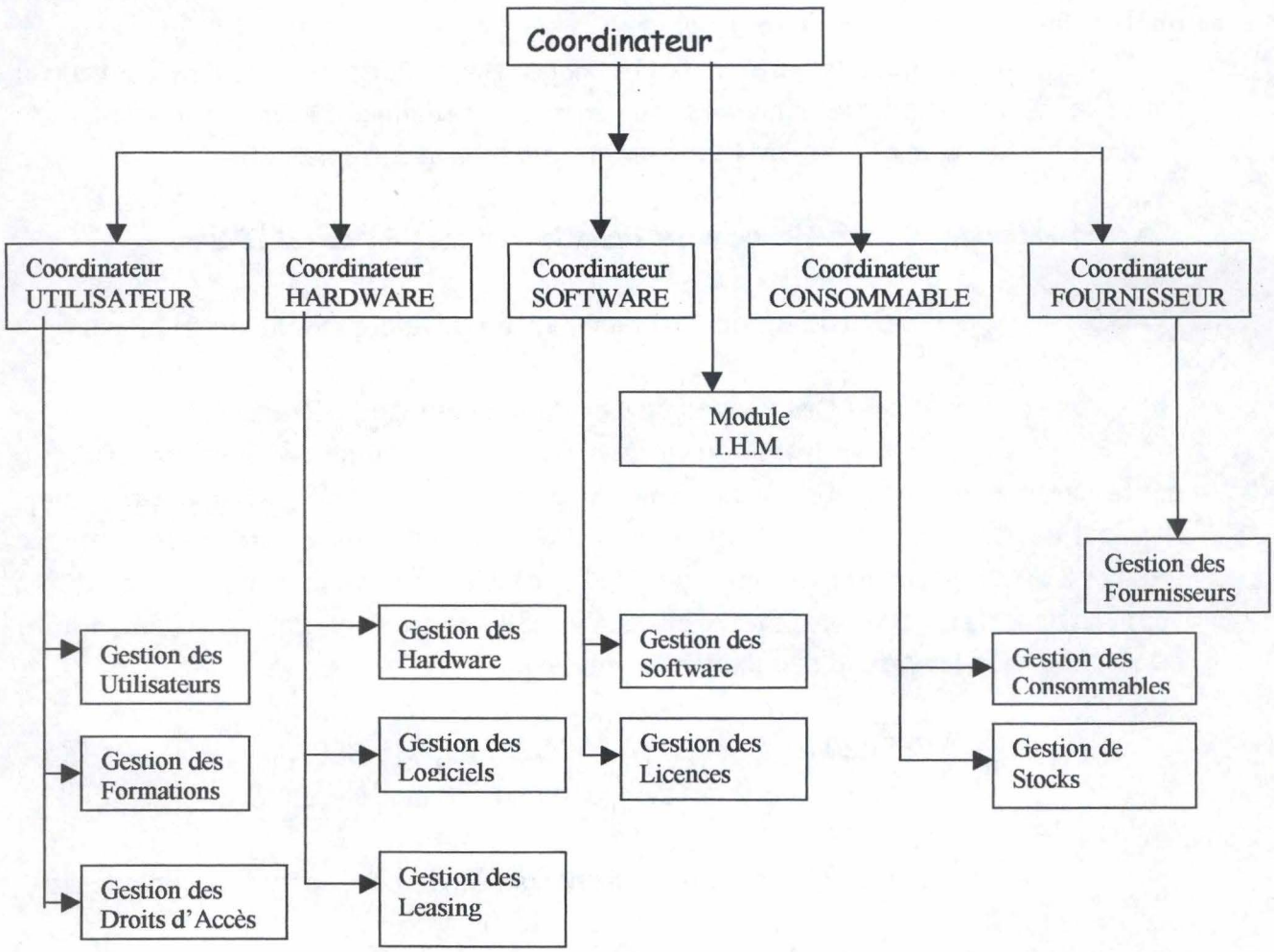


Fig. 5.2 Architecture de SLLParc.

V.5.3 Code initial de l'application.

Beaucoup de classes et librairies ont été mises en place à Swiss Life Luxembourg (SLL) pour le besoin des développements de ses applications. Si bien que chaque application les utilise afin de faciliter le travail et de rester dans le standard de la maison.

Ces classes et librairies sont des spécifications de classes standards de Centura Builder qui ont été testées et ont fait leur preuve au fil des années.

Les boutons de commande spécifiques de l'application (tels que *Fermer, Accepter, Chercher, Ajouter, Editer, Imprimer)* appartiennent à des classes SLL prédéfinies héritant des standards de Centura Builder.
Par exemple le bouton de commande *Fermer* hérite de la classe de bouton de commande simple de Centura Builder et a pour action de fermer la fenêtre qui le contient.

Aucun écran n'est créé individuellement ; il appartient toujours à une classe. C'est ainsi que l'écran de Login, par exemple, est identique à toutes les applications développées à Swiss Life Luxembourg (SLL).

Le formatage des dates a été standardisé dans la librairie SLLDates. Et les champs d'éditions, d'affichage,... ont été standardisés dans la librairie SLLChamps.

Cette étape permet aussi de définir les variables et les fonctions de l'application afin de rester dans les standards de SLL avant de commencer la programmation.

Tous les champs de la base de données sont des variables qu'il faut déclarer. La variable correspondra au nom du champ avec un préfixe qui est fonction du type de la variable. Ceci permet un langage commun pour tous les développeurs de Swiss Life Luxembourg (SLL).

Ces préfixes sont :

- Pour un booléen : bm
- Pour une date : dm
- Pour un long string (>250 caractères) : lm
- Pour un nombre : nm
- Pour un string (<=250 caractères) : sm

Voici un exemple de définition standard des variables d'une table :

```
TABLE HARDWARE

String: smCODEHW
String: smNOMFOUR
```

String: smNSERIE

String: smNLEASING

String: smGROUPEHW

String: smTVPEHW

String: smMARQUEHW

String: smDATACHAT

Number : nmPRIXU

Boolean: bmDISPO

Par fichier (Table), deux types de champs sont disponibles :

=>xxUU : on lui affecte automatiquement le login de l'utilisateur du fichier,
=>xxUT : on lui affecte automatiquement la date et l'heure de la modification du fichier. Où **xx** correspond à 2 caractères uniques de chaque fichier de la base de données.

Chaque table devra faire l'objet d'une fonction. Ceci nous permettra d'utiliser les librairies standard de Swiss Life Luxembourg pour extraire ou insérer les données de la base de données. Le but est d'être efficace, rapide et ne plus récrire les mêmes requêtes.

Voici un exemple d'une fonction standard qui définit une table :

```
Function: lHardware
Description: Les différentes données du fichier Hardware SLLParc.
Returns
Parameters
Static Variables
Local variables
Actions
        Set sPR_KeyPassage='HW'
        Set sNomFilePassage='HARDWARE'
        Set sIden+Passage='CODEHW'
        Set sIdentPassInto='smCODEHW'
        Set sIdentPassage2=''
        Set sIden+PassInto2=''
        Set sIdentPassageAcces=' '
        Set sIdentPassIntoAcces=''
        Set  sdePassage='NOMFOUR,NSERIE,NLEASING,GROUPEHW,TYPEHW,
            MARQUEHW,DATACHAT,PRIXU,DISPO'
```

```
Set
sversPassage=':smNOMFOUR,:smNSERIE,:mNLEASING,:smGROUPEHW,:smTYP
EHW,:smMARQUEHW,:smDATACHAT,:nmPRIXU,:bmDISPO'
Set
sUpdPassage='NOMFOUR=:smNOMFOUR,NSERIE=:smNSERIE,NLEASING=:sm
NLEASING,GROUPEHW=:smGROUPEHW,TYPEHW=:smTYPEHW,MARQUEHW=:
smMARQUEHW,DATACHAT=:smDATACHAT,PRIXU=:nmPRIXU,DISPO=:bmDIS
PO'
```

Voici un exemple d'une fonction standard qui extrait les données de la base de données et les met dans des variables déclarées :

```
Function: ASLLExtraireTous

Description : Extrait toutes les données en relation avec le dernier IFile exécuté.

Returns

Parameters

Static Variables

Local variables
        String: mLocalTexte
Actions
        If SalStrLength( sIdentPassage2 )=0
          Call SqlImmediate('select'||sdePassage||'from
          '||SLLDB||sNomFilePassage||' into '||sversPassage||' where
          '||sIdentPassage| |'=:' | |sIden+PassInto )

        Else
              Set    smLocalTexte= ' sélect  '||sdePassage||' from
              '||SLLDB||sNomFilePassage||' into'||sversPassage||' where
              '||sIdentPassage||'=:'||sIdentPassInto||' and
              '|| sIdentPassage2||'=:'||sIdentPassInto2
              Call SqlImmediate (smLocalTexte)
```

Le code initial ou librairie de SLLparc consiste à définir comme variables tous les champs de chaque table et comme fonction chaque table. Travail de longue haleine mais payant lors de la programmation.

V.5.4 Programmation.

Une fois encore, nous demandons au lecteur de lire l'essentiel du code Centura Builder à l'annexe 5, où nous présentons les lignes de codes écrits derrière quelques composants (programmation événementielle).

V.6 Conclusion du chapitre.

Ici, nous invitons le lecteur à consulter les annexes pour se faire une idée de l'application SLLParc. Toutefois, l'exécution de cette application reste recommandée. Notre vœu a été accompli, considérant que l'application est fonctionnelle ; les divers et multiples contacts avec l'utilisateur ont contribué positivement au développement réussi de SLLParc.

Notre premier contact avec le langage de programmation Centura Builder, l'adaptation au style d'interfaçage et aux guides propres à Swiss Life Luxembourg ont été plus bénéfiques.

Certes, au début, nous avions rencontré des difficultés que nous avons surmontées. Je ne saurai les énumérer toutes, mais disons néanmoins que l'adaptation du schéma du modèle conceptuel de données et du script de la base de données nous ont conduit à prendre du recul dans la génération des schéma et code.

L'apprentissage d'un autre langage de programmation est bénéfique. Malgré cela, nous avons tenu le pari : réaliser cette application dans le temps imparti. Ne reste à présent que sa maintenance qui permettra de la stabiliser et la faire évoluer.

Partie III

DE LA THEORIE A LA PRATIQUE ET VICE VERSA

Dans cette dernière partie, nous essayons de relier la théorie de la première partie et la pratique de la deuxième partie.

Dans ce seul chapitre, nous mettons en parallèle ces deux parties.

Chapitre 6 : Lien de la théorie de systèmes d'information à la pratique.

Chapitre 6 : De la théorie à la pratique et vice versa.

Sommaire

Chapitre 6 : De la théorie à la pratique et vice versa.

VI.1 Introduction.

Nous voulons confronter, dans ce chapitre, la théorie retenue à la première partie et la pratique de la deuxième partie. Il s'agit d'en sortir une convergence, une divergence, une remarque, un décalage...

Toute bonne pratique se repose sur la théorie. Mais, la pratique ne suit pas rigoureusement la théorie. La réalisation de ce logiciel sur mesure nous a confirmé cette proposition. On dirait que là où la théorie est abondante, la pratique passe vite ou vice versa.

Nous aborderons donc ce chapitre en suivant les quatre chapitres de notre première partie ; comme ce qui a d'ailleurs été fait à la deuxième partie.

VI.2 Théorie et pratique sur l'analyse fonctionnelle.

VI.2.1 Conception des systèmes d'information.

Partant de sa définition (I.2.1), SLLParc[1] est un système d'information et une application informatique. Etant donné que le traitement automatique de l'information se fait sur ordinateur, SLLParc est un système automatisé d'information comme nous l'avons défini au paragraphe I.2.2.

En pratique, à chaque fois qu'il y a une base de données pour une application, c'est un système d'information qui est développé.
SLLParc en tant que système d'information possède les trois niveaux d'abstraction (conceptuel, organisationnel et opérationnel) comme le résume la fig. 1.1.

Appliquant la fig. 1.2 du modèle général d'un système d'information à SLLParc, nous disons : un message-donnée, en provenance de l'environnement du système d'information (S.I.), est communiqué au système d'information qui le traite via un processeur, éventuellement à l'aide de sa mémoire, pour engendrer un message-résultat lequel est, à son tour, transmis à l'environnement du S.I.
Exemple : mon message-donnée est le code du Hardware (PC, Imprimante,...) et

[1] SLLParc est l'application ou le logiciel sur mesure que nous avons développé à Swiss Life Luxembourg pour la gestion du parc informatique dans le cadre du stage de fin d'études.

SLLParc me retournera comme message-résultat l'historique de ce matériel.

A propos du cycle de vie de SLLParc (voir fig. 1.3), ce dernier est en sa première version : son développement étant fini, c'est aujourd'hui l'étape de l'utilisation et de la maintenance.

La méthodologie de conception d'un système d'information (cfr. I.2.6) nous a beaucoup aidé dans la conception de SLLParc afin de ne pas dépasser le planning élaboré et de rester dans le délai de réalisation. Cette méthodologie est basée sur quatre pôles : les modèles (voir les paragraphes suivants), la démarche, les outils logiciels (AMC*Designor, Centura Builder), et l'organisation.

Cette théorie sur la conception des systèmes d'information a été bénéfique, c'est la raison pour laquelle nous l'avons rappelée. Tout développeur d'application y passe nécessairement dans sa pratique.

VI.2.2 Modèle conceptuel de données.

Si nous avons rappelé quelques concepts de base sur le modèle conceptuel de données, c'est parce qu'aujourd'hui la méthode de structuration des données est incontournable pour un développement d'une application. Tout analyste représente la réalité simplement par un modèle Entité /Association.

Aussi, après l'analyse des besoins, un schéma de base du modèle conceptuel de données a été conçu pour SLLParc (cfr. V.2.2.1). L'outil CASE utilisé est l'AMC*Designor. Ce schéma n'est pas statique. Il a été adapté pour répondre aux besoins de l'utilisateur et pour faciliter le développement suivant le langage de programmation utilisé. Cette adaptation (cfr. V.2.2.2) est le fruit de la validation par l'utilisateur (cfr. I.3.5).

Ici, la pratique nous a appris les défauts du modèle conceptuel de données dont nous avons signalé au paragraphe I.3.1.

VI.2.3 Modèle Logique de données.

Partant du schéma Entité - Association et généré par un outil CASE, le modèle logique définit une solution qui est « exécutable » par une machine abstraite (indépendante de la machine réelle). Le schéma qui en est issu, est la perception du programmeur ou de l'administrateur du système de gestion de base de données.

Le schéma du modèle logique de SLLParc (cfr. V.2.3) a été obtenu par génération du schéma du modèle conceptuel de données grâce à l'outil

CASE AMC*Designor. A l'examen de ce schéma, les règles évoquées au paragraphe II.4 sur le système de gestion de base de données relationnel sont respectées.

VI.2.4 Modèle physique de données.

Le schéma physique d'une base de données est constitué de son schéma logique complété des constructions et paramètres techniques qui le rendent opérationnel et efficace. Suivant les critères de performance.

Dans les SGBD modernes (comme Oracle 7.0 que nous avons choisi), ces constructions et paramètres sont inconnus des utilisateurs et des programmeurs (notion d'indépendance physique).

VI.2.5 Démarche de l'analyse fonctionnelle.

La fig. 1.6 nous représente le schéma des étapes partant du réel perçu (analyse d'opportunité) à l'analyse conceptuel. C'est le schéma de la démarche fonctionnelle.

Avec ce que nous avons réalisé pour SLLParc, nous nous retrouvons bien dans ce schéma. En effet, nous avons identifié le projet. Nous avons défini le projet cadre et avons fait une étude critique de l'existant. Après avoir finalisé cela, nous avons élaboré des solutions pour en choisir une : informatisation de la gestion du parc informatique. Après avoir élaboré un planning des tâches ultérieures, nous avons commencé l'analyse conceptuelle.

La démarche de l'analyse fonctionnelle est une étape qui précède l'analyse conceptuelle. Elle se fait toujours.

VI.2.6 Réalisation assistée des applications informatiques.

La théorie de cette notion souhaite que les applications informatiques soient le plus directement possible issues ou dérivées des spécifications fonctionnelles. Cette dérivation directe devrait contribuer de façon significative à la qualité des systèmes d'information opérationnels et à la productivité de leur développement, aussi à leur maintenance. Nous y avons soulevé trois points :

Premièrement, pour garantir que les applications informatiques soient un reflet fidèle des spécifications fonctionnelles, il faudra élaborer de règles générales de construction systématique, qui permettent de transformer progressivement les spécifications fonctionnelles en applications informatiques.

Deuxièmement, ces règles systématiques de construction et a fortiori l'automatisation sont souvent synonymes de rigidité dans le contexte des techniques informatiques actuelles. Ainsi, il faut prévoir des adaptations manuelles ad hoc.

Troisièmement, le recours à des techniques de transformations automatiques devant permettre des adaptations manuelles ponctuelles, ne saurait évacuer le problème de la documentation, indispensable à la maintenance inévitable des systèmes d'information.

Cette vision futuriste sur les systèmes d'information, a été tenue compte lors de la réalisation de SLLParc. Les adaptations manuelles ont été faite au-delà de code généré. Notre rapport de stage est une documentation indispensable sans omettre le Team Object Manager, module qui conserve l'historique de l'évolution du développement d'une application en Centura Builder.

VI.3 **Théorie et pratique sur les bases de données.**

VI.3.1 Concepts de base de données

Tous les concepts théoriques évoqués sont utiles et méritent d'être rappelés. Les avantages d'une base de données, par exemple, rehaussent l'importance d'avoir un logiciel (SLLParc) au lieu de gérer le parc informatique à l'aide des formulaires. Un programme d'application (comme SLLParc) accède à la base de données à partir du niveau externe, qui communique avec le niveau logique, ce dernier étant en liaison avec le niveau ou support physique de la base (Fig. 2.2).

Nous estimons que ces concepts sont utiles avant d'entreprendre la conception d'une base de données. C'est de la bonne théorie.

VI.3.2 Système de gestion de bases de données (SGBD)

Les données d'une base doivent être manipulées afin de fournir l'information cherchée. Ceci n'est possible qu'à l'aide d'un logiciel nommé SGBD (cfr. II.3).

Oracle 7.0 est le SGBD que nous avons utilisé pour SLLParc. L'architecture (Fig. 2.3) et le déroulement d'une recherche (Fig. 2.4) montrent la complexité et l'importance de ce logiciel. Un SGBD est à la base de données ce qu'un système d'exploitation est à l'ordinateur.

VI.3.3 Modèle relationnel (SGBDR)

Le SGBD de SLLParc (Oracle 7.0) est relationnel. SLLParc s'inscrit parmi les innombrables bases de données relationnelles qui sont en vogue pour le moment. Les notions de SGBDR sont rappelées au paragraphe II.4.

VI.3.4 Manipulation des données : Le langage SQL[2]

C'est le langage de requête utilisé pour manipuler les données d'une base. Aujourd'hui très populaire, le SQL est proposé par la plupart des SGBD modernes.

Nous avons utilisé énormément des requêtes SQL lorsque nous avons développé SLLParc (embedded SQL). Ces requêtes ont été associées aux composants de Centura Builder, notre langage de programmation. La connaissance théorique de SQL était donc indispensable.

VI.3.5 Algèbre relationnelle

Nous avions rappelé ces concepts (opérateurs ensembliste et additionnels) pour compléter la théorie sur le langage SQL.

VI.3.6 Outil CASE

AMC*Designor est l'outil CASE[3] que nous avions utilisé pour SLLParc. Il est un CASE tool vertical et Upper CASE (cfr. II.7.2). Il nous a permis d'obtenir le schéma logique (cfr. Annexe 2) et le script de la base de données (cfr. Annexe 4).

VI.4 **Théorie et pratique sur la conception des interfaces homme-machines.**

VI.4.1 Problématique de conception des IHM[4]

Une interface d'une application informatique doit être utile et utilisable. SLLParc a une interface utile car elle fournit à l'utilisateur les fonctions nécessaires pour produire les résultats attendus ; et utilisable parce que les utilisateurs ont le profil cognitif de la manipuler aisément. Ces qualités ont été obtenues après une séance de validation avec l'utilisateur, ce qui nous a permis de respecter les principes de conception énumérés à la Fig. 3.1.

---

[2] Structured Query Language.
[3] Computer Aided Software Engineering.
[4] Interface Homme – Machine.

En ce qui concerne la métaphore utilisée pour réduire davantage l'effort cognitif de l'utilisateur, l'interface de SLLParc a été construite sur base de la métaphore du mini-monde.

VI.4.2 Présentation des matériaux de construction d'une IHM

Pour construire l'interface de SLLParc, nous avons utilisé des objets interactifs concrets comme des menus, des boîtes de dialogue, des butons de commande, des buton-radios, ...

VI.4.3 Ergonomie logicielle des IHM

Les notions sur l'ergonomie logicielle (cfr. III.4) nous ont permis de construire l'interface de SLLparc en respectant les critères d'évaluation de la qualité et de conception d'une interface (dont le résultat est présenté à l'annexe 3).

VI.4.4 Quelques règles ergonomiques

La connaissance de ces règles sont indispensables pour l'élaboration d'une interface. On les considérera en tenant compte de l'environnement de travail et des besoins de l'utilisateur.

VI.5 **Théorie et pratique sur la méthodologie de développement de logiciels.**

VI.5.1 Etapes et Modèles de développement du logiciel.

Le développement de SLLparc s'identifie au modèle en V décrit au paragraphe IV.2.2 et illustré par la figure Fig. 4.2. Ce modèle propose des étapes importantes du développement d'un logiciel (IV.2.1).

SLLparc a aussi suivi le même chemin à savoir :

- Analyse des besoins ;
- Spécification globale ;
- Conceptions architecturale et détaillée ;
- Programmation ;
- Gestion de configurations ;
- Validation et vérification ;
- Rôle de la maquette.

VI.5.2 Techniques de spécification et méthodes d'analyse et de conception.

Comme source d'information, nous avons utilisé les énoncés informels c'est-à-dire des spécifications écrites en langage naturel. Et, le modèle entité-association est la technique graphique que nous avons utilisée pour spécifier les données de SLLparc.

VI.5.3 Test et Preuve de logiciel.

SLLParc a été testé plusieurs fois. Ces tests ont été dynamiques (black-box testing). Nous avons aussi testé SLLParc avec des tests statiques (white-box testing). La correction du code de programme est basée sur la connaissance du langage de programmation Centura Builder.

VI.5.4 Méthode générale de développement d'une application.

Toute la théorie décrite au paragraphe IV.5 a été suivie de près, sans être rigoureux dans l'ordre des activités. SLLParc a été développé suivant la méthode classique de développement de logiciel.

VI.6 **Conclusion du chapitre.**

Nous avons essayé de voir ce que la théorie a fourni à la pratique et vice versa dans ce domaine du développement de logiciel.

La théorie supporte la pratique. La bonne pratique est faite sur base d'une théorie suffisante.

CONCLUSION GENERALE

Ecrire un software est un travail très créatif et passionnant.

Tout au long de ce travail, nous nous sommes efforcé de pouvoir présenter quelques connaissances de base d'un développeur d'application informatique. Ceci suite à une application que nous avons eue le privilège et l'opportunité de développer pendant notre stage : SLLParc qui est une application de gestion de parc informatique de Swiss Life Luxembourg.

La première partie de ce mémoire est le résultat d'un énorme travail de synthèse théorique de quatre vastes domaines de l'informatique de gestion à savoir : l'analyse fonctionnelle, les bases de données, la conception des interfaces homme-machine et la méthodologie de développement de logiciel.

Dans la deuxième partie, nous avons expliqué, en rapport à la partie théorique, la conception et la réalisation de SLLParc. Le résultat en est ce logiciel sur mesure qui est utilisé à Swiss Life Luxembourg.

Par rapport à la théorie rappelée de la première partie et vu la pratique réalisée à la deuxième partie, la troisième partie tente de tirer un lien, un constat, une remarque, un point de vue entre les deux.

D'une gestion du parc informatique grâce aux formulaires, nous avons conçu et réalisé un logiciel sur mesure nommé SLLParc qui gère le parc informatique de Swiss Life Luxembourg. Plus encore, ce projet a été réalisé dans le délai, dans un environnement et avec un langage de programmation nouveaux pour nous.

En définitive, nous nous réjouissons de la réussite de ce challenge dans le temps imparti, et de cette application actuellement opérationnelle. L'étape de la maintenance de cette application est en cours pour la stabiliser et la faire évoluer. Cette expérience nous a été très bénéfique.

Restant ouvert à toute remarque constructive, nous nous excusons pour des erreurs non décelées dans le texte. Et aussi des questions non élucidées qu'attendait le lecteur.

Table des figures

BIBLIOGRAPHIE

[Bod, 94] BODART F., PIGNEUR Y., *Conception assistée des systèmes d'information. Méthode. Modèles. Outils*, 2° édition, Masson, Avril 1994.

[Bod, 99] BODART F., *Cours introductif à la conception des interfaces Homme - Machine*, Cours, Institut d'Informatique, FUNDP, 1999.

[Cas, 87] CASTELLANI X., *Méthode générale d'analyse des applications informatique*, Tome 0, Masson, Mai 1987.

[Dat, 95] DATE C. J. *An introduction to database systems*, sixth edition, Addison-Wesley Publishing Compagny, 1995.

[Dat, 95] DATE C. J. *Relational database* , sixth edition, Addison-Wesley Publishing Compagny, 1986.

[Del, 98] DELMAL P. *SQL 2 Application à Oracle, Access et Rdb, 2° édition*, De Boeck-Université, 1998.

[Eng, 00] ENGLEBERT V., *Case tools, Meta-Case tools, Meta-Modelisation*, Cours, Institut d'Informatique, FUNDP, 2000.

[Gal, 89] GALACSI, *Conception de bases de données. du schéma conceptuel aux schémas physiques*, Dunod, 1989.

[Gal, 96] GALLEZ D., *Base de données*, Cours, EPFC, 1996.

[Gau, 96] GAUDEL M.-C., all, *Précis de génie logiciel*, Masson, 1996.

[Hai, 86] HAINAUT J.-L., *Conception assistée des applications informatiques, 2- Conception de la base de données*, Masson, 1986.

[Hai, 98] HAINAUT J.-L., *Base de données - Technologie et conception*, Cours, Institut d'Informatique, FUNDP, 1998.

[Hai, 99] HAINAUT J.-L., *Base de données Matière approfondie*, Cours, Institut d'Informatique, FUNDP, 1999.

[Hai, 00] HAINAUT J.-L., *Bases de données et modèles de calcul, Outils et méthodes pour l'utilisateur*, 2° édition, Dunod, 2000.

[Mat, 94] MATHERON J.-P., *Comprendre Merise. Outils conceptuels et organisationnels*, Eyrolles, Avril 1994.

[Mif, 00] MIFUKU A., *Rapport de stage effectué à Swiss Life Luxembourg, Gestionnaire d'un parc informatique*, 1999-2000.

[Lou, 90] LOUVET G., *Se former à Merise. La modélisation conceptuelle. Les éditions d'organisations*, 1990.

[Som, 92] SOMMERVILLE I., *Le Génie Logiciel* , Addison-Wesley, 1992.

[Tar, 80] TARDIEU H., NANCI D., PASCOT D., *Conception d'un système d'information. Construction de la base de données*, Les éditions d'organisation, 1980.

[Van, 93] VANDERDONCKT J., *Corpus ergonomique minimal des applications de gestion*, Institut d'Informatique, FUNDP, 1993.

ANNEXES

Annexe 1 : Formulaires de gestion du parc informatique de Swiss Life avant le développement de SLLParc.

Annexe 2 : Schéma des modèles conceptuels et physique des données de SLLParc.

Annexe 3 : Maquette (Interface Homme - Machine) de SLLParc.

Annexe 4 : Script de la création de base de données de SLLParc.

Annexe 5 : Codes de programmation de SLLParc.

Annexe 6 : Mode d'emploi de SLLParc.

Annexe 1 :

Formulaires de gestion du parc informatique de Swiss Life avant le développement de SLLParc.

TypObj	Nom	User	Marque	NumSer	DtAcht	PrixUnit	Fourn	Garantie	Num Cont Leas	Durée Mois	Dtfin Leas
Ecran	CPQV70-I	PIERRE	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-I	PIERRE	DESKPRO 4000 6180 16/1620 CDS	8646BBF60796			Computer*Systems*	3 ans			
Clavier		PIERRE	COMPAQ	B05490A39EEB94	09-déc-96		Computer*Systems*				
Imprimante		PIERRE	HP LASERJET 6P	NL1V146411	09-déc-96	58 587	Computer*Systems*				
Ecran	CPQV70-D	PIA/JUTTA	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-J	PIA/JUTTA	DESKPRO 4000 5166 16/1620	8641BBD42480	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier		PIA/JUTTA	COMPAQ	B05430A39EE798	09-déc-96		Computer*Systems*				
Imprimante	VENUS	PIA/JUTTA	HP LASERJET 5M	NL1W038452	30-juil-96	55 883	TELINDUS				
Ecran	CPQV70-Q	OLIVIER	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-K	OLIVIER	DESKPRO 4000 5166 16/1620	8649BBD40938	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier		OLIVIER	COMPAQ	B05490A39EED6X	09-déc-96		Computer*Systems*				
Imprimante	PRINSAVE	OS/MB	HP LASERJET 5M	NLCB076873	11-sept-97	28 948	Computer*Systems*				
Ecran	CPQV70-N	CHRISTINE	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-M	CHRISTINE	DESKPRO 4000 5166 16/1620	8641BBD42539	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier	COMPAQ	CHRISTINE	COMPAQ	B05490A39EEBA7	09-déc-96		Computer*Systems*				
Ecran	CPQV70-C	BT	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-N	BT	DESKPRO 4000 5166 16/1620	8649BBD40085	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier		BT	COMPAQ	B0BA80E39FXDB3			Computer*Systems*				
Imprimante	HP6P-C	BT	HP LASERJET 6P	NLBB106130	09-déc-96	25 033	Computer*Systems*		19610101	36	1-janv-00
Ecran			IBM P50	88-55101	04-oct-95	21 600	TELINDUS				
PC	IBM-K		IBM	ZI554XMPK	18-mars-96	88 000	TELINDUS	3 ans			
Clavier			COMPAQ	B0A390B39G3PT9	16-avr-98		Computer*Systems*				
Modem			Sportster Flash USRobotics	071663-01	16-avr-98	5 367	Computer*Systems*				
Ecran	CPQV70-Y	MAURA	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-P	MAURA	DESKPRO 4000 5166 16/1620	8649BBD40168	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier		MAURA	COMPAQ	B05430A39E8QEK	09-déc-96		Computer*Systems*				
Imprimante		MAURA									
Ecran		INSPECT.	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-Q	INSPECT.	DESKPRO 4000 5166X M2400/16MB/	8649BBD40039	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier		INSPECT.	COMPAQ	B05490A39EED6I							
Imprimante		INSPECT.	KYOCERA FS-1500	AAS2500651							
Ecran	CPQV70-V		COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans			
PC	CPQ1620-R		DESKPRO 4000 5166 16/1620	8649BBD40167	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier			COMPAQ	B05490A39EED6J	09-déc-96						
Imprimante			HP LASERJET 6P	NLCB076853	11-sept-97	28 948	Computer*Systems*		19610105	27	1-janv-00
Ecran	CPQV75-1	AS	COMPAQ V75	853CF23AB826	30-mars-99	67 500	Telindus	3 ans			
PC	CPQ1620-S	AS	DESKPRO 4000 5166 16/1620	8643BBD40497	09-déc-96	50 326	Computer*Systems*	3 ans	19610101	36	1-janv-00
Clavier		AS	COMPAQ	B05490A39EEBAE	09-déc-96		Computer*Systems*				

TypObj	Nom	Marque	NumSer	DtAcht	PrixUnit Luf HT	Fourn	Garantie	Num Cont Leas	Durée Mois	Dtfin Leas
PC	IBM - C	IBM 330-P75	HP55000DV48	04-oct-95	52 830	Telindus	/	/	/	/
PC	SERVERNT	COMPAQ Deskpro EP 6400		30-mars-99	67 500	Telindus	3 ans			
Serveur	CPQ200	COMPAQ Prosignia 200	8808BWP10053	16-avr-98	97 637	Computer *Systems*		196110107	16	01/01/2000
Serveur	SLLWEB	COMPAQ Prosignia 300	8609HUN30160	12-juin-96	134 545	Telindus		19610101	36	01/01/2000
Serveur	Console	COMPAQ Proliant 2500	8649HWB20139	09-déc-96	378 333	Computer *Systems*		19610101	36	01/01/2000
Serveur	FIREWALL	COMPAQ Proliant 400	8910CJS10289	17-mars-99	102 973	Computer *Systems*				
Serveur	SLLPROXY	COMPAQ Proliant 400	8910CJS10287	17-mars-99	94 345	Computer *Systems*				
Serveur	ORACLE	COMPAQ Proliant 400	8910CJS10269	17-mars-99	94 345	Computer *Systems*				
Serveur	CPQ300	COMPAQ Proliant 400	8910CJS10268	17-mars-99	94 345	Computer *Systems*				
Mainframe	S44F1624	IBM AS400 Modèle F10		14-oct-98	772 390	PROGET	3	065207-FTRA01	36	31/12/2001
Imprimante		KYOCERA FS-1500					/	/	/	/
Ecran	CPQV70-R	COMPAQ V70		09-déc-96	22 391	Computer*Systems*	3 ans	19610101	36	01/01/2000
BlackBox		Server select 8-port to cpu		25-mars-99	74 635	Computer*Systems*				
Clavier		COMPAQ	B05490A39EEB8R			Computer*Systems*				

Ordinateur	Utilisateur	Conf. clavier	Imprimante	Mot de passe clavier	LOGIN *name*	Bios
CPQ1620-A	Rentiers (MMI)	BE	HP 6P	PLUS	Rentiers	
CPQ1620-B	RAGOT Emmanuelle	BE	HP 5Si	JUILLET	EMMA	
CPQ1620-D	DUCHON Céline	SF	HP 5M	MOZART	CELINE	
CPQ1620-E	BERG Luc	BE	H.P. 6P	BRAVO	BERG	
CPQ1620-F	LEGUINA Benno	BE	HP 5Si	PLUS	BENNO	
CPQ1620-G	NIEDERLAENDER Sophie	BE	HP 5Si	BRAVO	SOPHIE	
CPQ1620-H	MICHIELS Daniel	BE	HP 5Si	DAKAR	DANIEL	
CPQ1620-I	DUBRU Pierre	BE		PEDRO	PIERRE	
CPQ1620-J	WEBER Pia	SF	HP5M	SOLEIL	PIA	
CPQ1620-K	STOQUI Olivier	BE	HP 5M	OLIVE	OLIVIER	
CPQ1620-M	GEORIS Christine	BE	HP 5M	BOUCHON	CHRISTINE	
CPQ1620-N		BE	HP 6P	PLUS		
CPQ1620-O	DELPLANQUE France	BE		SUISSE	Internet	
CPQ1620-P	BERRINI Maura	SF		BONGO	MAURA	
CPQ1620-R		BE	HP6P	SUISSE		
CPQ1620-S	SCHMIT Arnaud	BE	HP 5Si	BRAVO	AS	
CPQ1620-U	BRAUN Roger	SF	HP 6P	TOTO	BRAUN	
CPQ1620-V	SIMON Ronny	SF	HP 5Si	MAETE	RONNY	
CPQ1620-X	MARTIN Georges	BE	H.P. 6P	MERCURE	GM	
CPQ1620-Y	FAGIANI Orietta	BE	H.P. 5Si	FELIX	OF	
CPQ1620-Z	ZINGRAFF Daniel	BE	HP 5M	BRAVO	DZ	
CPQ1620-AB	WINTER Eric			CLAVIER	ERIC	
CPQ1620-AC	VOMSCHEID Stéphane			CLAVIER	SV	
CPQ1620-AG	CUQUEMELLE Etienne			BIH	EC	
CPQ1620-AH	DEOM Thierry			PLUS	Thierry	
CPQ6400-A				MARS		
CPQ6400-B	BARY Armelle	BE	HP 5M	GIRAFE	ARMELLE	08/02/1999
CPQ6400-C	DELARUE Alain	BE	HP 5M	BRAVO	ALAIN	08/02/1999
CPQ6400-D				ROUGE		
IBM_K				BLUES		

Ordinateur	Utilisateur	Logiciel installé
CPQ6400-B	BARY Armelle	- Front Page - InternetExplorer - Acrobat Reader
CPQ6400-C	DELARUE Alain	-Crystal Report

Formulaire d'entrée

Swiss Life (Luxembourg) S.A.

Nom : CUQUEMELLE

Prénom : Etienne

Date d'entrée : 10-juin-98

Département : Administration & Finances

Service : Gestion

Equipements spécifiques : Calculatrice, Imprimante

Location du bureau :

Etage
- (•) 1et étage
- ( ) 2ième étage

Description : Local internet

Contrôle d'accés
- (•) Tous les terminaux sauf 2ième étage système
- ( ) Tous les terminaux

Gestion horaire
- (•) Du lundi au vendredi de 6 heures 30 à 21 heures
- ( ) Tout le temps

Applications

[x] Password	Changer le mot de passe pour les applications
[x] Diffusion	Gestion des adresses Swiss Life
[ ] Comptabilité	Logiciel de comptabilité
[x] Prinsave	Gestion et suivi des conrtats Prinsave
[ ] Virements	Impression des virements Swiss Life
[ ] Rentiers	Gestion et suivi des rentiers de l'assurance Groupe
[ ] Rente	Projet de rente
[ ] Optilux	Gestion et suivi des contrats Optilux
[ ] Projet Retinsave	Impression de projet Retinsave
[ ] Projet Prinsave	Impression de projet Prinsave
[ ] Projet Individuelle	Impression de projet Individuelle
[ ] Indy500	Gestion et suivi des assurances individuelles

Si Indy500 indiquez les différents rôles choisies :

A remplir par le responsible de service

Service Informatique

CUQUEMELLE
Etienne

Date de la 1ère formation d'initiation : . 11-juin-98 Heure : 8:30

Extension téléphonique : 222

Domain SLLUX / Réseau NT :

Nom d'utilisateur : EC

Mot de passe : * 123456 * à changer au premier login

Numéro de carte pointeuse : 57

Droits d'accées au répertoire :

H:	SERVERNT\SYS\HOME\EC
P:	P:\Administration & Finances Departement\Gestion service
P:	P:\Commun

Informations nécéssaires pour commencer

Formations

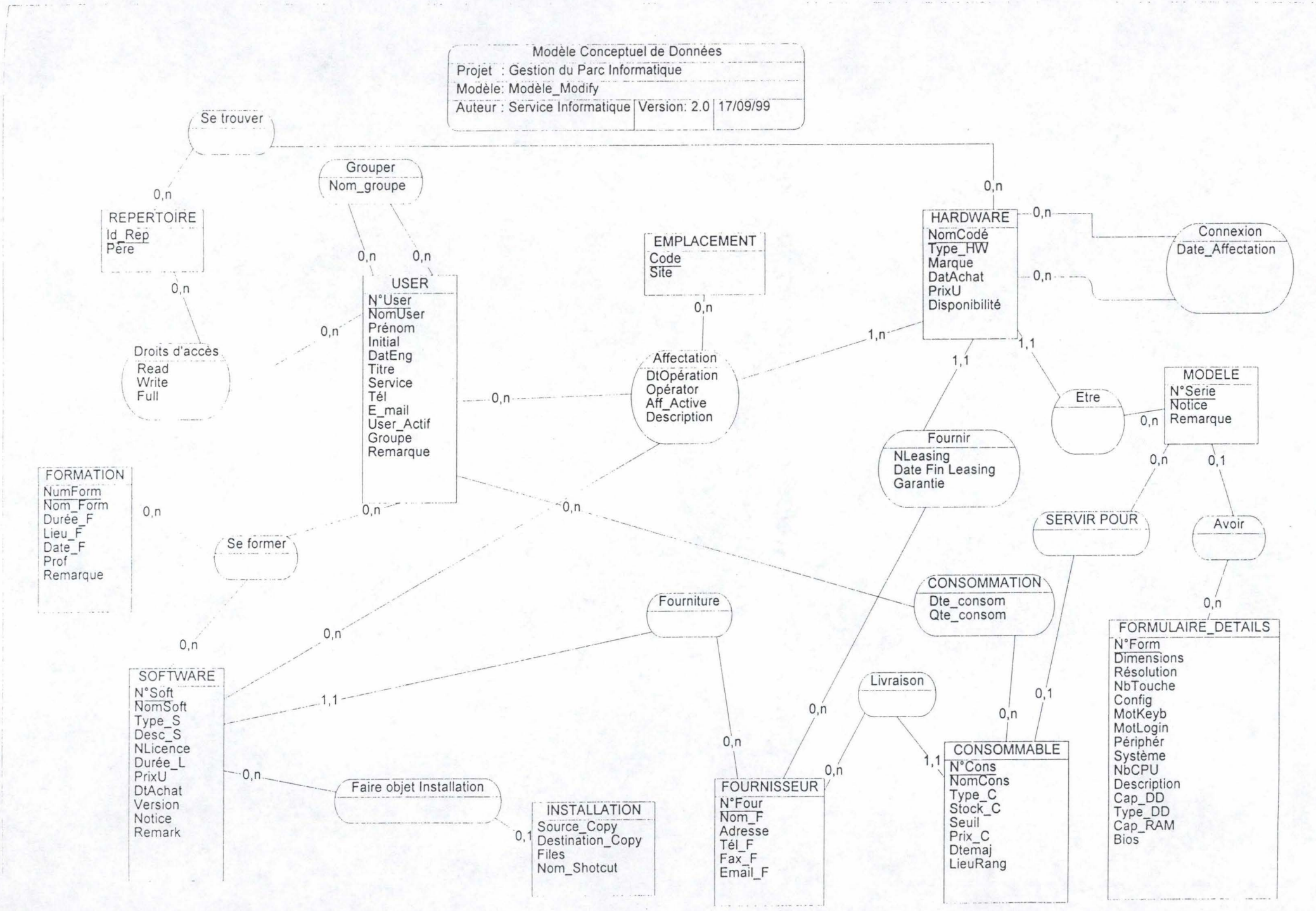
CUQUEMELLE
Etienne

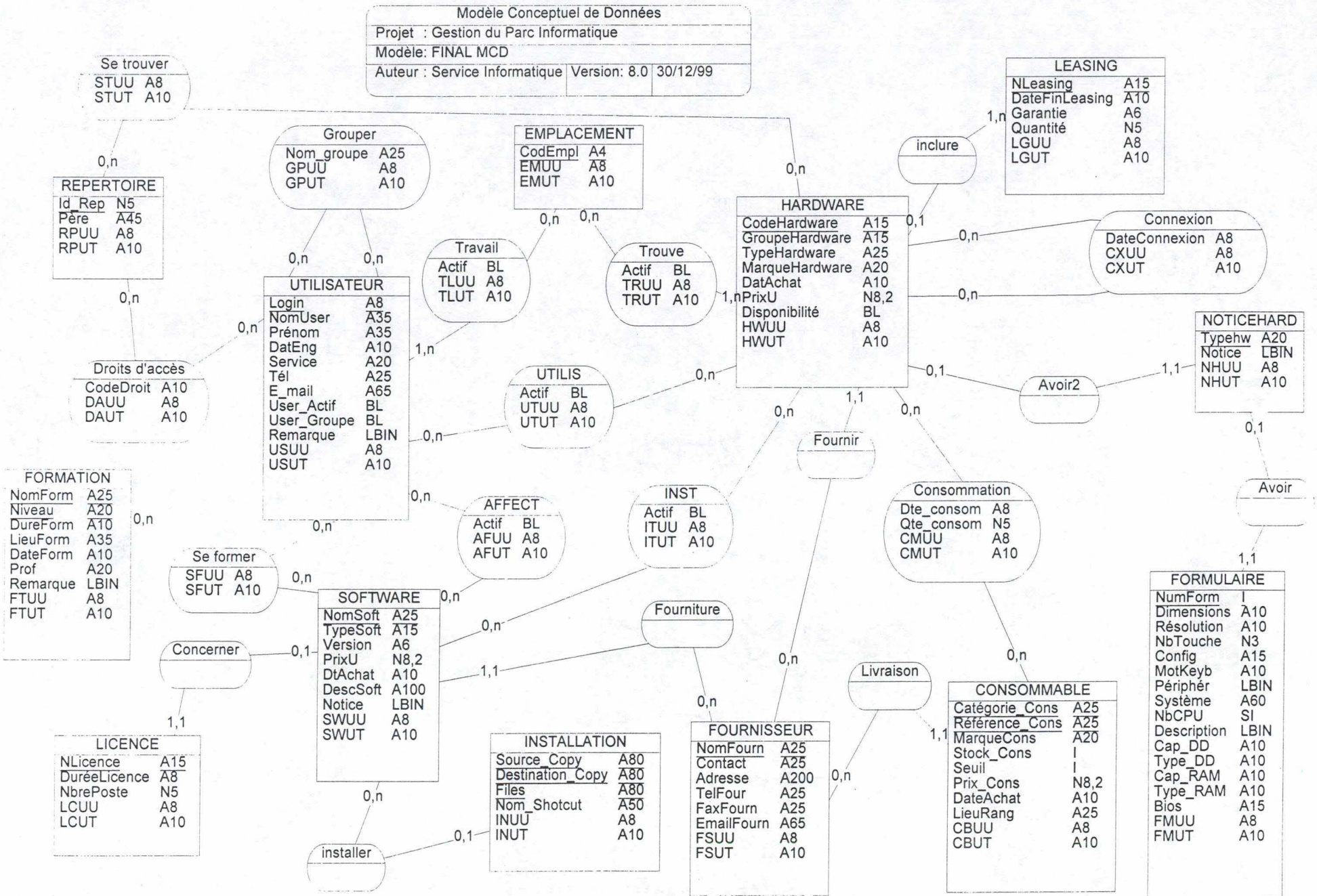
Applications	Sujet	Date + heure
Password	Initiation	16/juin/98 11:30
Diffusion	Formation de base	6/mar/98 9:00
Indy500	Formation Leshings	
Prinsave	Initiation Prinsave	15/juin/98 14:00
Virements		
Rentiers		
Rente		
Optilux		
Projet prinsave		
Projet Retinsave		
Projet Individuel		

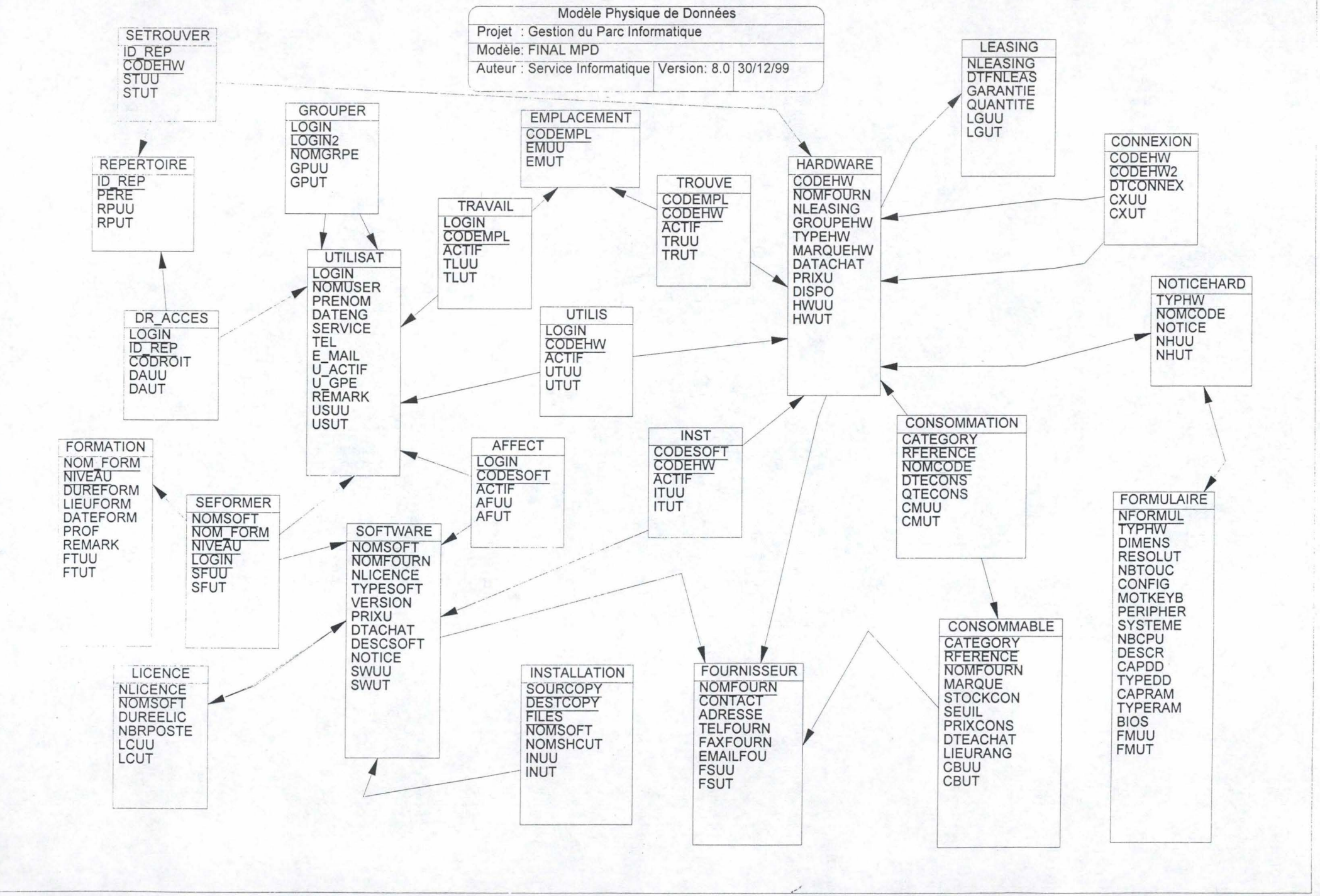
Outils bureautiques	Sujet	Date
Outlook		
Word		
Excel		

Annexe 2 :

Schémas des modèles conceptuel et logique de données de SLLParc.


Modèle Conceptuel de Données
Projet : Gestion du Parc Informatique
Modèle: Modèle_Modify
Auteur : Service Informatique
Version: 2.0
17/09/99
Se trouver
Grouper
Nom_groupe
REPERTOIRE
Id_Rep
Père
Droits d'accès
Read
Write
Full
USER
N°User
NomUser
Prénom
Initial
DatEng
Titre
Service
Tél
E_mail
User_Actif
Groupe
Remarque
EMPLACEMENT
Code
Site
Affectation
DtOpération
Opérator
Aff_Active
Description
HARDWARE
NomCodé
Type_HW
Marque
DatAchat
PrixU
Disponibilité
Connexion
Date_Affectation
Etre
MODELE
N°Serie
Notice
Remarque
Fournir
NLeasing
Date Fin Leasing
Garantie
FORMATION
NumForm
Nom_Form
Durée_F
Lieu_F
Date_F
Prof
Remarque
Se former
SERVIR POUR
Avoir
CONSOMMATION
Dte_consom
Qte_consom
Fourniture
FORMULAIRE_DETAILS
N°Form
Dimensions
Résolution
NbTouche
Config
MotKeyb
MotLogin
Périphér
Système
NbCPU
Description
Cap_DD
Type_DD
Cap_RAM
Bios
SOFTWARE
N°Soft
NomSoft
Type_S
Desc_S
NLicence
Durée_L
PrixU
DtAchat
Version
Notice
Remark
Faire objet Installation
INSTALLATION
Source_Copy
Destination_Copy
Files
Nom_Shotcut
Livraison
FOURNISSEUR
N°Four
Nom_F
Adresse
Tél_F
Fax_F
Email_F
CONSOMMABLE
N°Cons
NomCons
Type_C
Stock_C
Seuil
Prix_C
Dtemaj
LieuRang

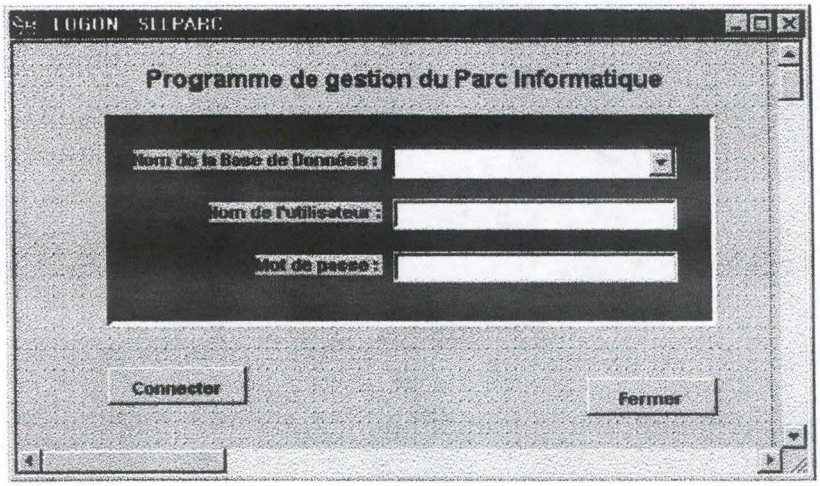

Modèle Conceptuel de Données
Projet : Gestion du Parc Informatique
Modèle: FINAL MCD
Auteur : Service Informatique
Version: 8.0
30/12/99
Se trouver
STUU A8
STUT A10
REPERTOIRE
Id_Rep N5
Père A45
RPUU A8
RPUT A10
Droits d'accès
CodeDroit A10
DAUU A8
DAUT A10
Grouper
Nom_groupe A25
GPUU A8
GPUT A10
UTILISATEUR
Login A8
NomUser A35
Prénom A35
DatEng A10
Service A20
Tél A25
E_mail A65
User_Actif BL
User_Groupe BL
Remarque LBIN
USUU A8
USUT A10
Travail
Actif BL
TLUU A8
TLUT A10
EMPLACEMENT
CodEmpl A4
EMUU A8
EMUT A10
Trouve
Actif BL
TRUU A8
TRUT A10
UTILIS
Actif BL
UTUU A8
UTUT A10
AFFECT
Actif BL
AFUU A8
AFUT A10
FORMATION
NomForm A25
Niveau A20
DureForm A10
LieuForm A35
DateForm A10
Prof A20
Remarque LBIN
FTUU A8
FTUT A10
Se former
SFUU A8
SFUT A10
SOFTWARE
NomSoft A25
TypeSoft A15
Version A6
PrixU N8,2
DtAchat A10
DescSoft A100
Notice LBIN
SWUU A8
SWUT A10
Concerner
LICENCE
NLicence A15
DuréeLicence A8
NbrePoste N5
LCUU A8
LCUT A10
installer
INSTALLATION
Source_Copy A80
Destination_Copy A80
Files A80
Nom_Shotcut A50
INUU A8
INUT A10
HARDWARE
CodeHardware A15
GroupeHardware A15
TypeHardware A25
MarqueHardware A20
DatAchat A10
PrixU N8,2
Disponibilité BL
HWUU A8
HWUT A10
inclure
LEASING
NLeasing A15
DateFinLeasing A10
Garantie A6
Quantité N5
LGUU A8
LGUT A10
Connexion
DateConnexion A8
CXUU A8
CXUT A10
Avoir2
NOTICEHARD
Typehw A20
Notice LBIN
NHUU A8
NHUT A10
Avoir
FORMULAIRE
NumForm I
Dimensions A10
Résolution A10
NbTouche N3
Config A15
MotKeyb A10
Périphér LBIN
Système A60
NbCPU SI
Description LBIN
Cap_DD A10
Type_DD A10
Cap_RAM A10
Type_RAM A10
Bios A15
FMUU A8
FMUT A10
Fournir
INST
Actif BL
ITUU A8
ITUT A10
Fourniture
FOURNISSEUR
NomFourn A25
Contact A25
Adresse A200
TelFour A25
FaxFourn A25
EmailFourn A65
FSUU A8
FSUT A10
Livraison
Consommation
Dte_consom A8
Qte_consom N5
CMUU A8
CMUT A10
CONSOMMABLE
Catégorie_Cons A25
Référence_Cons A25
MarqueCons A20
Stock_Cons I
Seuil I
Prix_Cons N8,2
DateAchat A10
LieuRang A25
CBUU A8
CBUT A10


Modèle Physique de Données
Projet : Gestion du Parc Informatique
Modèle: FINAL MPD
Auteur : Service Informatique
Version: 8.0
30/12/99
SETROUVER
ID_REP
CODEHW
STUU
STUT
REPERTOIRE
ID_REP
PERE
RPUU
RPUT
GROUPER
LOGIN
LOGIN2
NOMGRPE
GPUU
GPUT
EMPLACEMENT
CODEMPL
EMUU
EMUT
TRAVAIL
LOGIN
CODEMPL
ACTIF
TLUU
TLUT
TROUVE
CODEMPL
CODEHW
ACTIF
TRUU
TRUT
HARDWARE
CODEHW
NOMFOURN
NLEASING
GROUPEHW
TYPEHW
MARQUEHW
DATACHAT
PRIXU
DISPO
HWUU
HWUT
LEASING
NLEASING
DTFNLEAS
GARANTIE
QUANTITE
LGUU
LGUT
CONNEXION
CODEHW
CODEHW2
DTCONNEX
CXUU
CXUT
UTILISAT
LOGIN
NOMUSER
PRENOM
DATENG
SERVICE
TEL
E_MAIL
U_ACTIF
U_GPE
REMARK
USUU
USUT
DR_ACCES
LOGIN
ID_REP
CODROIT
DAUU
DAUT
UTILIS
LOGIN
CODEHW
ACTIF
UTUU
UTUT
NOTICEHARD
TYPHW
NOMCODE
NOTICE
NHUU
NHUT
FORMATION
NOM_FORM
NIVEAU
DUREFORM
LIEUFORM
DATEFORM
PROF
REMARK
FTUU
FTUT
SEFORMER
NOMSOFT
NOM_FORM
NIVEAU
LOGIN
SFUU
SFUT
AFFECT
LOGIN
CODESOFT
ACTIF
AFUU
AFUT
INST
CODESOFT
CODEHW
ACTIF
ITUU
ITUT
CONSOMMATION
CATEGORY
RFERENCE
NOMCODE
DTECONS
QTECONS
CMUU
CMUT
FORMULAIRE
NFORMUL
TYPHW
DIMENS
RESOLUT
NBTOUC
CONFIG
MOTKEYB
PERIPHER
SYSTEME
NBCPU
DESCR
CAPDD
TYPEDD
CAPRAM
TYPERAM
BIOS
FMUU
FMUT
SOFTWARE
NOMSOFT
NOMFOURN
NLICENCE
TYPESOFT
VERSION
PRIXU
DTACHAT
DESCSOFT
NOTICE
SWUU
SWUT
LICENCE
NLICENCE
NOMSOFT
DUREELIC
NBRPOSTE
LCUU
LCUT
INSTALLATION
SOURCOPY
DESTCOPY
FILES
NOMSOFT
NOMSHCUT
INUU
INUT
FOURNISSEUR
NOMFOURN
CONTACT
ADRESSE
TELFOURN
FAXFOURN
EMAILFOU
FSUU
FSUT
CONSOMMABLE
CATEGORY
RFERENCE
NOMFOURN
MARQUE
STOCKCON
SEUIL
PRIXCONS
DTEACHAT
LIEURANG
CBUU
CBUT

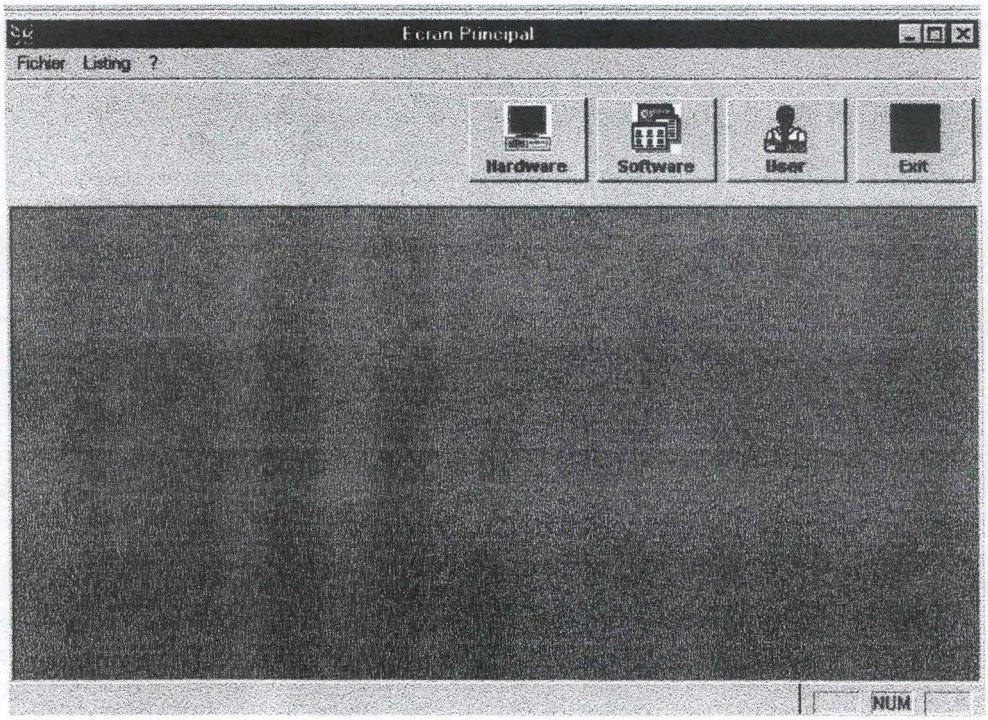
Annexe 3 :

Maquette (Interface Homme - Machine) de SLLParc.

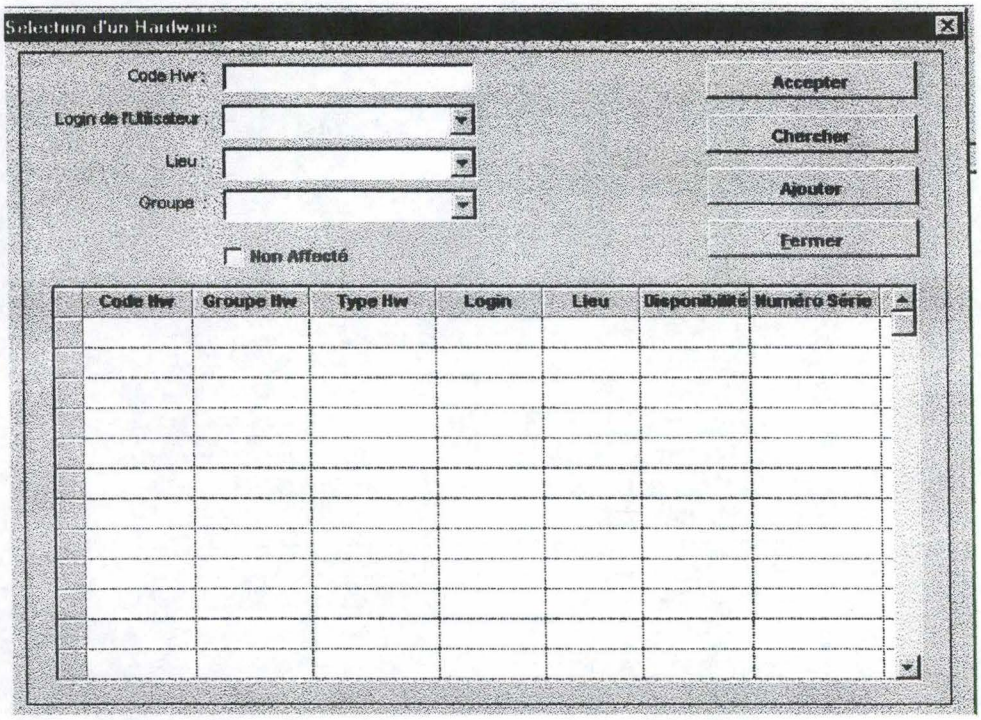
Ecran de Logon de l'application SLLParc



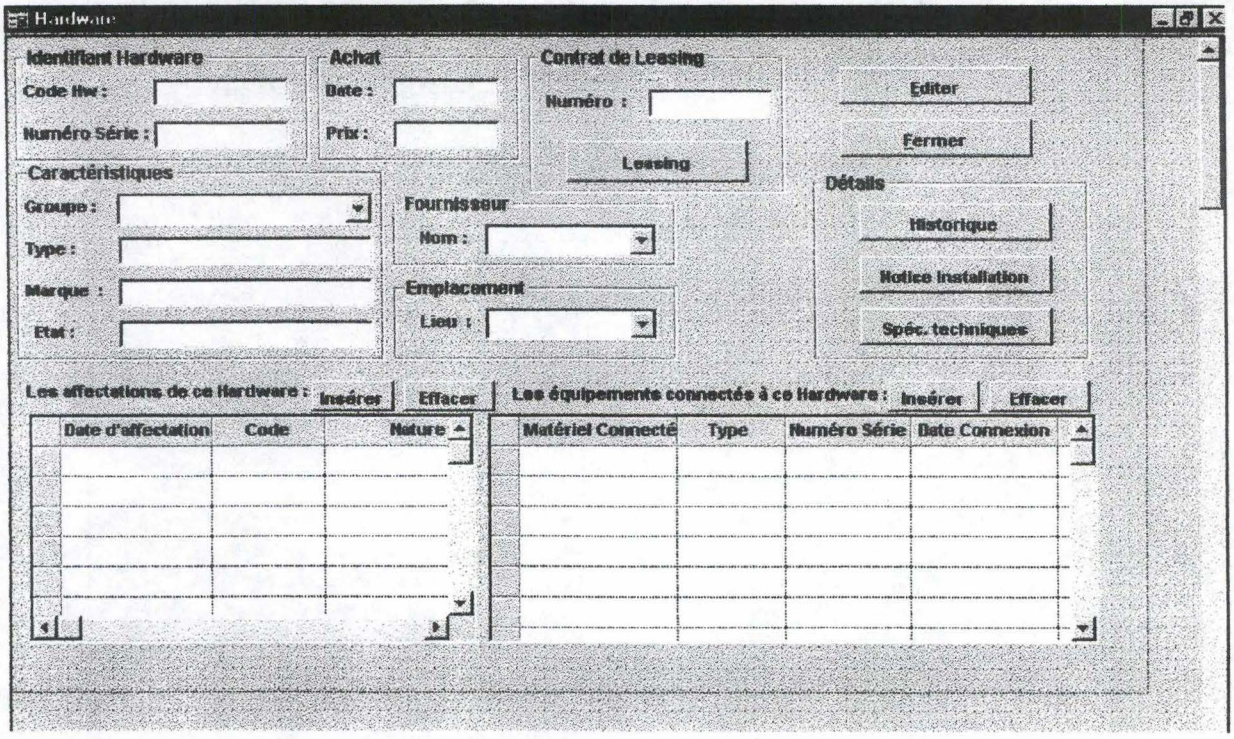
Ecran Principal de l'application SLLParc



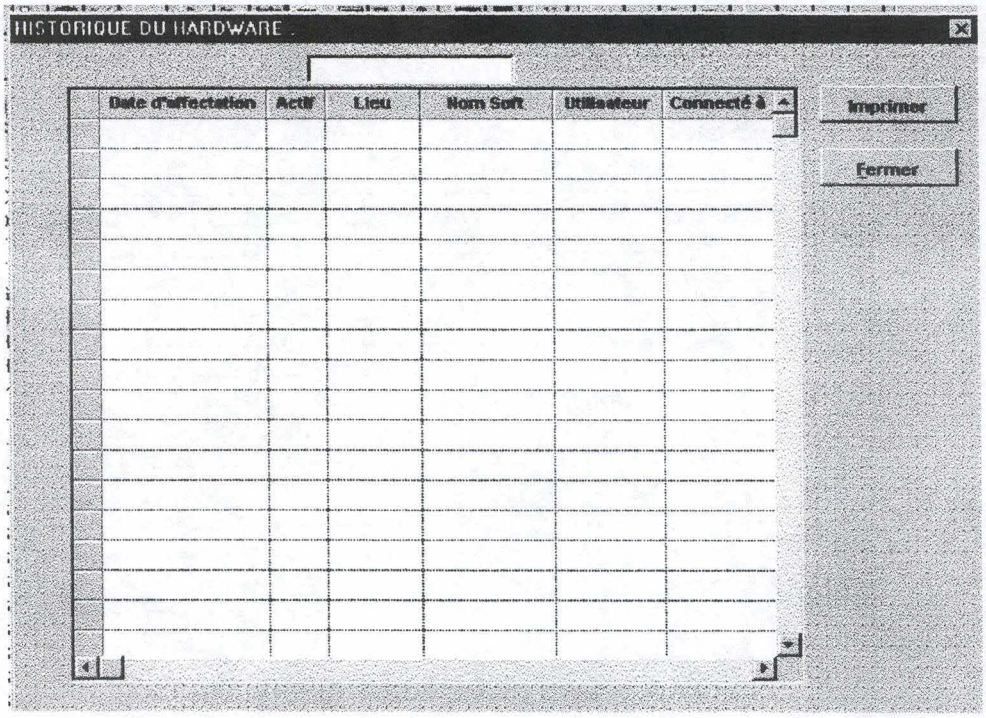
Ecran de sélection d'un hardware



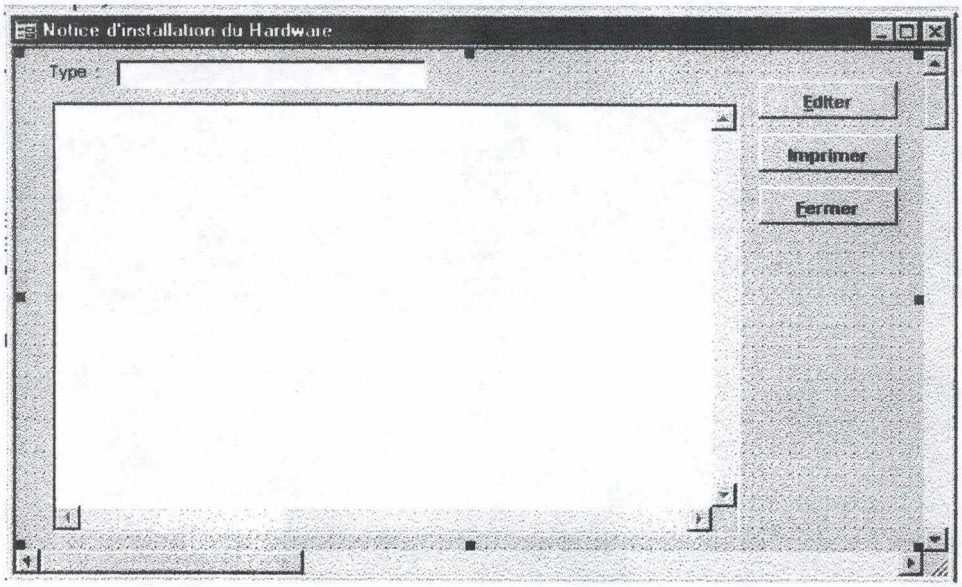
Ecran des données du hardware sélectionné



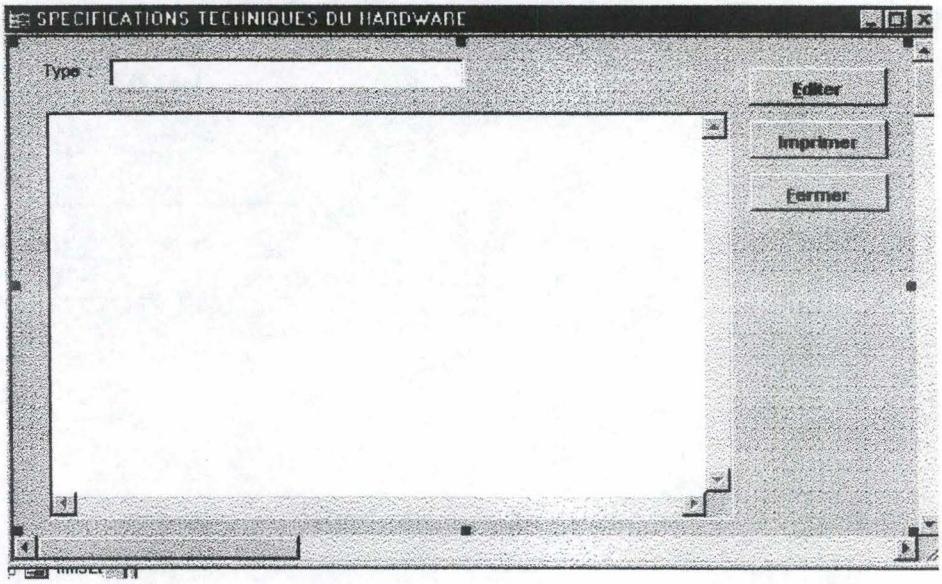
Ecran de l'historique des affectations et connexions du hardware sélectionné



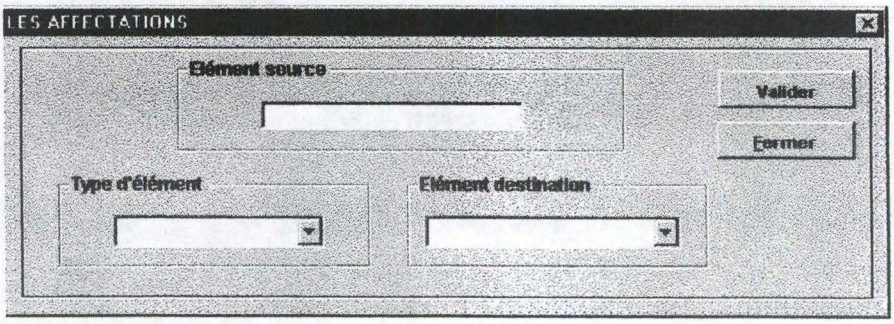
Ecran de la notice d'installation du hardware sélectionné



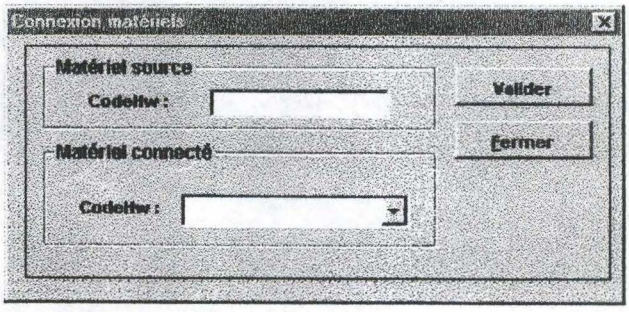
Ecran des spécifications techniques du hardware sélectionné



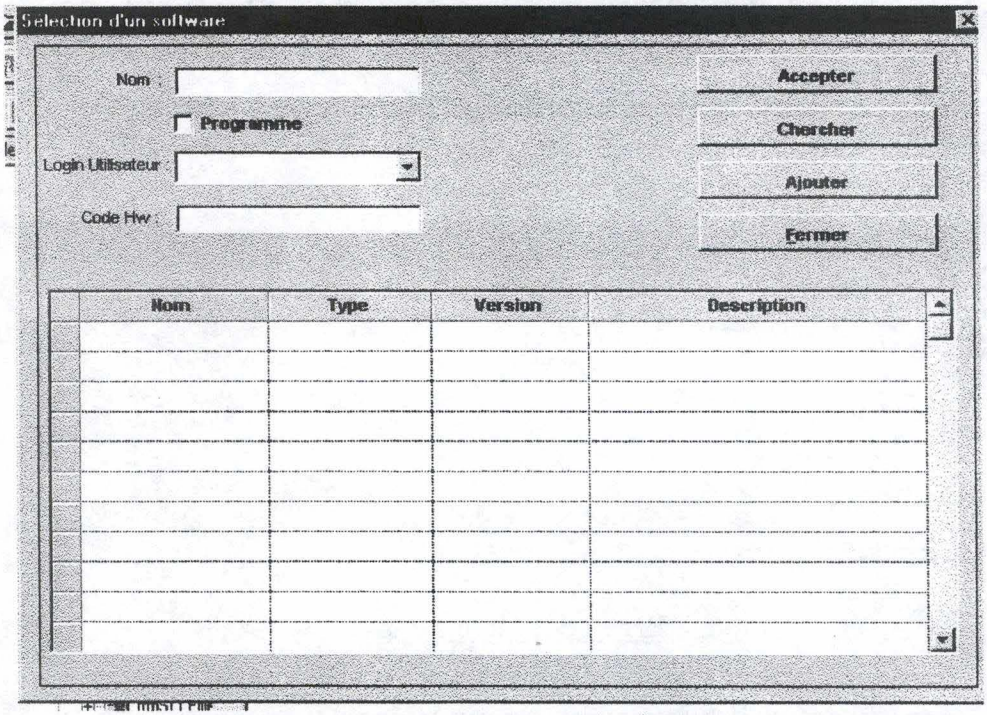
Ecran des affectations



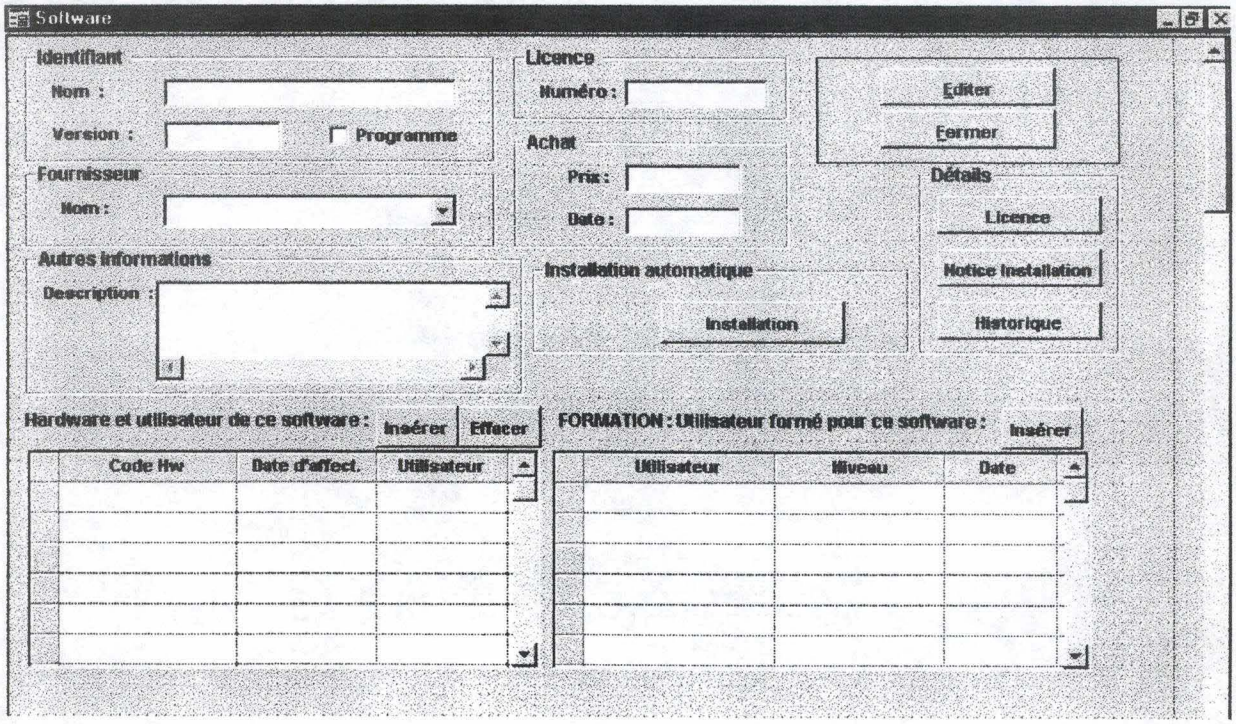
Ecran des connexions des hardwares



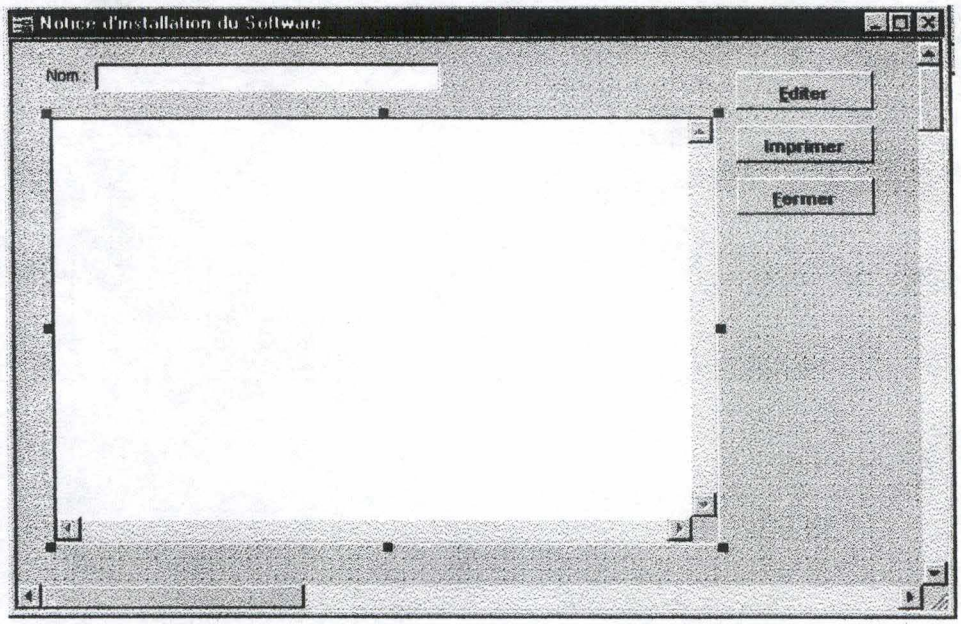
Ecran de sélection d'un software



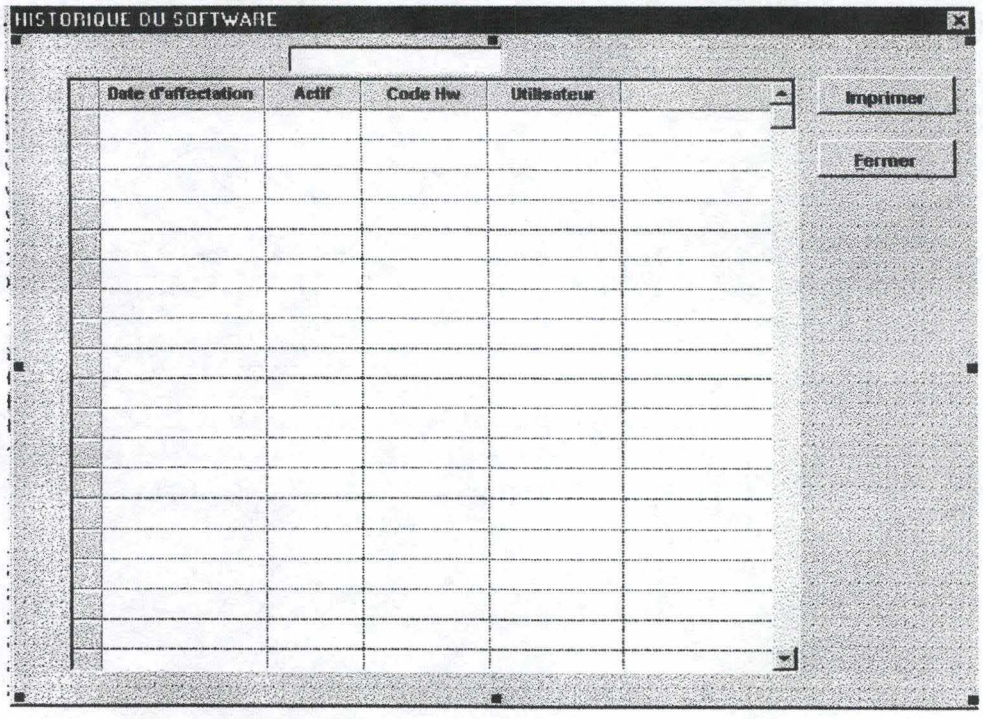
Ecran des données du software sélectionné



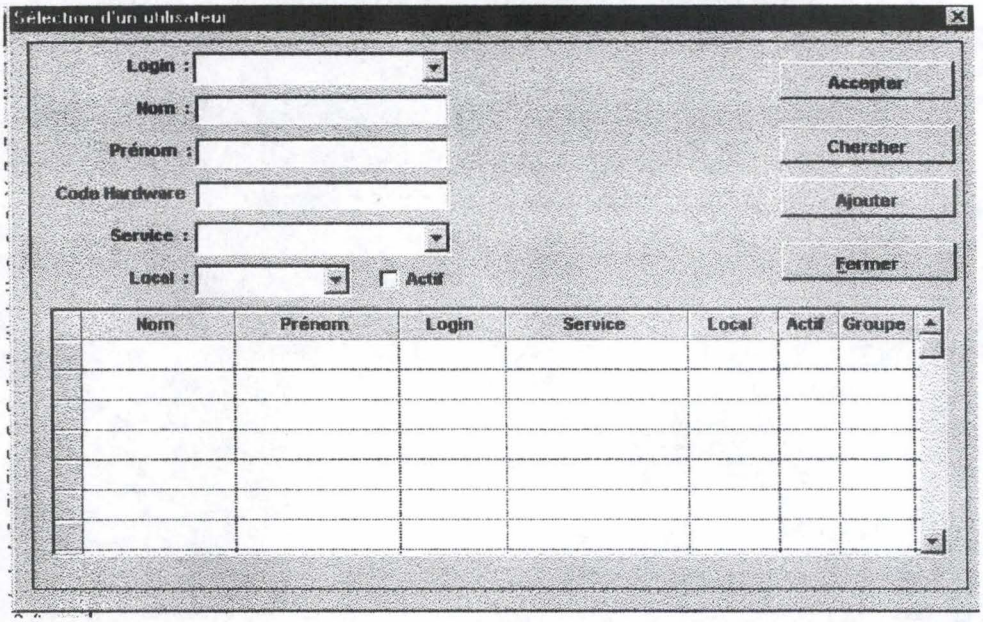
Ecran de notice d'installation d'un software



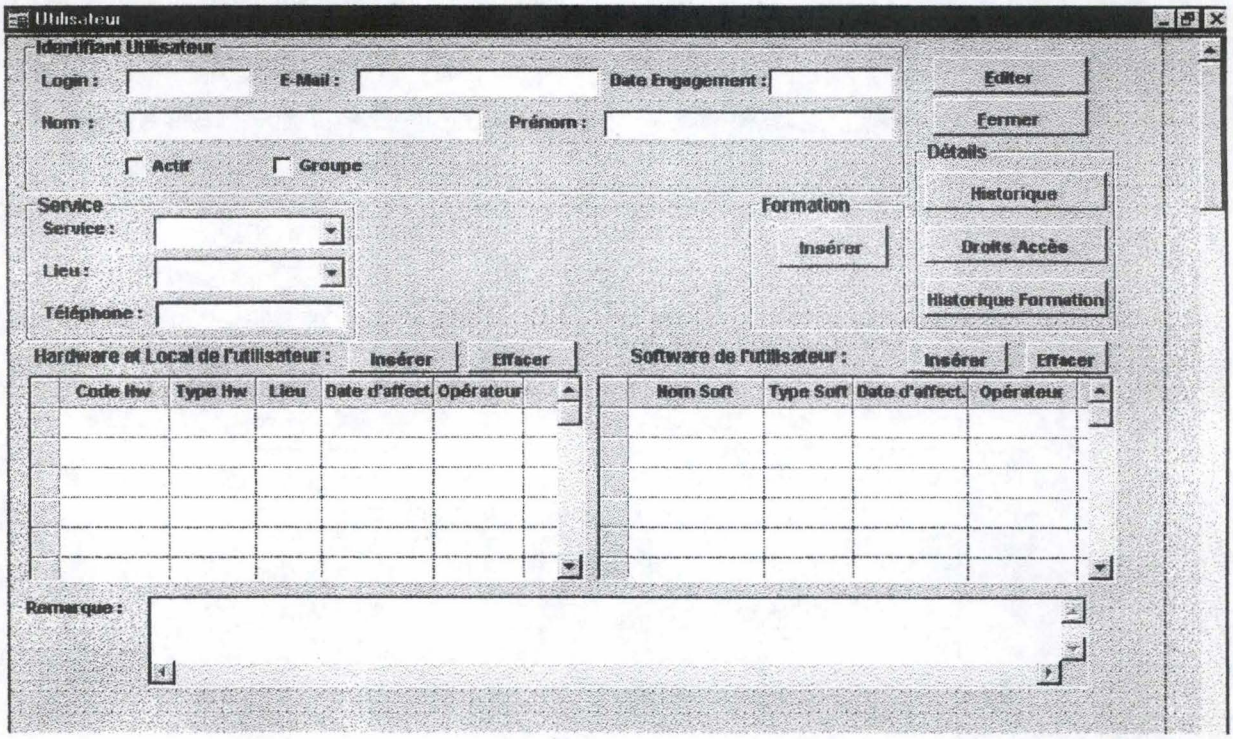
Ecran de l'historique des affectations du software sélectionné



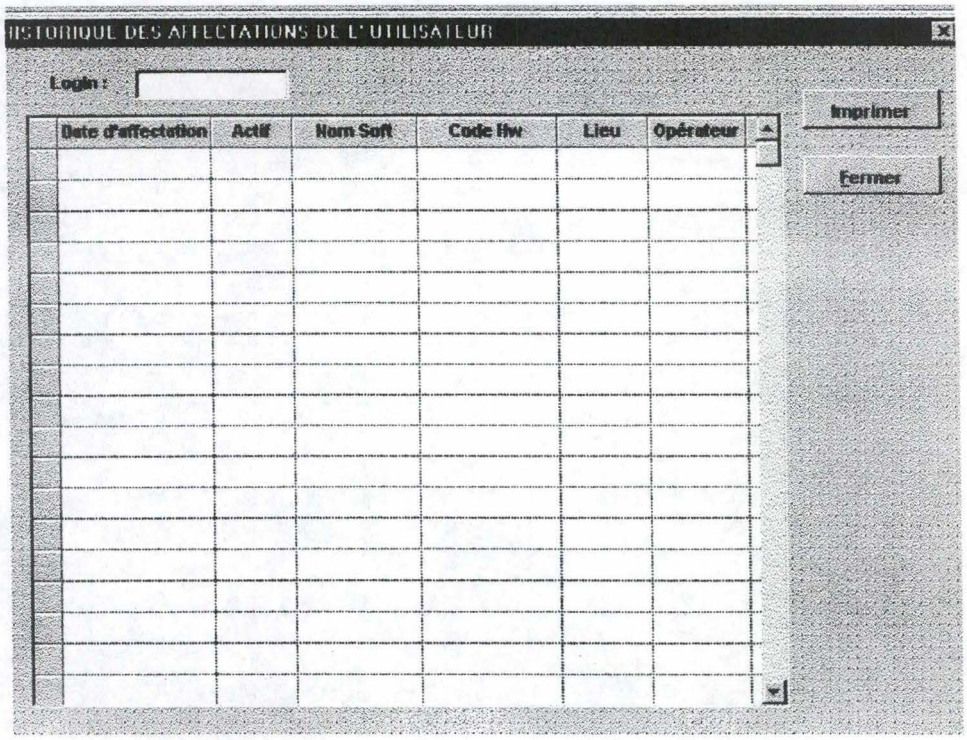
Ecran de sélection d'un utilisateur



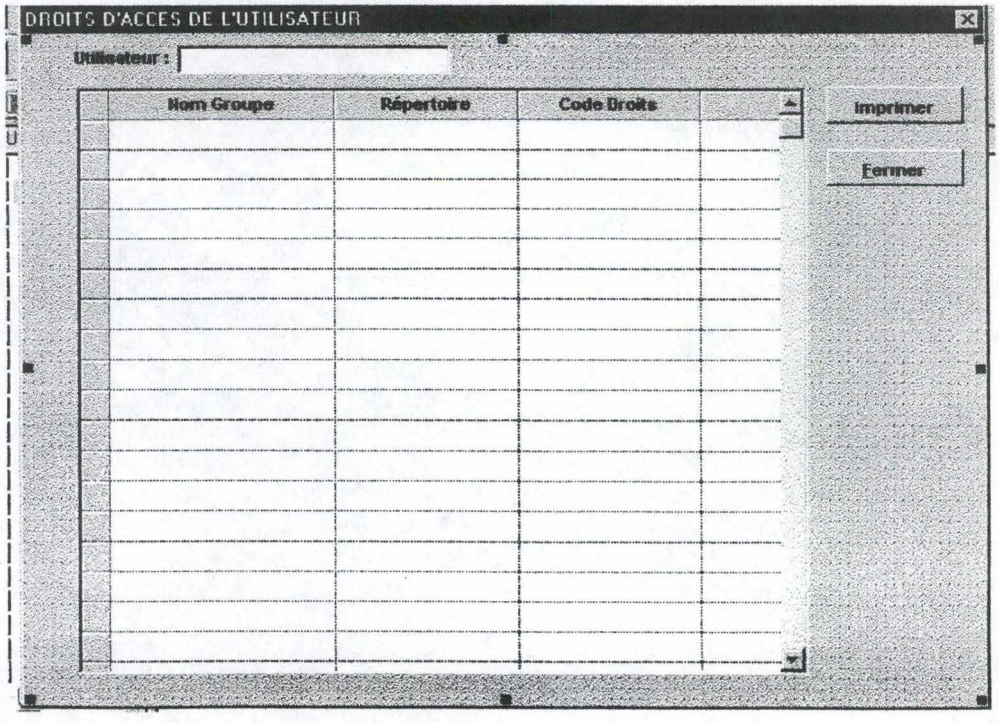
Ecran des données de l'utilisateur sélectionné



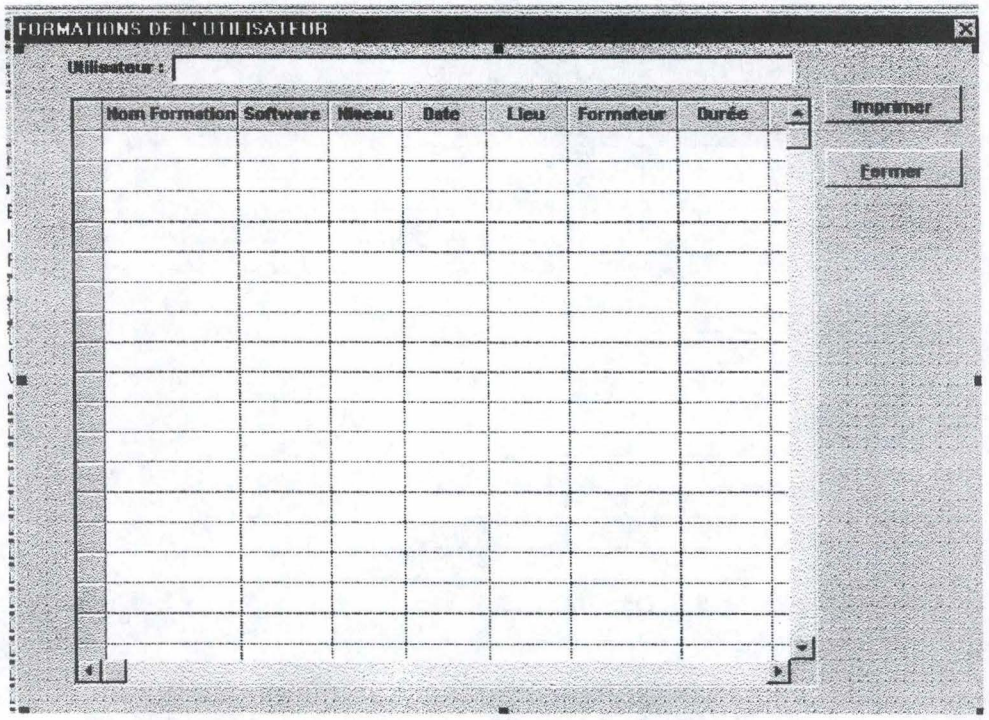
Ecran de l'historique des affectations des soft et hard à l'utilisateur



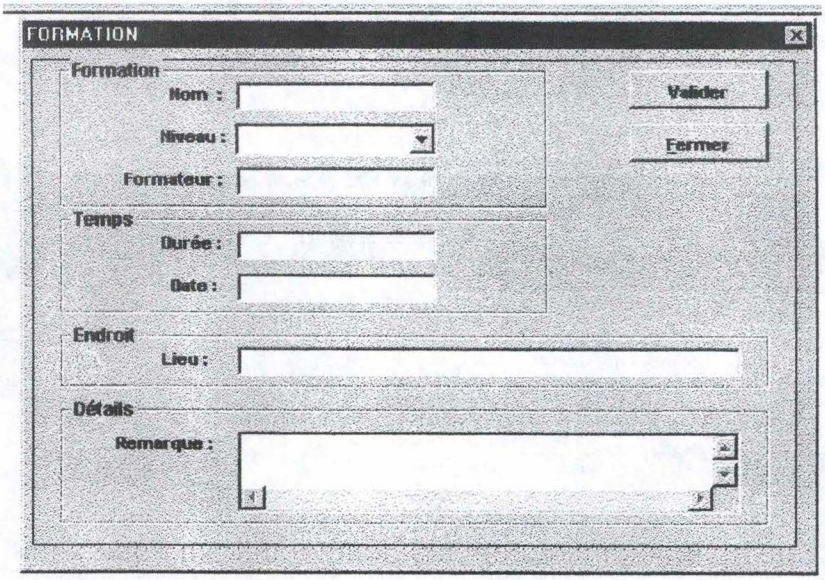
Ecran des droits de l'utilisateur



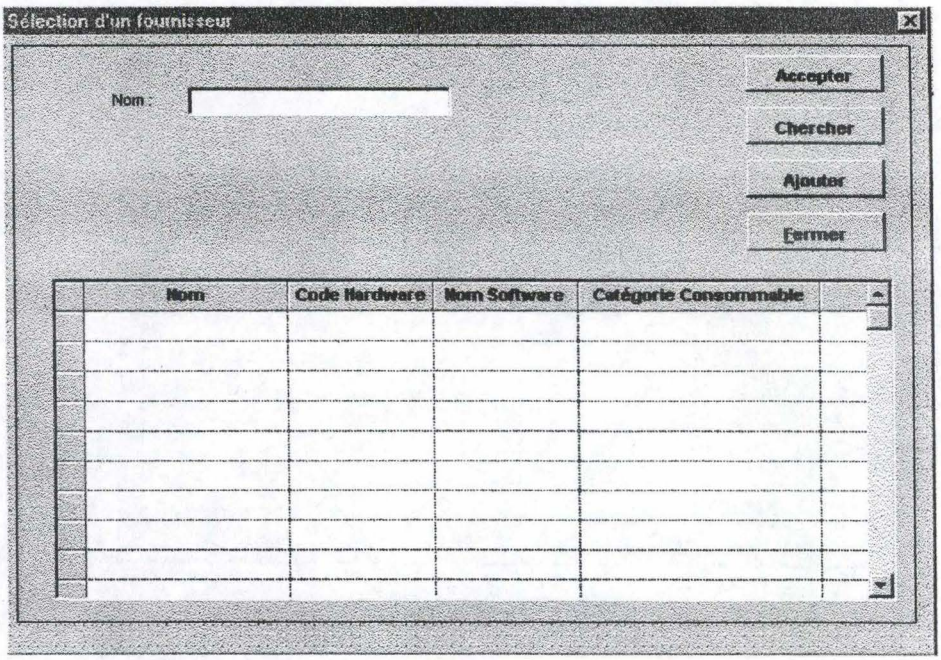
Ecran des formations de l'utilisateur



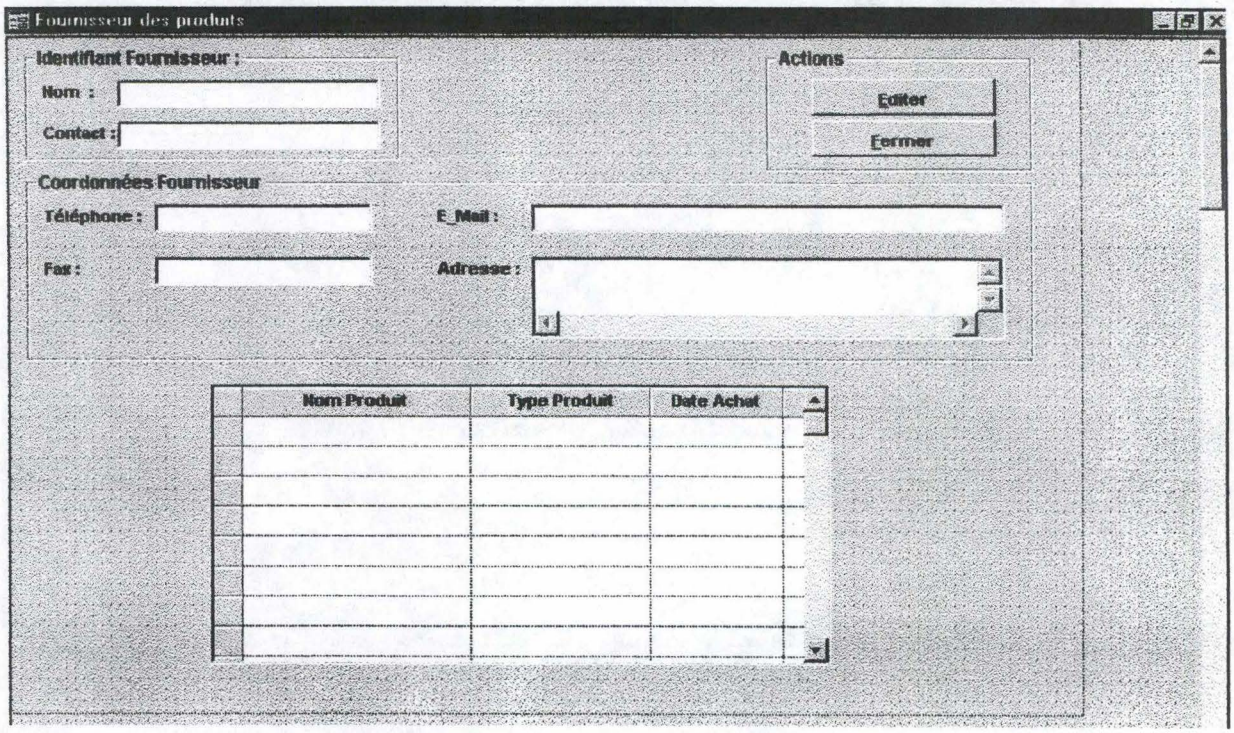
Ecran de sasie de formation de l'utilisateur



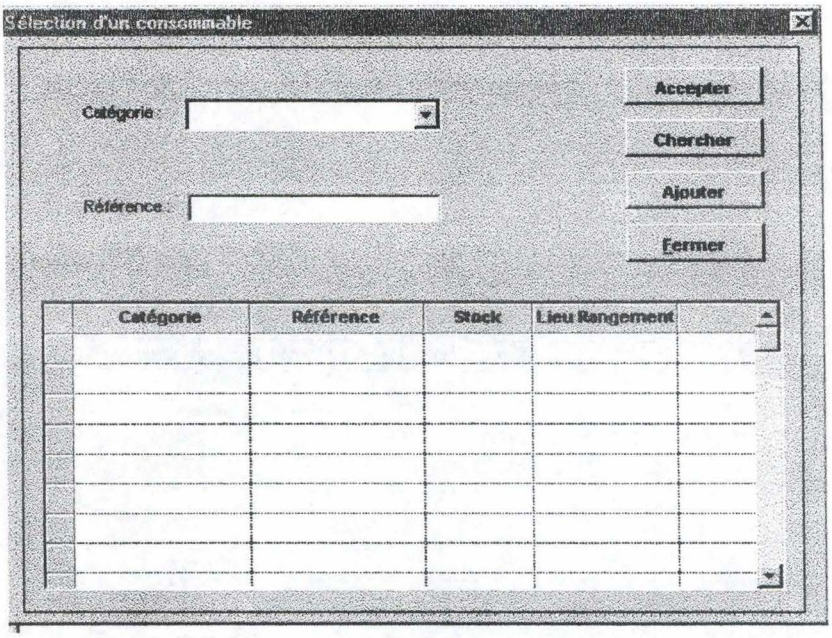
Ecran de sélection d'un fournisseur



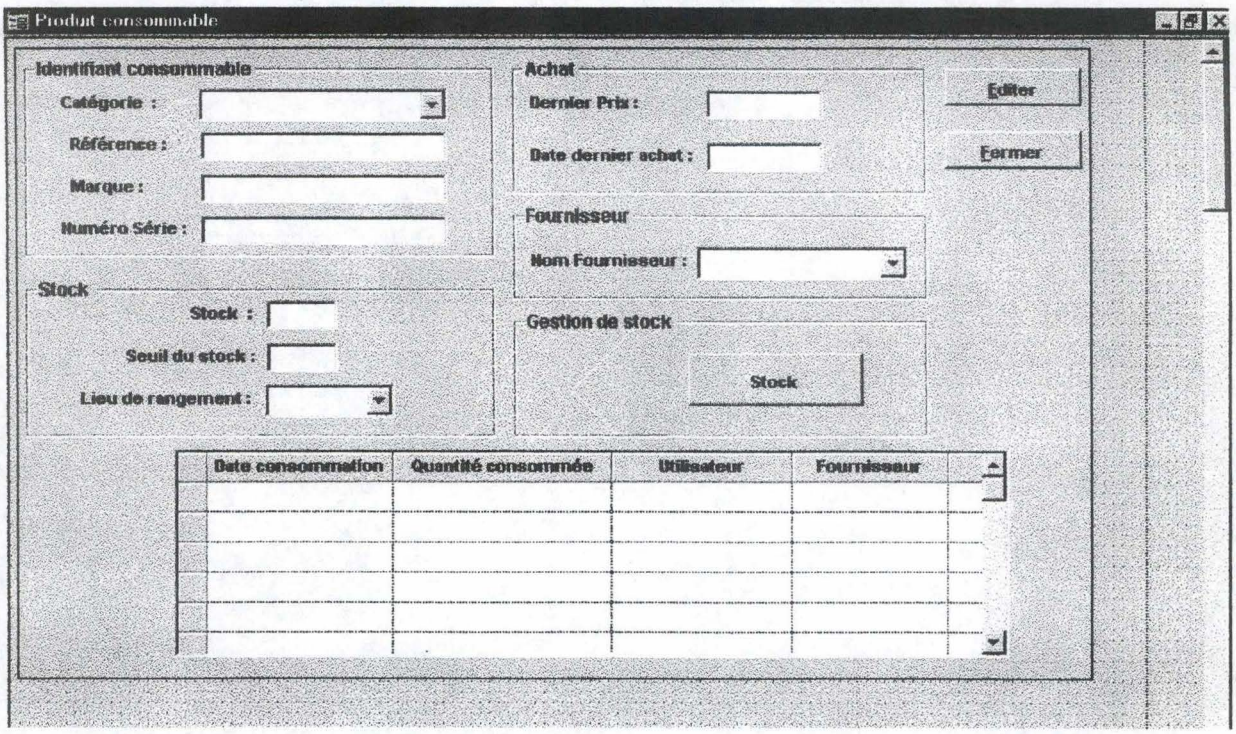
Ecran des données du fournisseur sélectionné



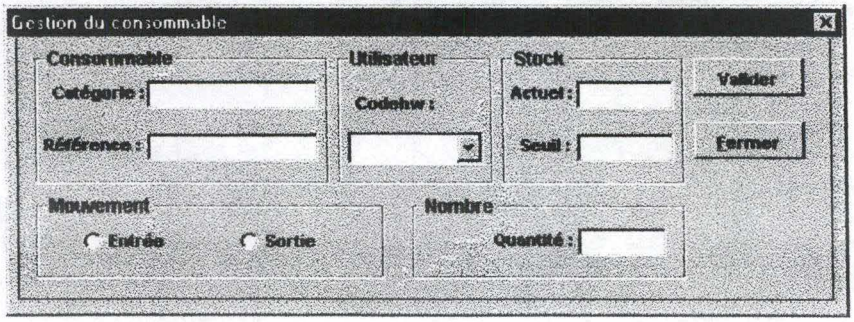
Ecran de Sélection d'un produit consommable



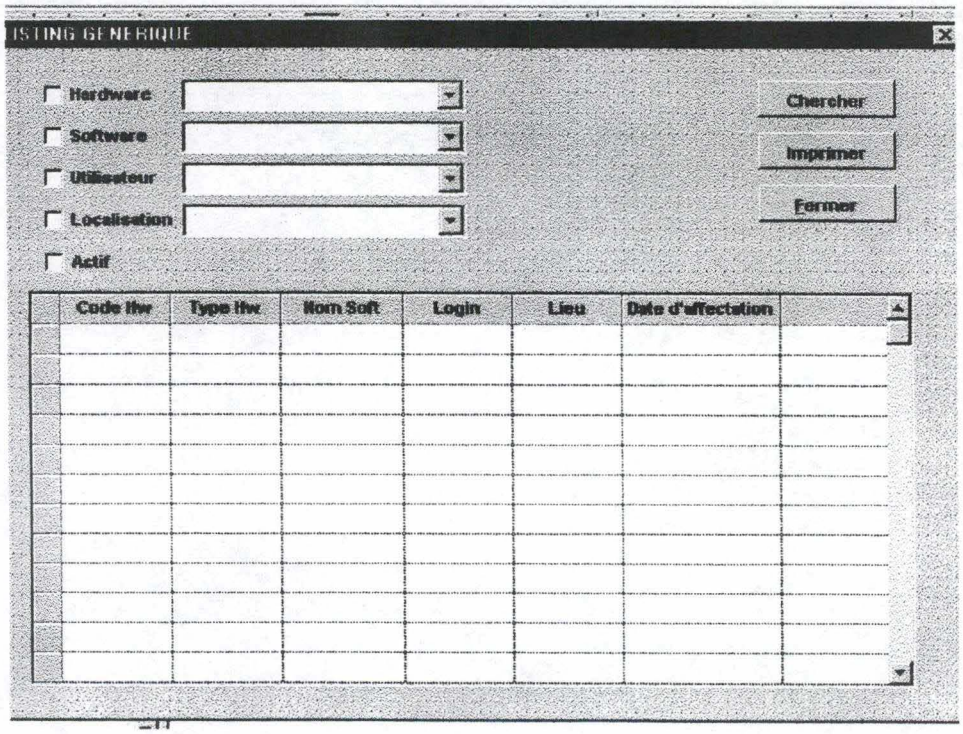
Ecran des données d'un produit consommable sélectionné



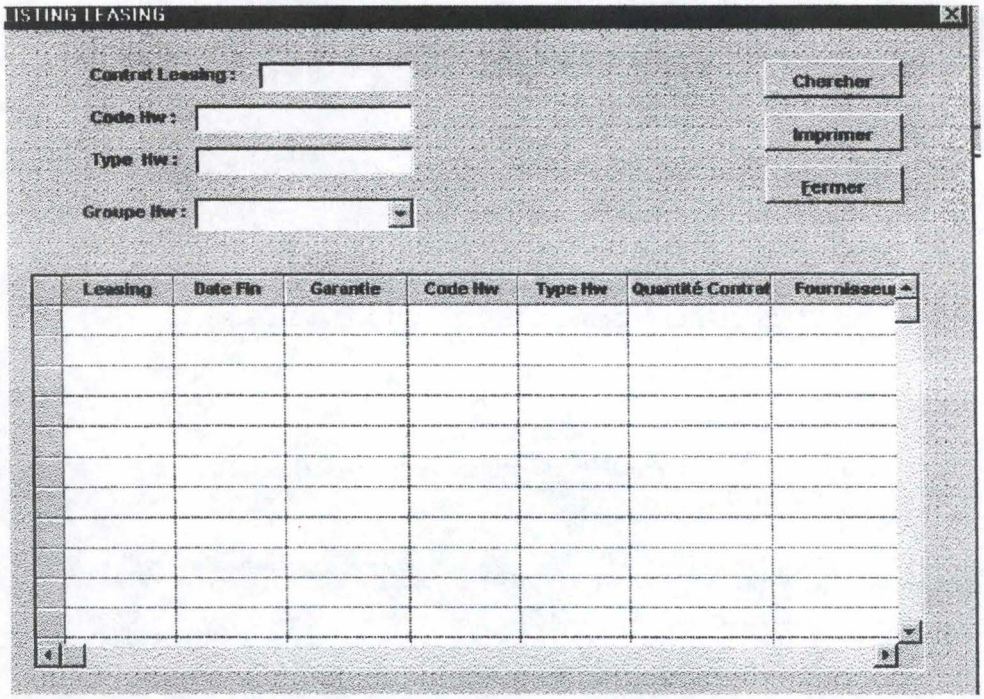
Ecran de gestion de stock du produit consommable sélectionné



Ecran de listing générique



Ecran de listing de leasing de Hardware



Annexe 4 :

Script de la création de la base de données de SLLParc.

```
create table AFFECT
(
   LOGIN                CHAR(8)         not null,
   CODESW               CHAR(25)        not null,
   ACTIF                NUMBER(1)             ,
   AFUU                 CHAR(8)               ,
   AFUT                 CHAR(10)
);
create unique index AFFECT_PK on AFFECT (ACTIF asc, LOGIN asc, CODESW asc);
create index AFFECT_PK1 on AFFECT (ACTIF asc, LOGIN asc);
create index AFFECT_PK2 on AFFECT (ACTIF asc, CODESW asc);

create table CONNEXION
(
   CODEHW               CHAR(15)        not null,
   CODEHW2              CHAR(15)        not null,
   DTCONNEX             CHAR(8)               ,
   CXUU                 CHAR(8)               ,
   CXUT                 CHAR(10)
);
create unique index CONNEXION_PK on CONNEXION (CODEHW asc, CODEHW2 asc);
create index CONNEXION_PK1 on CONNEXION (CODEHW asc);
create index CONNEXION_PK2 on CONNEXION (CODEHW2 asc);

create table CONSOMMABLE
(
   CATEGORY             CHAR(25)        not null,
   RFERENCE             CHAR(25)        not null,
   NOMFOURN             CHAR(25)        not null,
   NSERIE               CHAR(20)              ,
   MARQUE               CHAR(20)              ,
   STOCKCON             INTEGER         not null,
   SEUIL                INTEGER         not null,
   PRIXCONS             NUMBER(8,2)           ,
   DTEACHAT             CHAR(10)              ,
   LIEURANG             CHAR(25)              ,
   CBUU                 CHAR(8)               ,
   CBUT                 CHAR(10)
);
create unique index CONSOMMABLE_PK on CONSOMMABLE (CATEGORY asc,
RFERENCE asc);
create index CONSOMMABLE_PK1 on CONSOMMABLE (CATEGORY asc);
create index CONSOMMABLE_PK2 on CONSOMMABLE (RFERENCE asc);
create index CONSOMMABLE_FK1 on CONSOMMABLE (NOMFOURN asc);
```

```
create index CONSOMMABLE_FK2 on CONSOMMABLE (NSERIE asc);
create table CONSOMMATION
(
   CATEGORY                CHAR(25)          not null,
   RFERENCE                CHAR(25)          not null,
   CODEHW                  CHAR(15)          not null,
   DTECONS                 CHAR(8)                ,
   QTECONS                 NUMBER(5)          not null,
   CMUU                    CHAR(8)                ,
   CMUT                    CHAR(10)
);
create unique index CONSOMMATION_PK on CONSOMMATION (CATEGORY asc,
RFERENCE asc, CODEHW asc);
create index CONSOMMATION_P1 on CONSOMMATION (CATEGORY asc);
create index CONSOMMATION_PK2 on CONSOMMATION (RFERENCE asc);
create index CONSOMMATION_PK3 on CONSOMMATION (CODEHW asc);

create table DRACCES
(
   LOGIN                   CHAR(8)           not null,
   ID_REP                  NUMBER(5)          not null,
   CODROIT                  CHAR(10)               ,
   DAUU                    CHAR(8)                ,
   DAUT                    CHAR(10)
);
create unique index DRACCES_PK on DRACCES (LOGIN asc, ID_REP asc);

create table EMPLACEMENT
(
   CODEMPL                  CHAR(4)           not null,
   EMUU                    CHAR(8)                ,
   EMUT                    CHAR(10)
);
create unique index EMPLACEMENT_PK on EMPLACEMENT (CODEMPL asc);

create table FORMATION
(
   NOMFORM                  CHAR(25)           not null,
   NIVEAU                  CHAR(20)          not null,
   DUREFORM                 CHAR(10)               ,
   LIEUFORM                 CHAR(35)               ,
   DATEFORM                 CHAR(10)               ,
   PROF                    CHAR(20)               ,
   REMARK                   LONG RAW               ,
   FTUU                    CHAR(8)                ,
```

```
   FTUT                 CHAR(10)
);
create unique index FORMATION_PK on FORMATION (NOMFORM asc, NIVEAU asc);
create index FORMATION_PK1 on FORMATION (NOMFORM asc);
create index FORMATION_PK2 on FORMATION (NIVEAU asc);

create table FORMULAIRE
(
   NFORMUL              INTEGER          not null,
   TYPHW                CHAR(20)         not null,
   DIMENS               CHAR(10)                 ,
   RESOLUT              CHAR(10)                 ,
   NBTOUC               NUMBER(3)                ,
   CONFIG               CHAR(15)                 ,
   MOTKEYB              CHAR(10)                 ,
   PERIPHER             LONG RAW                 ,
   SYSTEME              CHAR(60)                 ,
   NBCPU                SMALLINT                 ,
   DESCR                LONG RAW                 ,
   CAPDD                CHAR(10)                 ,
   TYPEDD               CHAR(10)                 ,
   CAPRAM               CHAR(10)                 ,
   TYPERAM              CHAR(10)                 ,
   BIOS                 CHAR(15)                 ,
   FMUU                 CHAR(8)                  ,
   FMUT                 CHAR(10)
);
create unique index FORMULAIRE_PK on FORMULAIRE (NFORMUL asc);
create index FORMULAIRE_FK1 on FORMULAIRE (TYPHW asc);

create table FOURNISSEUR
(
   NOMFOURN             CHAR(25)         not null,
   CONTACT              CHAR(25)                 ,
   ADRESSE              CHAR(200)                ,
   TELFOURN             CHAR(25)                 ,
   FAXFOURN             CHAR(25)                 ,
   EMAILFOU             CHAR(65)                 ,
   FSUU                 CHAR(8)                  ,
   FSUT                 CHAR(10)
);
create unique index FOURNISSEUR_PK on FOURNISSEUR (NOMFOURN asc);
```

```
create table GROUPER
(
   LOGIN                CHAR(8)          not null,
   LOGIN2               CHAR(8)          not null,
   NOMGRPE              CHAR(25)              ,
   GPUU                 CHAR(8)               ,
   GPUT                 CHAR(10)
);
create unique index GROUPER_PK on GROUPER (LOGIN asc, LOGIN2 asc);

create table HARDWARE
(
   CODEHW               CHAR(15)         not null,
   NOMFOURN             CHAR(25)         not null,
   NSERIE               CHAR(20)         not null,
   NLEASING             CHAR(15)              ,
   GROUPEHW             CHAR(15)         not null,
   TYPEHW               CHAR(25)              ,
   MARQUEHW             CHAR(20)              ,
   DATACHAT             CHAR(10)              ,
   PRIXU                NUMBER(8,2)           ,
   DISPO                NUMBER(1)        not null,
   HWUU                 CHAR(8)               ,
   HWUT                 CHAR(10)
);
create unique index HARDWARE_PK on HARDWARE (CODEHW asc);
create index HARDWARE_FK1 on HARDWARE (NOMFOURN asc);
create index HARDWARE_FK2 on HARDWARE (NLEASING asc);
create index HARDWARE_FK2 on HARDWARE (NSERIE asc);

create table INST
(
   CODESW               CHAR(25)         not null,
   CODEHW               CHAR(15)         not null,
   ACTIF                NUMBER(1)             ,
   ITUU                 CHAR(8)               ,
   ITUT                 CHAR(10)
);
create unique index INST_PK on INST (ACTIF asc, CODESW asc, CODEHW asc);
create index INST_PK1 on INST (ACTIF asc, CODESW asc);
create index INST_PK2 on INST (ACTIF asc, CODEHW asc);
```

```
create table INSTALLATION
(
   SOURCOPY                CHAR(80)          not null,
   DESTCOPY                CHAR(80)          not null,
   FILES                   CHAR(80)          not null,
   NOMSOFT                 CHAR(25)                ,
   NOMSHCUT                CHAR(50)                ,
   INUU                    CHAR(8)                 ,
   INUT                    CHAR(10)
);
create unique index INSTALLATION_PK on INSTALLATION (SOURCOPY asc,
DESTCOPY asc, FILES asc);
create index INSTALLATION_FK1 on INSTALLATION (NOMSOFT asc);

create table LEASING
(
   NLEASING                CHAR(15)          not null,
   DTFNLEAS                CHAR(10)                ,
   GARANTIE                CHAR(6)                 ,
   QUANTITE                NUMBER(5)               ,
   LGUU                    CHAR(8)                 ,
   LGUT                    CHAR(10)
);
create unique index LEASING_PK on LEASING (NLEASING asc);

create table LICENCE
(  NLICENCE                CHAR(15)          not null,
   NOMSOFT                 CHAR(25)          not null,
   DUREELIC                CHAR(8)                 ,
   NBRPOSTE                NUMBER(5)               ,
   LCUU                    CHAR(8)                 ,
   LCUT                    CHAR(10)
);
create unique index LICENCE_PK on LICENCE (NLICENCE asc);
create index LICENCE_FK1 on LICENCE (NOMSOFT asc);

create table NOTICEHARD
(
   TYPHW                   CHAR(20)          not null,
   NOMCODE                 CHAR(15)          not null,
   NFORMUL                 INTEGER                 ,
   NOTICE                  LONG RAW                ,
   NHUU                    CHAR(8)                 ,
   NHUT                    CHAR(10)
```

```
);
create unique index NOTICEHARD_PK on NOTICEHARD (TYPHW asc);
create index NOTICEHARD_FK1 on NOTICEHARD (NOMCODE asc);
create index NOTICEHARD_FK2 on NOTICEHARD (NFORMUL asc);

create table REPERTOIRE
(
   ID_REP                NUMBER(5)        not null,
   PERE                  CHAR(45)              ,
   RPUU                  CHAR(8)               ,
   RPUT                  CHAR(10)
);
create unique index REPERTOIRE_PK on REPERTOIRE (ID_REP asc);

create table SEFORMER
(
   NOMSOFT               CHAR(25)         not null,
   NOMFORM               CHAR(25)         not null,
   NIVEAU                CHAR(20)         not null,
   LOGIN                 CHAR(8)          not null,
   SFUU                  CHAR(8)               ,
   SFUT                  CHAR(10)
);
create index SEFORMER_PK on SEFORMER (LOGIN asc);
create index SEFORMER_PK1 on SEFORMER (NOMSOFT asc);
create index SEFORMER_PK2 on SEFORMER (NOMFORM asc);
create index SEFORMER_PK3 on SEFORMER (NIVEAU asc);

create table SETROUVER
(  ID_REP                NUMBER(5)         not null,
   CODEHW                CHAR(15)         not null,
   STUU                  CHAR(8)               ,
   STUT                  CHAR(10)
);
create unique index SETROUVER_PK on SETROUVER (ID_REP asc, CODEHW asc);

create table SOFTWARE
(
   NOMSOFT               CHAR(25)         not null,
   NOMFOURN              CHAR(25)         not null,
   NLICENCE              CHAR(15)              ,
   TYPESOFT              CHAR(15)         not null,
   VERSION               CHAR(6)               ,
   PRIXU                 NUMBER(8,2)           ,
   DTACHAT               CHAR(10)              ,
```

```
  DESCSOFT              CHAR(100)          ,
  NOTICE                LONG RAW           ,
  SWUU                  CHAR(8)            ,
  SWUT                  CHAR(10)
);
create unique index SOFTWARE_PK on SOFTWARE (NOMSOFT asc);
create index SOFTWARE_FK1 on SOFTWARE (NOMFOURN asc);
create index SOFTWARE_FK2 on SOFTWARE (NLICENCE asc);

create table TRAVAIL
(
  LOGIN                 CHAR(8)            not null,
  CODEMPL               CHAR(4)            not null,
  ACTIF                 NUMBER(1)          ,
  TLUU                  CHAR(8)            ,
  TLUT                  CHAR(10)
);
create unique index TRAVAIL_PK on TRAVAIL (ACTIF asc, LOGIN asc, CODEMPL asc);
create index TRAVAIL_PK1 on TRAVAIL (ACTIF asc, CODEMPL asc);
create index TRAVAIL_PK2 on TRAVAIL (ACTIF asc, LOGIN asc,);

create table TROUVE
(
  CODEMPL               CHAR(4)            not null,
  CODEHW                CHAR(15)           not null,
  ACTIF                 NUMBER(1)          ,
  TRUU                  CHAR(8)            ,
  TRUT                  CHAR(10)
);
create unique index TROUVE_PK on TROUVE (ACTIF asc, CODEMPL asc, CODEHW
asc);
create index TROUVE_PK1 on TROUVE (ACTIF asc, CODEMPL asc);
create index TROUVE_PK2 on TROUVE (ACTIF asc, CODEHW asc);

create table UTILIS
(
  LOGIN                 CHAR(8)            not null,
  CODEHW                CHAR(15)           not null,
  ACTIF                 NUMBER(1)          ,
  UTUU                  CHAR(8)            ,
  UTUT                  CHAR(10)
);
create unique index UTILIS_PK on UTILIS (ACTIF asc, LOGIN asc, CODEHW asc);
```

```
create  index UTILIS_PK1 on UTILIS (ACTIF asc, LOGIN asc);
create  index UTILIS_PK2 on UTILIS (ACTIF asc, CODEHW asc);

create table UTILISAT
(
   LOGIN                CHAR(8)          not null,
   NOMUSER                CHAR(35)          not null,
   PRENOM               CHAR(35)                ,
   DATENG               CHAR(10)                ,
   SERVICE              CHAR(20)                ,
   TEL                 CHAR(25)                 ,
   EMAIL                CHAR(65)                ,
   ACTIF                NUMBER(1)          not null,
   GROUPE                NUMBER(1)          not null,
   REMARK               LONG RAW                ,
   USUU                 CHAR(8)                  ,
   USUT                 CHAR(10)
);
create unique index UTILISAT_PK on UTILISAT (LOGIN asc);
create index UTILISAT_PK1 on UTILISAT (NOMUSER asc);
```

Annexe 5 :

Codes de programme de SLLParc.

Ici nous donnons les codes Centura écrits derrière quelques uns de boutons de commande de l'application SLLParc.

a) Code derrière le bouton de commande **Connecter** de l'écran de Login.

```
Pushbutton: pbConnecter
        Class Child Ref Key: 0
        Class ChildKey: 0
        Class: clpbSimple
        Property Template:
        Class DLL Name:
        Title: Connecter
        Window Location and Size
                Left:  0.983"
                Top:  2.571"
                Width:  1.5"
                Width Editable? Class Default
                Height: Class Default
                Height Editable? Class Default
        Visible? Class Default
        Keyboard Accelerator: Class Default
        Font Name: Class Default
        Font Size: Class Default
        Font Enhancement: Class Default
        Picture File Name:
        Picture Transparent Color: Class Default
        Image Style: Class Default
        Text Color: Class Default
        Background Color: Class Default
        Message Actions
                On SAM_Click
                        !
                        Call SalWaitCursor( TRUE )
                        ! Set SqlUser='SYSADM'
                        ! Set SqlPassword='SYSADM'
                        !
                        If SalStrLength( SalStrTrimX( dfUserName ) )>0
                                Set SqlUser=SalStrTrimX(  dfUserName )
                        If SalStrLength( SalStrTrimX( dfPassWord ) )>0
                                Set SqlPassword=SalStrTrimX( dfPassWord )
                        Set SqlDatabase=SalStrTrimX(cmbDBName )
                        !
                        If SLLConnectAll ()
                                !
                                Call SalDestroyWindow( frmLoginSLLParc )
```

```
            !
            Call SalCreateWindow( frmMdiPrincipal , hWndNULL )
            !
    !
    Call SalWaitCursor( FALSE )
    !
```

b) Code derrière le bouton de commande **Fermer** de l'écran de Login.

Aucun code n'y a été écrit car tous les boutons de commande **Fermer** appartiennent à une classe et ont pour action de fermer la fenêtre qui le contient. Classe prédéfinie dans la librairie propre de Swiss Life.

c) Code derrière le bouton de commande **Hardware** de l'écran principal pour ouvrir son écran de sélection.

```
Pushbutton: pbHardware
    Class Child Ref Key: 0
    Class ChildKey: 0
    Class: clpbToolBar
    Property Template:
    Class DLL Name:
    Title: Hardware
    Window Location and Size
        Left:  4.983"
        Top:  0.155"
        Width:  1.3"
        Width Editable? Class Default
        Height: 0.667"
        Height Editable? Class Default
    Visible? Class Default
    Keyboard Accelerator: Class Default
    Font Name: Class Default
    Font Size: Class Default
    Font Enhancement: Class Default
    Picture File Name: N:\PROGRAM\Utilitaires\ICONS\Computer.ico
    Picture Transparent Color: Class Default
    Image Style: Class Default
    Text Color: Class Default
    Background Color: Class Default
    Message Actions
        On SAM_Click
            Call SalModalDialog ( dlgSelectionHardware, hWndForm )
```

d) Code derrière le bouton de commande **Accepter** de l'écran Sélection.

```
Pushbutton: pbAccepterSelectionHw
        Class Child Ref Key: 0
        Class ChildKey: 0
        Class: clpbDialogBoxAccepter
        Property Template:
        Class DLL Name:
        Title:
        Window Location and Size
                Left:  7.683"
                Top:  0.155"
                Width:  2.3"
                Width Editable? Class Default
                Height: Class Default
                Height Editable? Class Default
        Visible? Class Default
        Keyboard Accelerator: Class Default
        Font Name: Class Default
        Font Size: Class Default
        Font Enhancement: Class Default
        Picture File Name:
        Picture Transparent Color: Class Default
        Image Style: Class Default
        Text Color: Class Default
        Background Color: Class Default


        Message Actions
                On PAM_Confirmer
                        !
                        Set smCODEHW=dfCodeHw
                        Set frmMdiPrincipal.dfIdentifiantEnCours=smCODEHW
                        !
                        Call SqlExists( 'select CodeHw from '||SLLDB||'Hardware where
CodeHw=:smCODEHW ', bExists )
                        If not bExists
                                Set bValid=FALSE
                        !
                On PAM_Modifier
                        !
                        If bfrmSLLFilleHw
                                !
```

```
                    Call SalBringWindowToTop( frmMdiPrincipal.frmSLLFilleHw
)
                    Call          SalSendMsg(frmMdiPrincipal.frmSLLFilleHw,
SAM_Create, 0, 0 )
                    !
                Else
                    Call   SalCreateWindow(   frmMdiPrincipal.frmSLLFilleHw,
frmMdiPrincipal )
                !
```

e) Code derrière le bouton de commande **Rerchecher** de l'écran Sélection.

```
Pushbutton: pbChercherSelectionHw
    Class Child Ref Key: 0
    Class ChildKey: 0
    Class: clpbDialogBoxChercher
    Property Template:
    Class DLL Name:
    Title:
    Window Location and Size
        Left:  7.683"
        Top:   0.571"
        Width:  2.3"
        Width Editable? Class Default
        Height: Class Default
        Height Editable? Class Default
    Visible? Class Default
    Keyboard Accelerator: Class Default
    Font Name: Class Default
    Font Size: Class Default
    Font Enhancement: Class Default
    Picture File Name:
    Picture Transparent Color: Class Default
    Image Style: Class Default
    Text Color: Class Default
    Background Color: Class Default
    Message Actions
        On SAM_Click
            !
            Set bmTableUtilis=FALSE
            Set bmTableTrouve=FALSE
            Set frmMdiPrincipal.dfIdentifiantEnCours=''
            !
            If cbNonDisponible
                !
```

```
                    Set smSelectLoc=''||SLLDB||'Hardware.Dispo=\'1\''
                    !
                Else
                    !
                    Set smSelectLoc=''||SLLDB||'Hardware.Dispo=\'0\''
                    !
                !
                If SalStrLength(SalStrTrimX(dfCodeHw))>0
                    Set smCODEHW=SalStrUpperX( SalStrTrimX(dfCodeHw ))
                    Set smSelectLoc=smSelectLoc||' and
'||SLLDB||'Hardware.CODEHW'||' LIKE \''||smCODEHW||'%\' '
                !
                If SalStrLength( SalStrTrimX(cmbLoginUser ))>0
                    Set smLOGIN=SalStrUpperX( SalStrTrimX(cmbLoginUser
))
                    Set smSelectLoc=smSelectLoc||' and
'||SLLDB||'utilis.login'||' LIKE \''||smLOGIN||'%\' '
                    Set bmTableUtilis=TRUE
                !
                If SalStrLength(SalStrTrimX(cmbLieu))>0
                    Set smCODEMPL=SalStrUpperX( SalStrTrimX(cmbLieu ))
                    Set smSelectLoc=smSelectLoc||' and
'||SLLDB||'trouve.CODEMPL'||' LIKE \''||smCODEMPL||'%\' '
                    Set bmTableTrouve=TRUE
                !
                If SalStrLength(SalStrTrimX(cmbGroupeHw))>0
                    Set smGROUPEHW=SalStrUpperX(SalStrTrimX(
cmbGroupeHw ))
                    Set smSelectLoc=smSelectLoc||' and GROUPEHW'||' LIKE
\''||smGROUPEHW||'%\' '
                !
                !
                Set nmRequete=4
                If bmTableUtilis and bmTableTrouve
                    !
                    Set nmRequete=1
                    !
                    Call SqlExists( 'select 1 from
'||SLLDB||'Hardware,'||SLLDB||'utilis,'||SLLDB||'trouve where
                '||SLLDB||'Hardware.codehw='||SLLDB||'utilis.codehw and
                    '||SLLDB||'Hardware.codehw='||SLLDB||'trouve.codehw
and '||smSelectLoc ,bExists )
                    !
                    ! If not bExists
                        Set nmRequete=4
```

```
                    !
                Else
                    !
                    If bmTableUtilis
                        !
                        Set nmRequete=2
                        !
                        !
                        Call SqlExists( 'select 1 from
'||SLLDB||'Hardware,'||SLLDB||'utilis where
    '||SLLDB||'Hardware.codehw='||SLLDB||'utilis.codehw and
                                '||smSelectLoc ,bExists )
                        !
                        ! If not bExists
                            Set nmRequete=4
                        !
                    Else
                        If bmTableTrouve
                            !
                            Set nmRequete=3
                            !
                            Call SqlExists( 'select 1 from
'||SLLDB||'Hardware,'||SLLDB||'trouve where
'||SLLDB||'Hardware.codehw='||SLLDB||'trouve.codehw and '||smSelectLoc ,bExists )
                            !
                            ! If not bExists
                                Set nmRequete=4
                !
                If nmRequete=1
                    !
                    Set smSQLSelect='select
'||SLLDB||'Hardware.CODEHW,'||SLLDB||'Hardware.groupehw,
    '||SLLDB||'Hardware.typehw,'||SLLDB||'Hardware.nserie,'||SLLDB||'Hardware
.dispo,'||SLLDB||'trouve.codempl,'||SLLDB||'utilis.login
            FROM '||SLLDB||'Hardware,'||SLLDB||'utilis,'||SLLDB||'trouve into
:smCODEHW,:smGROUPEHW,:smTYPEHW,:smNSERIE,:bmDISPO,:smCODEMPL,:smLO
GIN where '||SLLDB||'Hardware.codehw='||SLLDB||'utilis.codehw and
'||SLLDB||'Hardware.codehw='||SLLDB||'trouve.codehw and '||SLLDB||'utilis.actif=1
and '||SLLDB||'Trouve.actif=1 and '||smSelectLoc
                    !
                Else
                    If nmRequete=2
                        !
                        Set smCODEMPL=''
```

```
                    Set                    smSQLSelect='select
'||SLLDB||'Hardware.CODEHW,'||SLLDB||'Hardware.groupehw,'||SLLDB||'Hardware.
typehw,'||SLLDB||'Hardware.nserie,'||SLLDB||'Hardware.dispo,'||SLLDB||'utilis.login
FROM '||SLLDB||'Hardware,'||SLLDB||'utilis into
:smCODEHW,:smGROUPEHW,:smTYPEHW,:smNSERIE,:bmDISPO,:smLOGIN where
'||SLLDB||'Hardware.codehw='||SLLDB||'utilis.codehw and '||SLLDB||'utilis.actif=1
and '||smSelectLoc
                    !
                Else
                    If nmRequete=3
                        !
                        Set smLOGIN=''
                        Set                    smSQLSelect='select
'||SLLDB||'Hardware.CODEHW,'||SLLDB||'Hardware.groupehw,
    '||SLLDB||'Hardware.typehw,'||SLLDB||'Hardware.nserie,'||SLLDB||'Hardware
.dispo,'||SLLDB||'trouve.codempl FROM '||SLLDB||'Hardware,'||SLLDB||'trouve into
:smCODEHW,:smGROUPEHW,:smTYPEHW,:smNSERIE,:bmDISPO,:smCODEMPL where
'||SLLDB||'Hardware.codehw='||SLLDB||'trouve.codehw and '||SLLDB||'trouve.actif=1
and '||smSelectLoc
                        !
                    Else
                        ! nmRequete=4
                        Set smLOGIN=''
                        Set smCODEMPL=''
                        Set                    smSQLSelect='select
'||SLLDB||'Hardware.CODEHW,'||SLLDB||'Hardware.groupehw,'||SLLDB||'Hardware.
typehw,'||SLLDB||'Hardware.nserie,'||SLLDB||'Hardware.dispo FROM
'||SLLDB||'Hardware into
:smCODEHW,:smGROUPEHW,:smTYPEHW,:smNSERIE,:bmDISPO where
'||smSelectLoc
                        !
                        !
            Call SalSendMsg( tblSelectionHardware,
PAM_RemplirTblSelection, 0, 0 )
            !
    !
```

f) Code derrière le bouton de commande **Ajouter** de l'écran Sélection.

```
Pushbutton: pbAjouterSelectionFourn
    Class Child Ref Key: 0
    Class ChildKey: 0
    Class: clpbDialogBoxAjouter
    Property Template:
```

```
Class DLL Name:
Title:
Window Location and Size
        Left:  8.083"
        Top:  0.988"
        Width:  1.5"
        Width Editable? Class Default
        Height: Class Default
        Height Editable? Class Default
Visible? Class Default
Keyboard Accelerator: Class Default
Font Name: Class Default
Font Size: Class Default
Font Enhancement: Class Default
Picture File Name:
Picture Transparent Color: Class Default
Image Style: Class Default
Text Color: Class Default
Background Color: Class Default
Message Actions
        !
        On PAM_Insert
                !
                Set smNOMFOUR=''
                !
                !
                If bfrmSLLFilleFournisseur
                        !
                        Call                                    SalBringWindowToTop(
frmMdiPrincipal.frmSLLFilleFournisseur )
                        Call  SalSendMsg(  frmMdiPrincipal.frmSLLFilleFournisseur,
SAM_Create, 0, 0 )
                        !
                Else
                        Call                                    SalCreateWindow(
frmMdiPrincipal.frmSLLFilleFournisseur, frmMdiPrincipal )
                !
        !
```

g) Code derrière le bouton de commande Editer de l'écran-Fille.

```
Pushbutton: pbEditerUser
        Class Child Ref Key: 0
        Class ChildKey: 0
        Class: clpbEditer
```

```
Property Template:
Class DLL Name:
Title:
Window Location and Size
        Left:  10.083"
        Top:  0.155"
        Width:  1.7"
        Width Editable? Class Default
        Height: Class Default
        Height Editable? Class Default
Visible? Class Default
Keyboard Accelerator: Class Default
Font Name: Class Default
Font Size: Class Default
Font Enhancement: Class Default
Picture File Name:
Picture Transparent Color: Class Default
Image Style: Class Default
Text Color: Class Default
Background Color: Class Default
Message Actions
        !
        On PAM_Confirmer
                !
                Set smLOGIN=SalStrUpperX(SalStrTrimX(dfLoginUser))
                Set smNOMUSER=SalStrUpperX(SalStrTrimX(dfNomUser))
                Set smPRENOM=SalStrUpperX(SalStrTrimX(dfPrenomUser))
                Set smDATENG=SLLDateVersAS400( dfDatEngagUser )
                Set smSERVICE=SalStrUpperX(SalStrTrimX(cmbServiceUser))
                Set smTEL=SalStrUpperX(SalStrTrimX(dfTelUser))
                Set smEMAIL=SalStrUpperX(SalStrTrimX(dfMailUser))
                If cbActif
                        Set bmACTIF=TRUE
                Else
                        Set bmACTIF=FALSE
                If cbGroupe
                        Set bmGROUPE=TRUE
                Else
                        Set bmGROUPE=FALSE
                Set lmREMARQUE=SalStrUpperX(SalStrTrimX(mlRemarkUser))
                Set smCODEMPL=SalStrUpperX(SalStrTrimX(cmbLieu))
                !
        !
        On PAM_ModifierInsert
                !
```

```
                Call IUtilisateur( )
                Call ASLLInsererTous( )
                !
                If SalStrLength( smCODEMPL )>0
                        !
                        Call SqlExists( 'select 1 from '||SLLDB||'travail where
login=:smLOGIN and codempl=:smCODEMPL and actif=1 ', bExists )
                        If not bExists
                                !
                                Call SqlImmediate( 'update '||SLLDB||'Travail set
actif=0 where login=:smLOGIN ' )
                                Call ITravail( )
                                Set bmACTIF=1
                                Call SqlImmediate( 'insert into '||SLLDB||'travail
('||sdePassage||',TLUU,TLUT) values ('||sversPassage||',user,current timestamp ) ' )
                                !
                        !
                        Call SqlCommit( hGeneral )
                        !
                        Call IUtilisateur( )
                        !
                !
        !
        On PAM_ModifierTransf
                !
                Call IUtilisateur( )
                Call ASLLUpdateTous( )
                !
                If SalStrLength( smCODEMPL )>0
                        !
                        Call SqlExists( 'select 1 from '||SLLDB||'travail where
login=:smLOGIN and codempl=:smCODEMPL and actif=1 ', bExists )
                        If not bExists
                                !
                                Call SqlImmediate( 'update '||SLLDB||'Travail set
actif=0 where login=:smLOGIN ' )
                                Call ITravail( )
                                Set bmACTIF=1
                                Call SqlImmediate( 'insert into '||SLLDB||'travail
('||sdePassage||',TLUU,TLUT) values ('||sversPassage||',user,current timestamp ) ' )
                                !
                        !
                        Call SqlCommit( hGeneral )
                        !
                        Call IUtilisateur( )
```

```
                    !
                !
```

h) Code derrière le bouton de commande **Historique** de l'écran-Fille.

```
Pushbutton: pbHistoHw
        Class Child Ref Key: 0
        Class ChildKey: 0
        Class: clpbAntiNormal
        Property Template:
        Class DLL Name:
        Title: Historique
        Window Location and Size
                Left:  9.283"
                Top:  1.321"
                Width:  2.1"
                Width Editable? Class Default
                Height: Class Default
                Height Editable? Class Default
        Visible? Class Default
        Keyboard Accelerator: Class Default
        Font Name: Class Default
        Font Size: Class Default
        Font Enhancement: Class Default
        Picture File Name:
        Picture Transparent Color: Class Default
        Image Style: Class Default
        Text Color: Class Default
        Background Color: Class Default
        Message Actions
                On SAM_Click
                        Call SalModalDialog( dlgHistoriqueTousHw, hWndForm )
                        !
```

i) Code de contrôle sur un champ d'édition.

```
Data Field: dfNumeroLeasing
        Class Child Ref Key: 0
        Class ChildKey: 0
        Class: cldfStrSimple
        Property Template:
        Class DLL Name:
        Data
                Maximum Data Length: 15
                Data Type: Class Default
```

```
            Editable? Class Default
        Display Settings
            Window Location and Size
                Left:  2.283"
                Top:  0.488"
                Width:  2.0"
                Width Editable? Class Default
                Height: 0.25"
                Height Editable? Class Default
            Visible? Class Default
            Border? Class Default
            Justify: Class Default
            Format: Class Default
            Country: Class Default
            Font Name: Class Default
            Font Size: Class Default
            Font Enhancement: Class Default
            Text Color: Class Default
            Background Color: Class Default
            Input Mask: Class Default
        Message Actions
            On SAM_Create
                If SalStrLength(smNLEASING)>0
                    Set dfNumeroLeasing=smNLEASING
                    Call SalDisableWindow( dfNumeroLeasing )
                    Call SalSetFocus( dfGarantieLeasing )
                    Set dfNLInvisible=smNLEASING
                Else
                    Set dfNumeroLeasing=''
                    Call SalSetFocus( dfNumeroLeasing )
                    Set dfNLInvisible=''
            On SAM_Validate
                If  SalStrLength(  SalStrTrimX(  dfNumeroLeasing))=0  or
SalIsNull(dfNumeroLeasing)
                    Set bValid=FALSE
                    Call SalMessageBox( 'Le numéro du leasing est obligatoire.',
'ERREUR', MB_Ok )
                    Call SalSetFocus( dfNumeroLeasing )
```

j) <u>Code de contrôle sur une liste déroulante</u>.

```
Combo Box: cmbNomFournis
        Class Child Ref Key: 0
        Class ChildKey: 0
```

```
Class: clcbStrNormal
Property Template:
Class DLL Name:
Window Location and Size
        Left:  5.183"
        Top:  1.488"
        Width:  1.9"
        Width Editable? Class Default
        Height: 2.774"
        Height Editable? Class Default
Visible? Class Default
Editable? Class Default
String Type: Class Default
Maximum Data Length: 25
Sorted? Class Default
Always Show List? Class Default
Vertical Scroll? Class Default
Font Name: Class Default
Font Size: Class Default
Font Enhancement: Class Default
Text Color: Class Default
Background Color: Class Default
Input Mask: Class Default
List Initialization
Message Actions
        On SAM_DropDown
                !
                Call SalListClear( cmbNomFournis )
                !
                If SqlPrepareAndExecute( hGeneral, 'select distinct nomfour from
'||SLLDB||'Fournisseur  into  :smNOMFOUR order by nomfour  ' )
                        !
                        Call SqlFetchNext( hGeneral, nmDernierLu )
                        While nmDernierLu!=FETCH_EOF
                                !
                                Call SalListAdd( hWndItem, smNOMFOUR )
                                !
                                Call SqlFetchNext( hGeneral, nmDernierLu )
                                !
                        !
                        Call SqlCommit( hGeneral )
                        !
                !
        On SAM_Validate
                !
```

```
                Set smNOMFOUR=SalStrTrimX( SalStrUpperX( cmbNomFournis )
)
                !
                If SalStrLength(smNOMFOUR )>0
                        !
                        Call SqlExists('select 1 from '||SLLDB||'Fournisseur where
'||SLLDB||'Fournisseur.NOMFOUR=:smNOMFOUR' , bExists )
                        If not bExists
                                !
                                If SalMessageBox( 'Ce fournisseur saisi n\'existe
pas.Voulez-vous qu\'il soit ajouté ?', 'Repondez', MB_YesNo )=6
                                        !
                                        Call IFournisseur( )
                                        Call SqlImmediate( 'insert into
'||SLLDB||'Fournisseur (NOMFOUR) values (:smNOMFOUR) ' )
                                        !
                                        Call IHardware( )
                                        !
                                Else
                                        !
                                        Set cmbNomFournis=''
                                        Set smNOMFOUR=cmbNomFournis
                                        Set bValid=FALSE
                                        !
                                !
                        !
                        Call SqlCommit( hGeneral )
                        Call IHardware( )
                        !
                !
                Else
                        !
                        If SalStrLength(SalStrUpperX( cmbNomFournis))=0 or
SalIsNull (cmbNomFournis)
                                Set bValid=FALSE
                                Call SalMessageBox( 'Le fournisseur du hardware est
obligatoire', 'AVERTISSEMENT', MB_Ok )
                                Call SalSetFocus( cmbNomFournis )
                !
```

Les quelques fonctions suivantes ont été tirées de la librairie de développement de Swiss Life.

Function: SLLDateVersAS400

```
Description: Du format Date/Heure au format AS400 string
Returns
   String:
   Parameters
   Date/Time: dtDate Static Variables
   Local variables
   String:  sDateC  Date/Time:  tDateBlanc
   Actions
     !fdtDate=dtDateBlanc
     Set sDateC=" EIse
     Set sDateC=SalFmtFormatDateTime(dtDate,
     SLLDateFmtAS400 )
      Return sDateC
```

Function: SLLAS400VersDate

```
Description: Du format AS400 string vers Date/Heure.
Returns
 Date/Time: Parameters
  String: sDateC
Static Variables
Local    variables    Date/Time:    dtDate
Date/Time: dtLocBlanc Actions
If sDateC=" or sDateC="
Set dtDate=dtLocBlanc Else
    Set dtDate=SalStrToDate( sDateC )
    Set dtDate=Sa!DateConstruct( SalStrToNumber( SalStrLeftX( sDateC, 4 ) ),
    SalStrToNumber( SalStrRightX( SalStrLeftX( sDateC, 7 ), 2 ) ), SalStrToNumber(
    SalStrRightX( sDateC, 2 ) ), 0, 0, 0 )
    Call SalFmtFormatDateTime( dtDate, SLLDateFmtClient )
Return dtDate
```

Function: ASLLOuvrirFenetreFille

Description : Ouvre une fenêtre fille en fonction du nom de la fenêtre. Il met un booléen à vrai quand la fenêtre est ouverte.

Si la fenêtre est déjà ouverte alors il l'agrandit et envoie un message de SAM_Create.

```
Returns
Parameters
Static Variables
Local variables
Actions
!
    If hEcranOuvert = hWndNULL
    Set hEcranOuvert=SalCreateWindow( frmFenetreAOuvnr,
      hFenetrePrincipale )
    Else
    Call SalBringWindowToTop( hEcranOuvert )
    Call SalSendMsg( hEcranOuvert, PAM^Enable, 0. 0 )
   Call SalSendMsg( hEcranOuvert, SAM_Create, 0. 0 )
!
```

Function: SLLConnectAll

Description: Procédure qui me connecte à ma base de données
Returns
Parameters
Static Variables
Local variables
Actions

```
If SqlConnect (hGeneral)
   IfSqlConnect(hBlock)
      Call SqlSetIsolationLevel( hBlock, 'RL' )
      Call SqlSetIsolationLevel( hGeneral, 'RL' )
      Return TRUE
Return FALSE
```

Function: SLLDisConnectAll

Description: Procédure qui ferme toutes les connexions à la base de données.
Returns
Parameters
Static Variables
Local variables
Actions

```
Call SqlDisconnect( hGeneral )
Call SqlDisconnect( hBlock )
```

Function: ASLLUpdateTous

Description: Mise à jour des fichiers dernièrement définis par IFile en fonction des données contenues en mémoire.
Returns
Parameters
Static Variables
Local variables
Actions

```
If SalStrLength( sIdentPassage2 )=0

  Return SqlImmediate( 'Update '||SLLDB||sNomFilePassage||'

 set'||sUpdPassage||',

'||sPR_KeyPassage||'UU=user,'||sPR_KeyPassage||'UT=
currenttimestamp
where '||sIdentPassage||'=:'||sIdentPassInto )
!
Else

  Return SqlImmediate( 'Update '||SLLDB||sNomFilePassage||'

  Set
   '|]sUpdPassage||','||sPR_KeyPassage||'UU=user.'||sPR_KeyPassage||'
UT=c urrent timestamp
where          '||sIdentPassage||'=:'||sIdentPassInto||'          and
'||sIdentPassage2||'=:'||sIdentPassInto2)
```

Function: ASLLInsererTous

Description: Insère un enregistrement avec toutes les données en relation avec le dernier IFile exécuté.

Returns

Parameters

Static Variables

Local variables

Actions

```
!
   If SalStrLength( SalStrTrimX( sIdentPassage2 ) )>0
      Return SqlImmediate( ' insert into '||SLLDB||sNomFilePassage||'
   ('||sIdentPassage||','||sIdentPassage2||','||sdePassage||','||sPR_KeyPassage||
   'UU.'UsPR_KeyPassageII'UT)
   values
(:'||sIdentPassInto||',:'||sIdentPassInto2||','||sversPassage||',user,current
timestamp)' )
   Else
   Return SqlImmediate( ' insert into '||SLLDB||sNomFilePassage||'
   ('||sIdentPassage||','||sdePassage||','||sPR_KeyPassage||'UU,'||sPR_KeyPassage||
   'UT)
   values (:'||sIdentPassInto||\'||sversPassage|]',user,current timestamp)' )
Return False
```

Annexe 6 :

Mode d'emploi de SLLParc.

MODE D'EMPLOI DU GESTIONNAIRE DU PARC INFORMATIQUE SLLPARC

Introduction

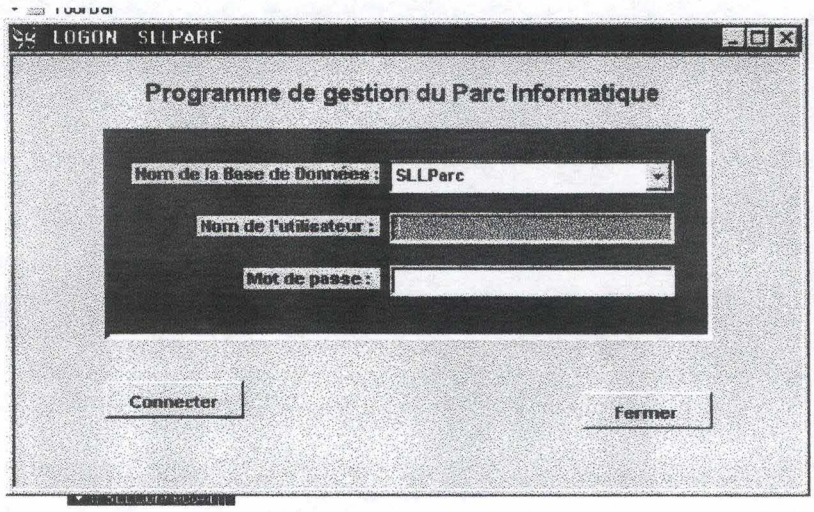
- A travers ce document, nous avons pour objectif de montrer au lecteur le fonctionnement de cette application. Nous le faisons avec des écrans de l'application afin d'être plus prêt de la réalité en expliquant le rôle de chaque bouton de commande.

- Nous imaginons un utilisateur fidèle. Dans l'exécution réelle, il y a des messages de contrôle qui peuvent apparaître en cas d'une erreur. Ces messages ne sont pas visibles ici.

- Certes, nous ne répondrons pas à toutes les attentes du lecteur mais nous l'invitons à exécuter SLLParc.

Ecran de Logon de l'application SLLParc

En exécutant SLLParc, nous arrivons à l'écran d'accès suivant. Le nom de la base de données y figure par défaut. L'utilisateur saisit son nom puis son mot de passe pour lui permettre de se connecter à la base de données.



Le bouton de commande **Connecter** permet de se relier à la base de données SLLParc. Par contre, le bouton de commande **Fermer** vous fait quitter l'application.

Ecran Principal de l'application SLLParc

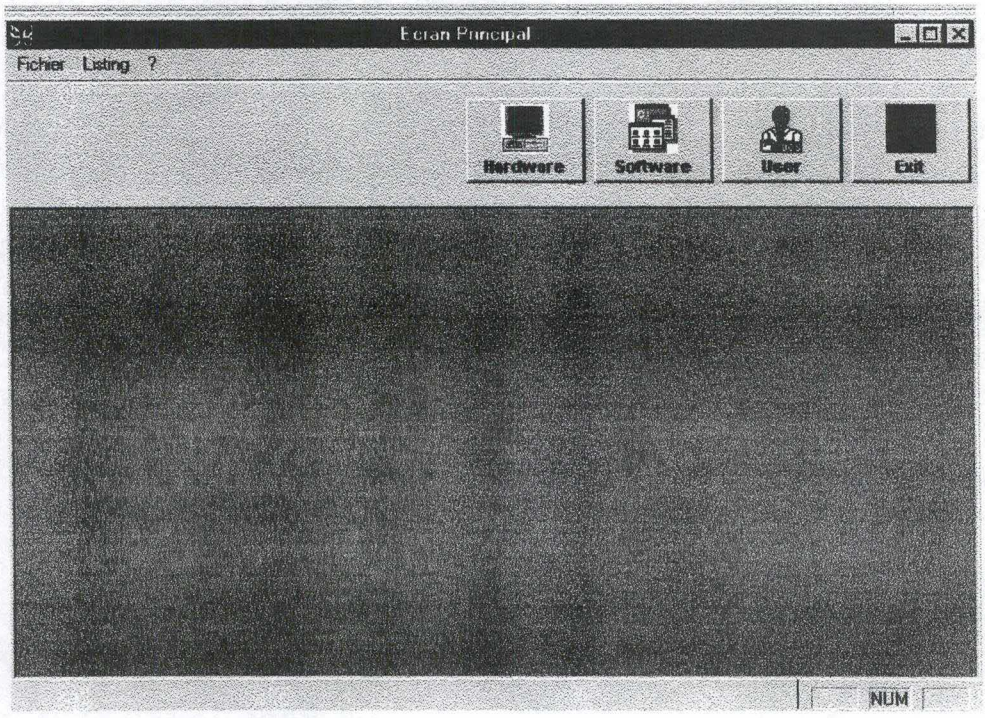
Si vos données précédentes sont exactes et que la connexion à la base de données SLLParc s'est passée normalement, l'application vous ouvre l'écran suivant qui est l'écran principal de l'application.

Il contient : * sur la barre de menu => le menu Fichier par lequel l'on peut accéder au hardware, au software, à l'utilisateur, au consommable, au fournisseur, à l'ajout d'une formation et à la sortie de l'application.

⇨ le menu Listing qui propose l'exécution de deux listing de l'application.

⇨ Le point d'interrogation (?) donne quelques informations sur SLLParc.

* sur la barre d'outil les quatre boutons de commande qui permettent d'accéder directement aux fonctionnalités régulières de SLLParc.



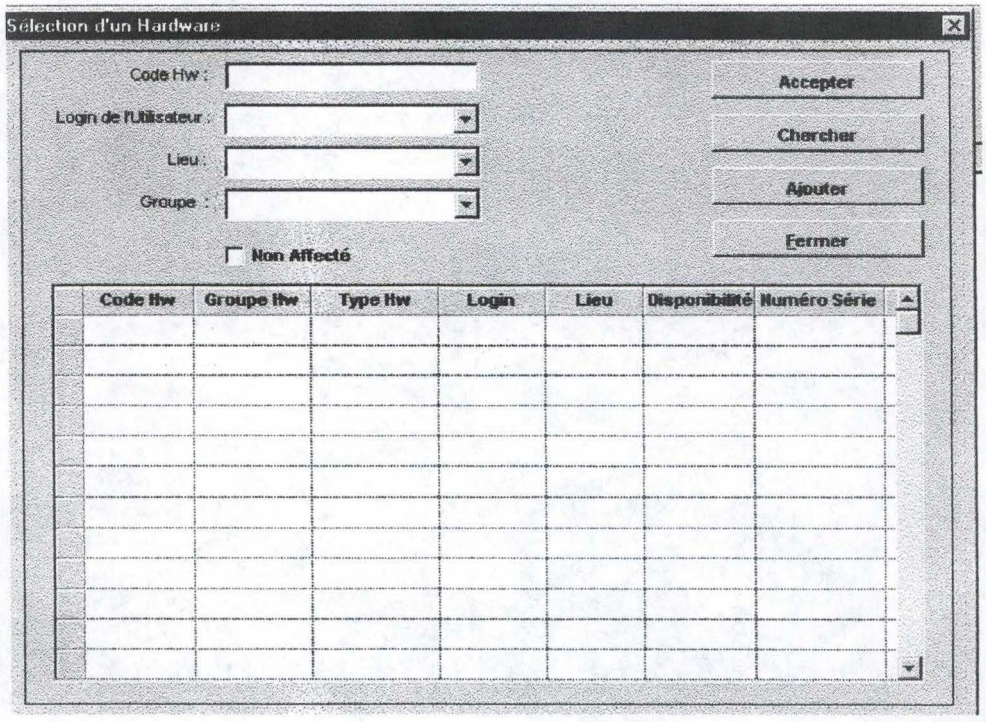
- Un clic sur le bouton de commande **Hardware** revient au même à un clic sur Hardware du menu Fichier déroulé. Et cette action vous ouvre l'écran de sélection d'un hardware.
- Un clic sur le bouton de commande **Software** revient au même à un clic sur Software du menu Fichier déroulé. Et cette action vous ouvre l'écran de sélection d'un Software.
- Un clic sur le bouton de commande **User** revient au même à un clic sur User du menu Fichier déroulé. Et cette action vous ouvre l'écran de sélection d'un Utilisateur.
- Un clic sur Consommable du menu Fichier déroulé ouvre l'écran de sélection d'un consommable.

- Un clic sur Fournisseur du menu Fichier déroulé ouvre l'écran de sélection d'un fournisseur.
- Un clic sur Formation du menu Fichier déroulé ouvre l'écran d'ajout d'un formation.
- Un clic sur le bouton de commande **Exit** revient au même à un clic sur Exit du menu Fichier déroulé. Et cette action vous fait sortir de l'application.

HARDWARE

Ecran de sélection d'un hardware

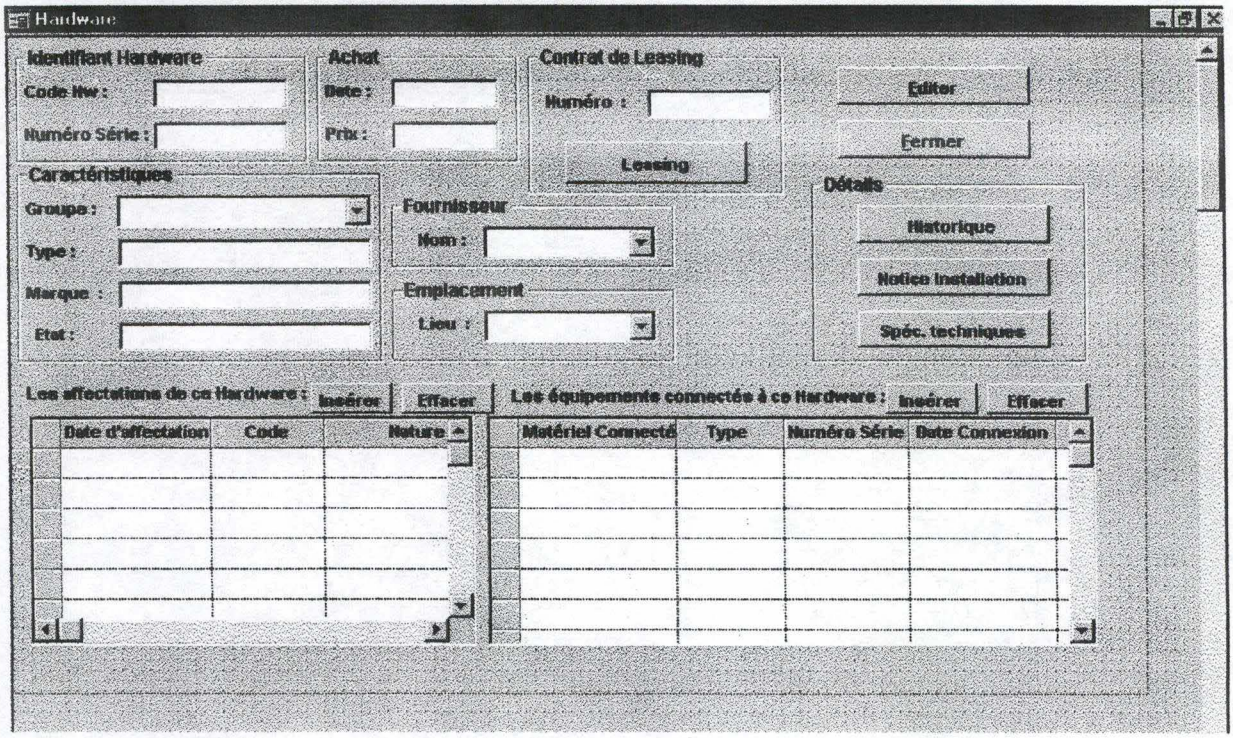
Cet écran permet de sélectionner un seul hardware parmi les hardwares affichés dans le tableau et qui répondent aux critères de sélection saisis et après un clic sur **Chercher**.



- Un clic sur le bouton de commande **Accepter** n'est valide qu'après avoir sélectionné une ligne concernée dans le tableau par simple clic. Et cette action vous ouvre l'écran des données du hardware sélectionné.
- En constatant que votre matériel cherché ne s'y trouve pas ou que vous voulez ajouter un nouveau hardware, ouvrez l'écran des données du hardware vide pour l'ajouter par un clic sur le bouton de commande **Ajouter**.
- Un clic sur le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.

Ecran des données du hardware sélectionné

Si cet écran est ouvert en mode *AJOUT*, vous n'avez qu'à saisir les données et appuyer sur le bouton de commande **Confirmer** pour valider les saisies.
Si par contre, il est ouvert avec des données du hardware sélectionné, vous pouvez cliquer sur le bouton de commande **Editer** pour modifier vos données sauf l'identifiant.
Ou procédez aux opérations désirées ci-dessous.
Le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.



Un clic sur le bouton de commande **Leasing** vous permet d'ouvrir l'écran de leasing du hardware sélectionné.

Un clic sur le bouton de commande **Historique** vous permet d'ouvrir l'écran de l'historique des affectations et connexions du hardware sélectionné.

Un clic sur le bouton de commande **Notice d'installation** vous permet d'ouvrir l'écran de la notice d'installation du hardware sélectionné.

Un clic sur le bouton de commande **Spéc. techniques** vous permet d'ouvrir l'écran des spécifications techniques du hardware sélectionné.

Un clic sur le bouton de commande **Insérer (à gauche)** vous permet d'ouvrir l'écran des affectations.

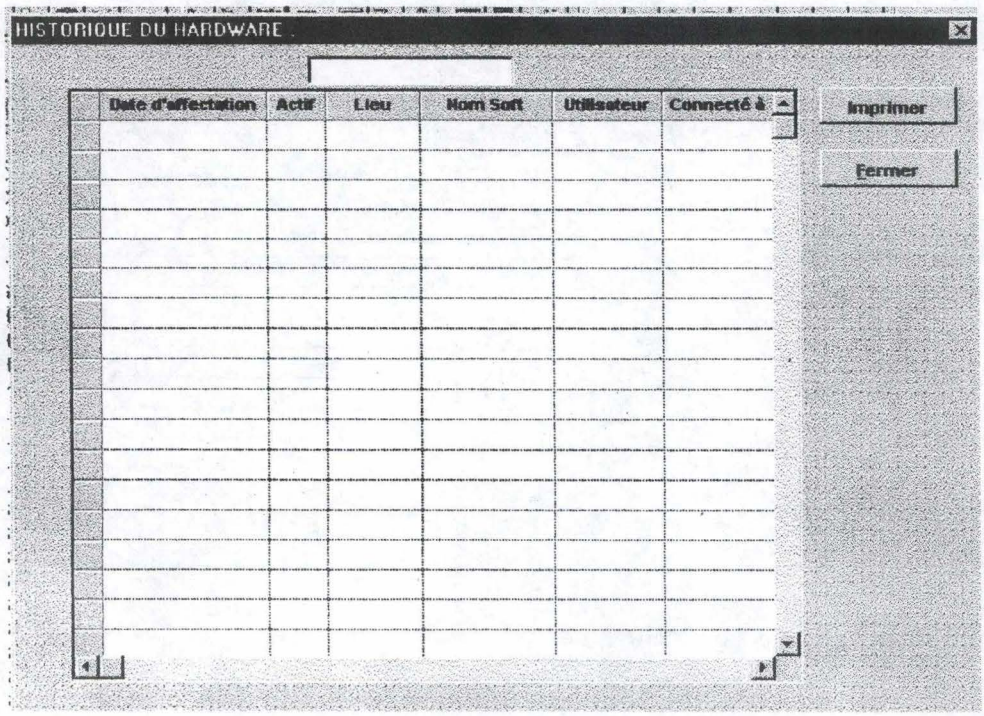
Un clic sur le bouton de commande **Insérer (à droite)** vous permet d'ouvrir l'écran des connexions des hardwares.

Un clic sur le bouton de commande **Effacer (à gauche)** vous permet d'annuler une affectation existante dans le tableau de gauche.

Un clic sur le bouton de commande **Effacer (à droite)** vous permet d'annuler une connexion existante dans le tableau de droite.

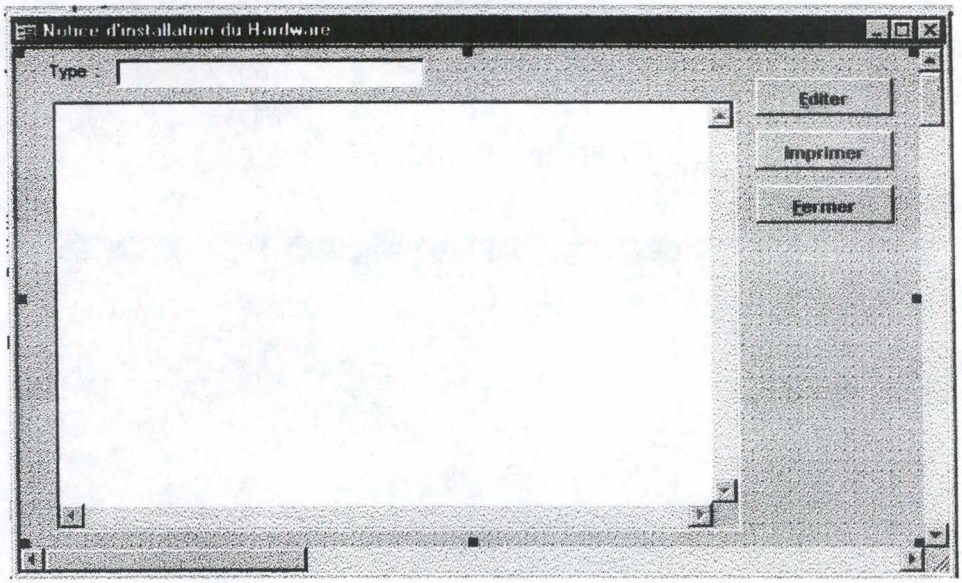
Un double-clic sur les tableaux vous permet d'obtenir des détails de l'élément concerné.

Ecran de l'historique des affectations et connexions du hardware sélectionné



Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cet historique.

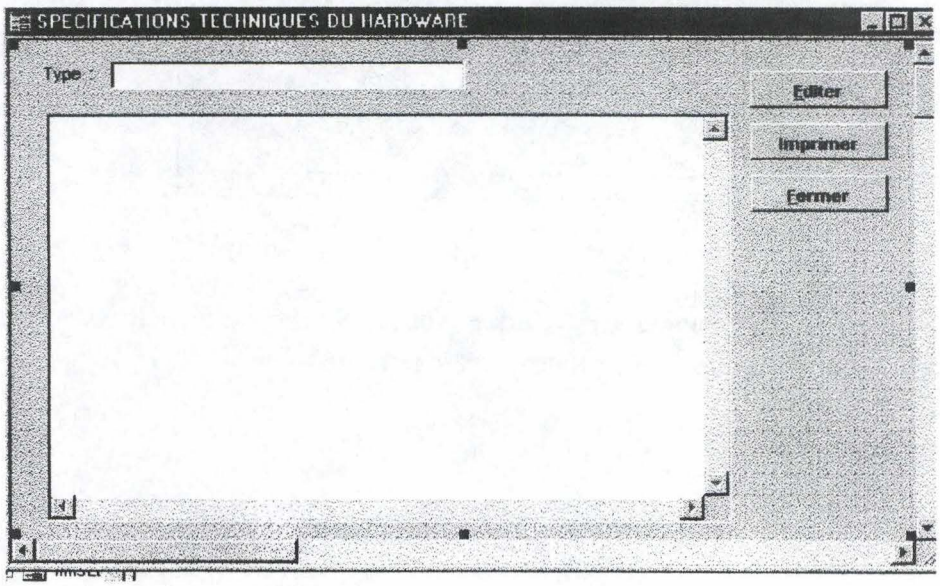
Ecran de la notice d'installation du hardware sélectionné



Un clic sur le bouton de commande **Editer** vous permet de saisir ou de modifier cette notice.

Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cette notice.

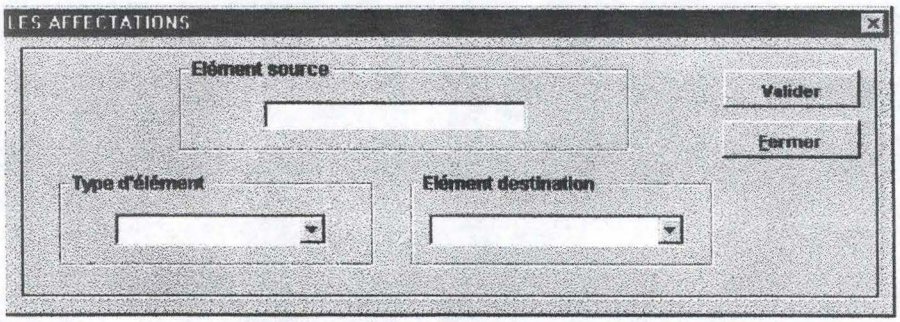
Ecran des spécifications techniques du hardware sélectionné



Un clic sur le bouton de commande **Editer** vous permet de saisir ou de modifier cette notice.

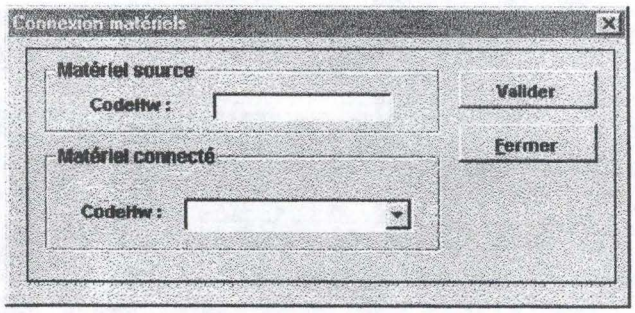
Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer ces données.

Ecran des affectations



Un clic sur le bouton de commande **Valider** vous permet de confirmer l'affectation réalisée.

Ecran des connexions des hardwares

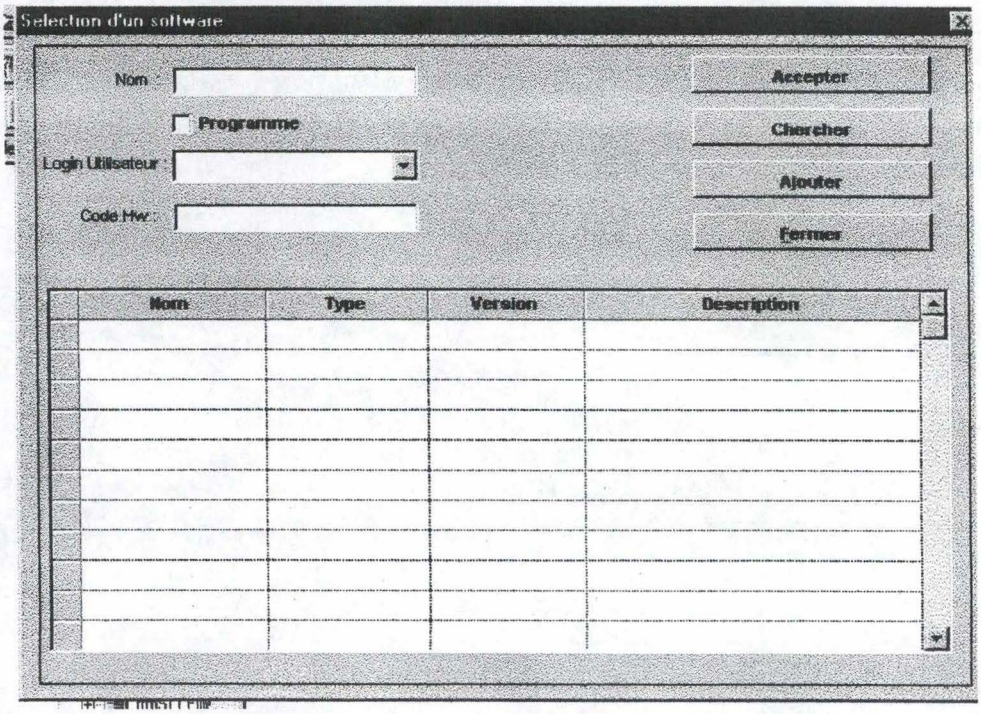


Un clic sur le bouton de commande **Valider** vous permet de confirmer la connexion réalisée. Le système empêche deux connexions identiques.

SOFTWARE

Ecran de sélection d'un software

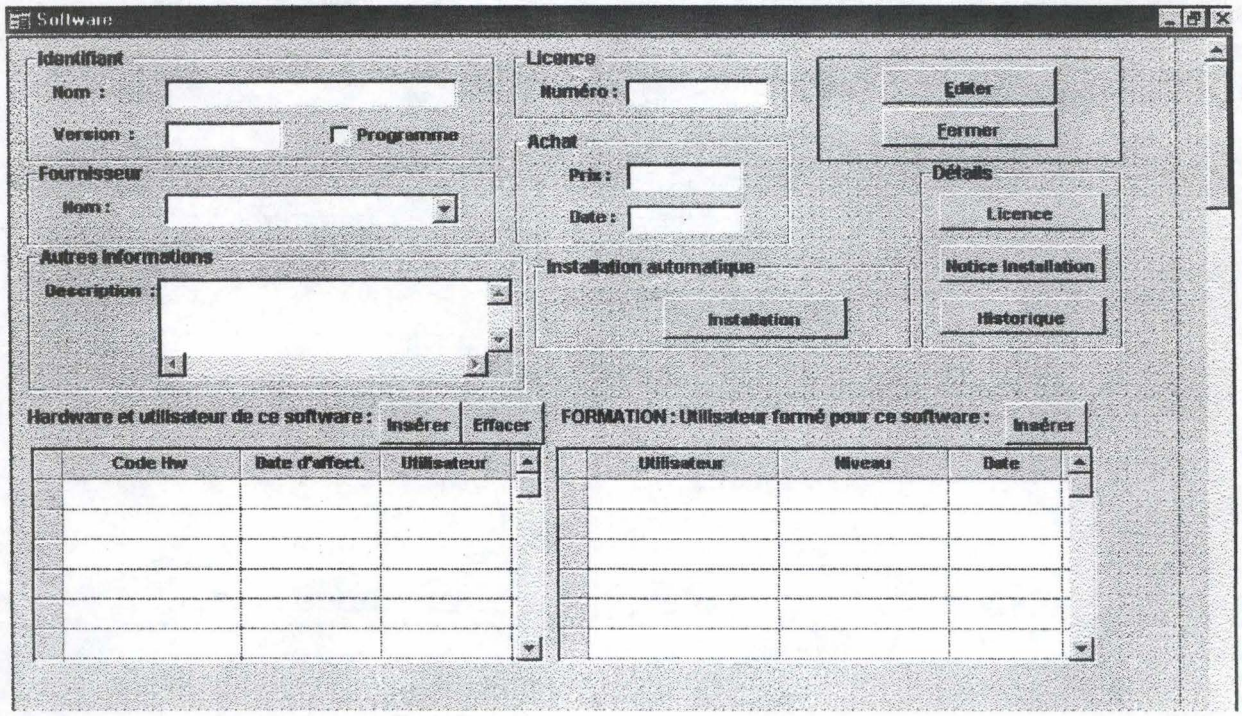
Cet écran permet de sélectionner un seul software parmi les softwares affichés dans le tableau et qui répondent aux critères de sélection saisis et après un clic sur **Chercher**.



- Un clic sur le bouton de commande **Accepter** n'est valide qu'après avoir sélectionné une ligne concernée dans le tableau par simple clic. Et cette action vous ouvre l'écran des données du software sélectionné.
- En constatant que votre logiciel cherché ne s'y trouve pas ou que vous voulez ajouter un nouveau software, ouvrez l'écran des données du software vide pour l'ajouter par un clic sur le bouton de commande **Ajouter**.
- Un clic sur le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.

Ecran des données du software sélectionné

Si cet écran est ouvert en mode *AJOUT*, vous n'avez qu'à saisir les données et appuyer sur le bouton de commande **Confirmer** pour valider les saisies.
Si par contre, il est ouvert avec des données du software sélectionné à l'écran précédent, vous pouvez cliquer sur le bouton de commande **Editer** pour modifier vos données sauf l'identifiant.
Ou procédez aux opérations désirées ci-dessous.
Le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.



Un clic sur le bouton de commande **Licence** vous permet d'ouvrir l'écran de licence du software sélectionné.

Un clic sur le bouton de commande **Notice d'installation** vous permet d'ouvrir l'écran de la notice d'installation du software sélectionné.

Un clic sur le bouton de commande **Historique** vous permet d'ouvrir l'écran de l'historique des affectations du software sélectionné.

Un clic sur le bouton de commande **Installation** vous permet d'ouvrir l'écran des installations automatique des softwares (non fonctionnel).

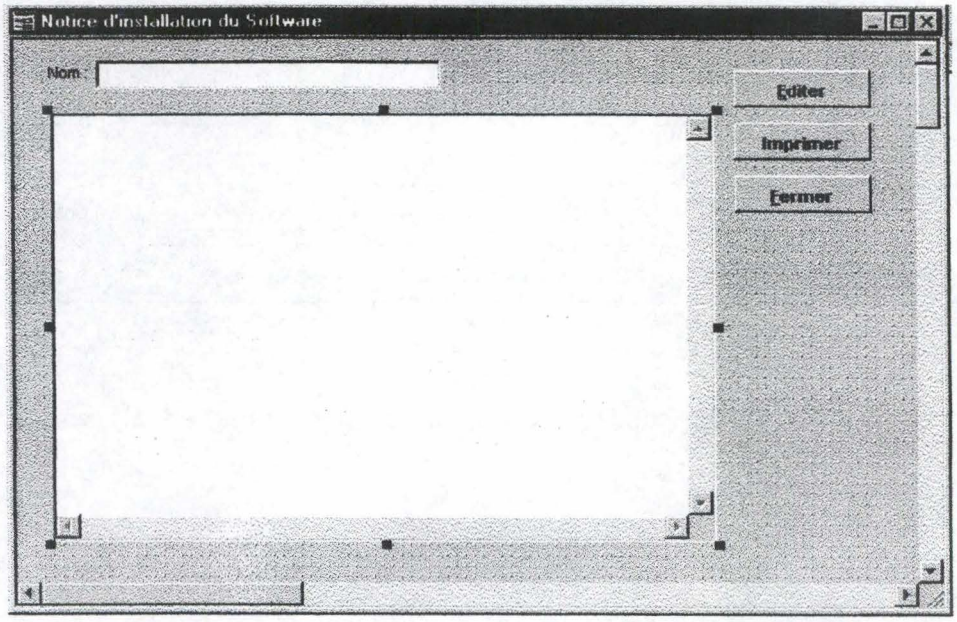
Un clic sur les boutons de commande **Insérer (à gauche)** vous permet d'ouvrir l'écran des affectations.

Un clic sur le bouton de commande **Effacer** vous permet d'annuler une affectation existante dans le tableau de gauche.

Un clic sur les boutons de commande **Insérer (à droite)** vous permet d'ouvrir l'écran des affectations de formation.

Un double-clic sur les tableaux vous permet d'obtenir des détails de l'élément concerné.

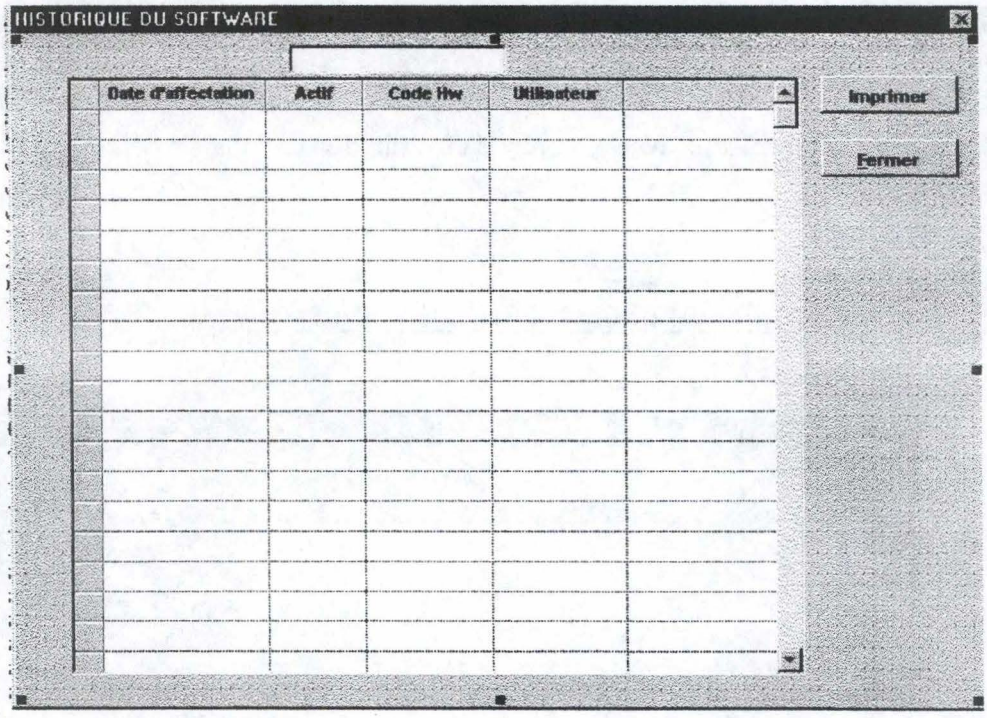
Ecran de notice d'installation d'un software



Un clic sur le bouton de commande **Editer** vous permet de saisir ou de modifier cette notice.

Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cette notice.

Ecran de l'historique des affectations du software sélectionné

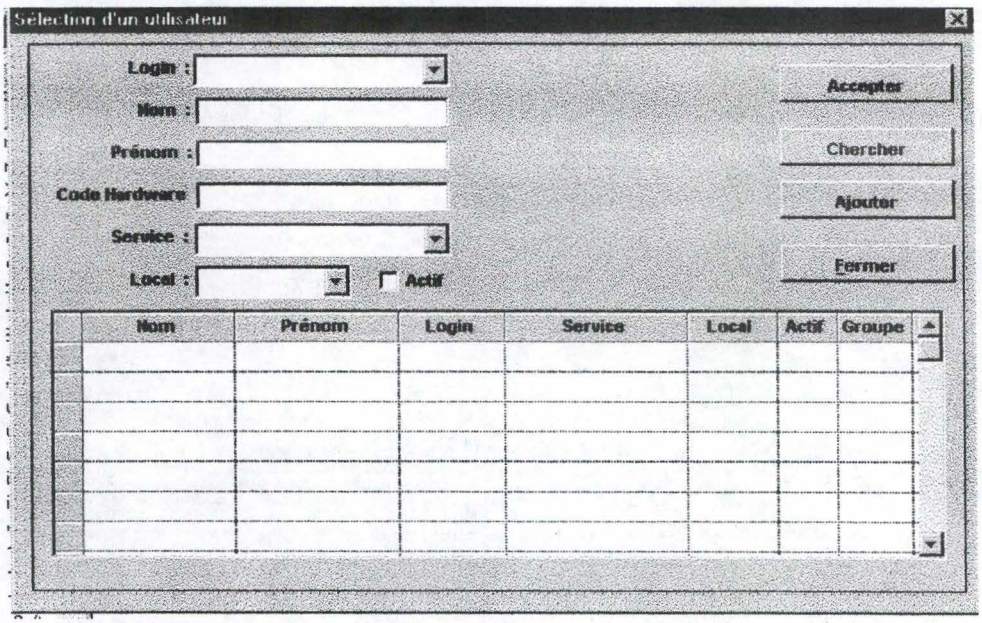


Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cet historique.

UTILISATEUR

Ecran de sélection d'un utilisateur

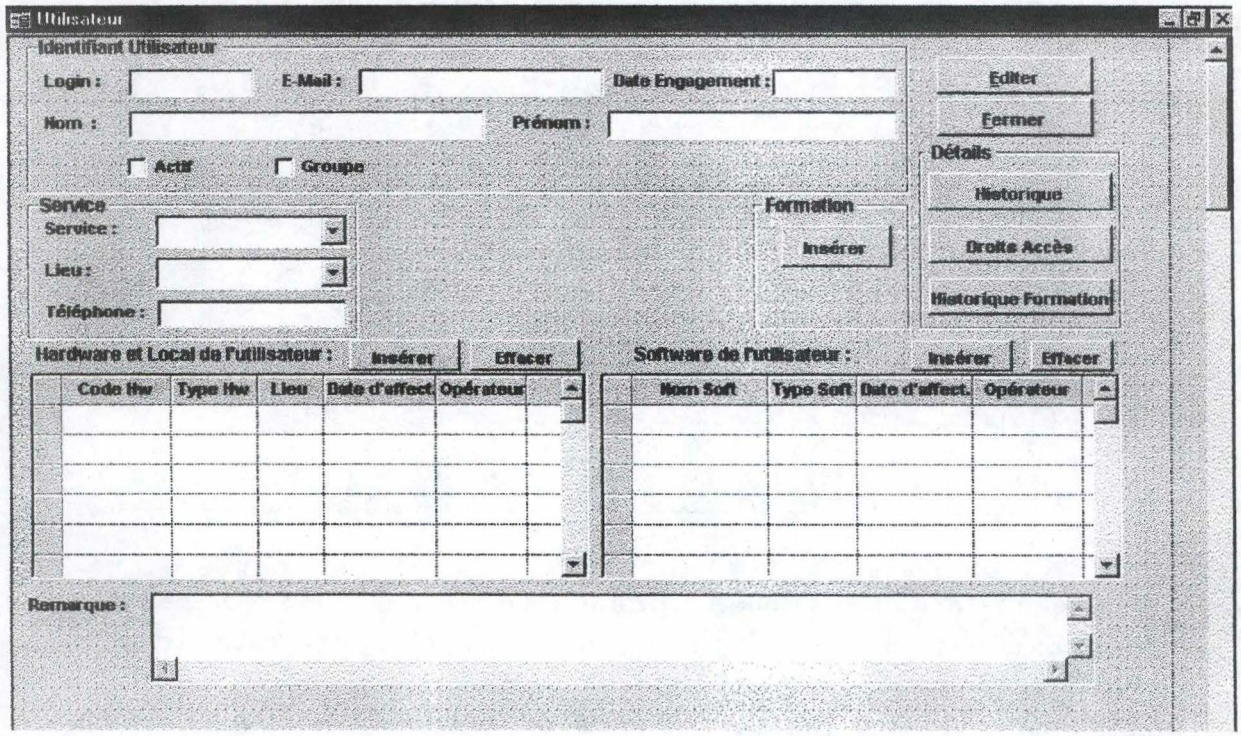
Cet écran permet de sélectionner un seul utilisateur parmi ceux affichés dans le tableau et qui répondent aux critères de sélection saisis et après un clic sur **Chercher**.



- Un clic sur le bouton de commande **Accepter** n'est valide qu'après avoir sélectionné une ligne dans le tableau par simple clic. Et cette action vous ouvre l'écran des données de l'utilisateur sélectionné.
- En constatant que votre utilisateur cherché ne s'y trouve pas ou que vous voulez ajouter un nouvel utilisateur, ouvrez l'écran des données de l'utilisateur vide pour l'ajouter par un clic sur le bouton de commande **Ajouter**.
- Un clic sur le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.

Ecran des données de l'utilisateur sélectionné

Si cet écran est ouvert en mode *AJOUT*, vous n'avez qu'à saisir les données et appuyer sur le bouton de commande **Confirmer** pour valider les saisies.
Si par contre, il est ouvert avec des données de l'utilisateur sélectionné à l'écran précédent, vous pouvez cliquer sur le bouton de commande **Editer** pour modifier vos données sauf l'identifiant.
Ou procédez aux opérations désirées ci-dessous.
Le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.



Un clic sur le bouton de commande **Historique** vous permet d'ouvrir l'écran de l'historique des affectations à l'utilisateur sélectionné.

Un clic sur le bouton de commande **Droits Accès** vous permet d'ouvrir l'écran des droits d'accès de l'utilisateur concerné (non fonctionnel).

Un clic sur le bouton de commande **Historique Formation** vous permet d'ouvrir l'écran de l'historique des formations informatiques suivies par l'utilisateur sélectionné.

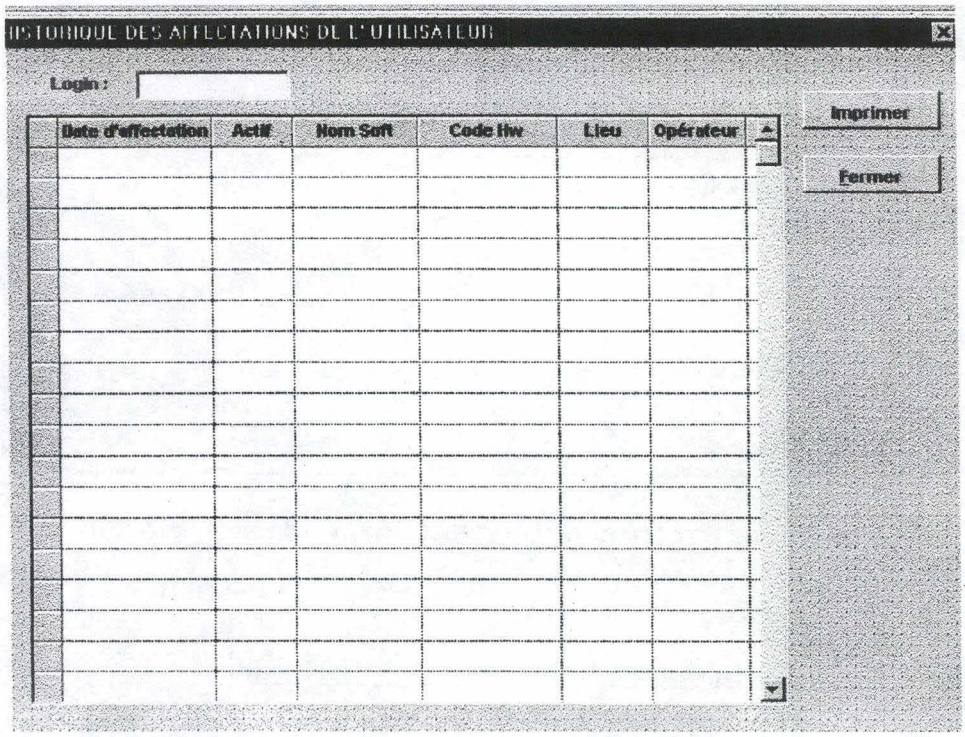
Un clic sur les boutons de commande **Insérer (Formation)** vous permet d'ouvrir l'écran d'ajout d'une formation informatiques.

Un clic sur les boutons de commande **Insérer (à droite ou à gauche)** vous permet d'ouvrir l'écran des affectations .

Un clic sur le bouton de commande **Effacer (à droite ou à gauche)** vous permet d'annuler une affectation existante dans le tableau de droite ou de gauche.

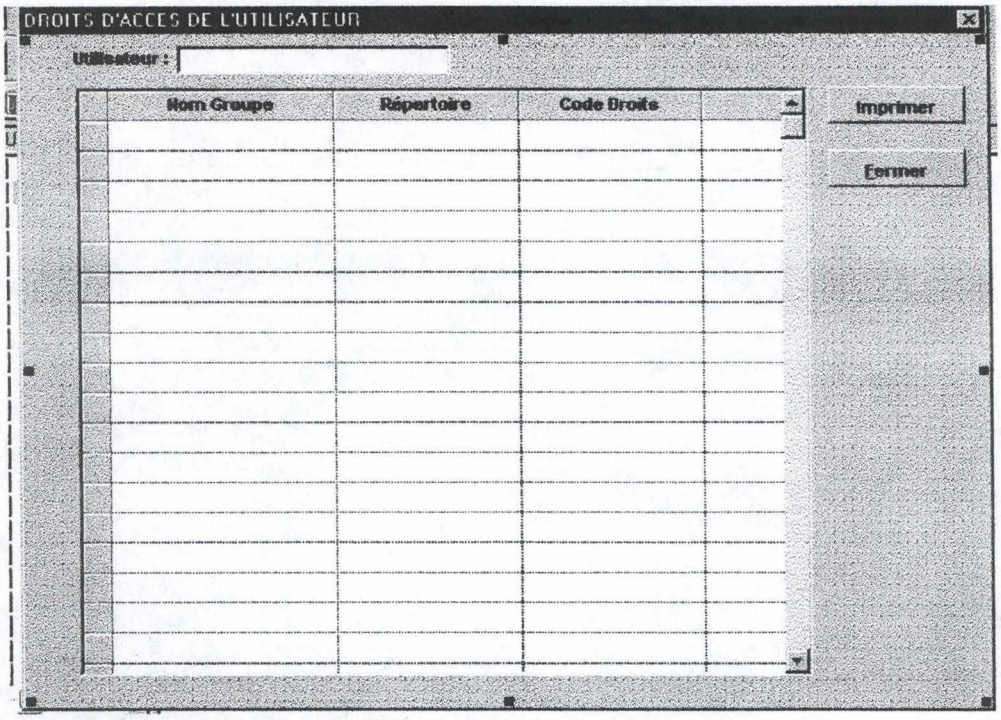
Un double-clic sur les tableaux vous permet d'obtenir des détails de l'élément concerné.

Ecran de l'historique des affectations des soft et hard à l'utilisateur



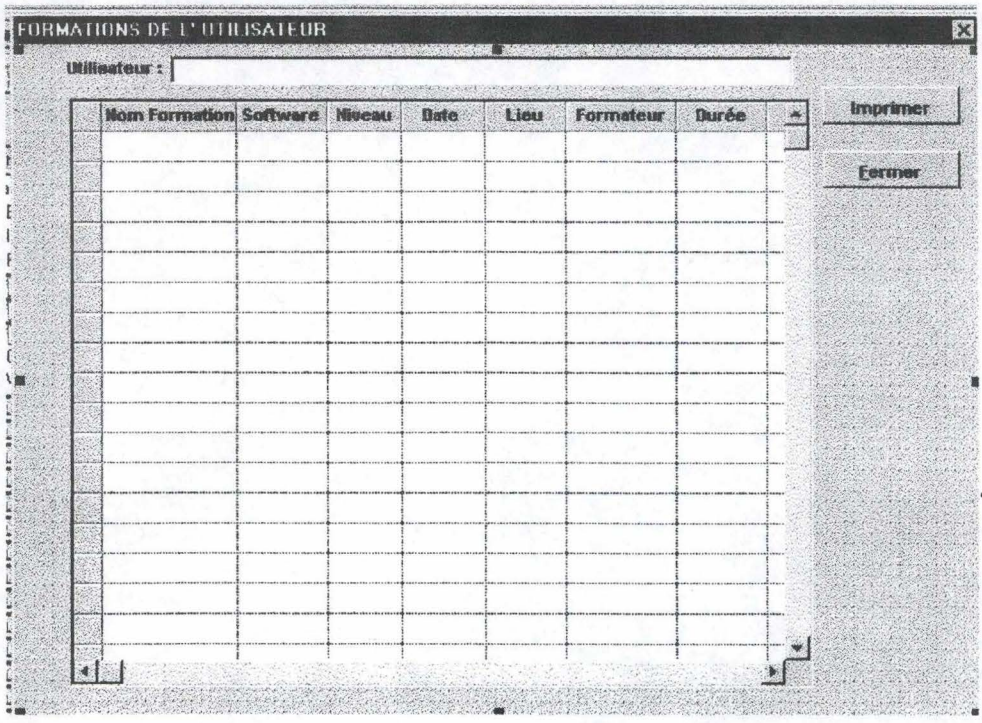
Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cet historique.

Ecran des droits d'accès de l'utilisateur



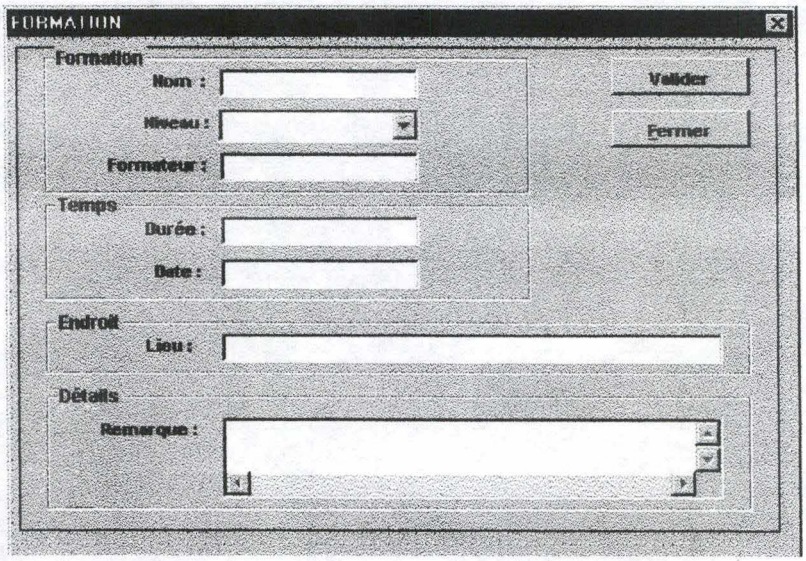
Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cet historique.

Ecran des formations de l'utilisateur



Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer cet historique.

Ecran de sasie de formation de l'utilisateur

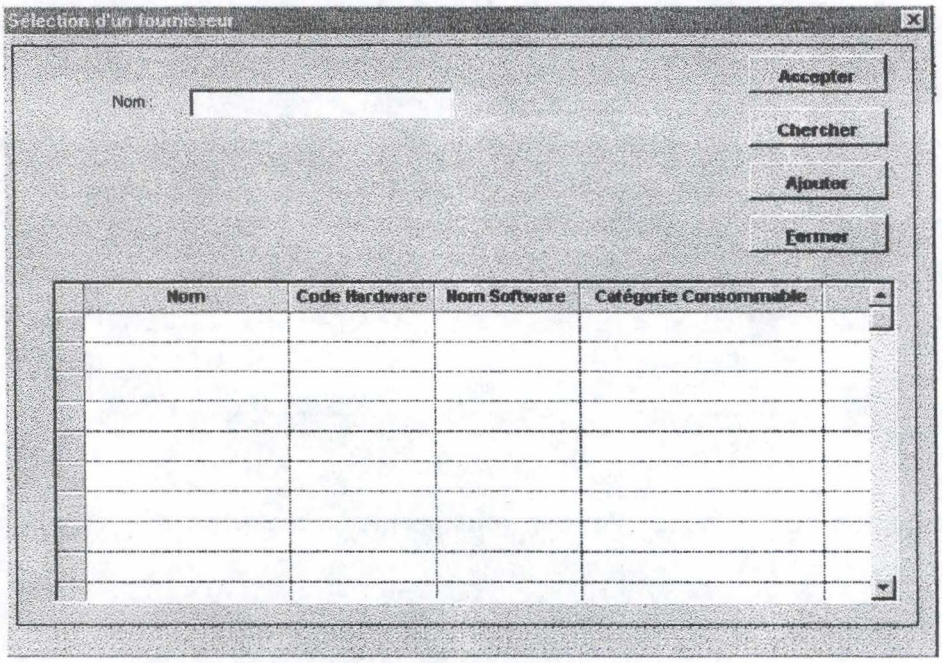


Un clic sur le bouton de commande **Valider** vous permet d'accepter la nouvelle formation saisie.

FOURNISSEUR

Ecran de sélection d'un fournisseur

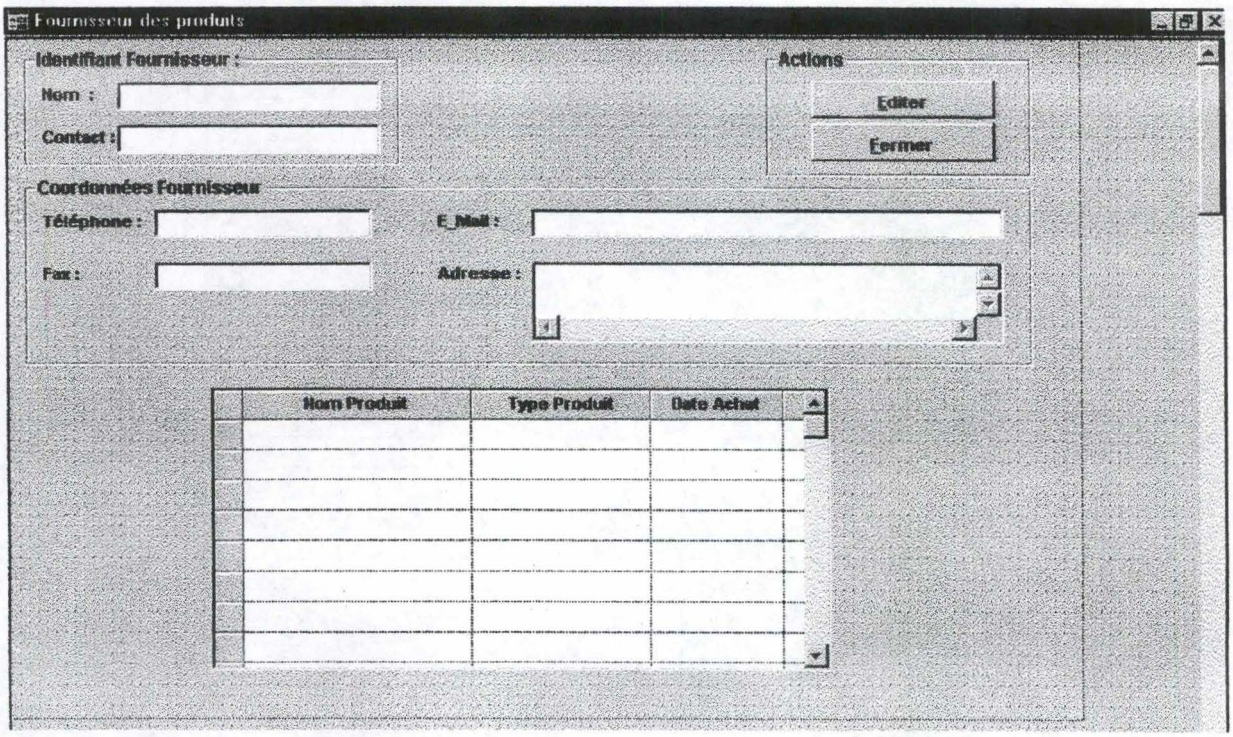
Cet écran permet de sélectionner un seul fournisseur parmi ceux affichés dans le tableau et qui répondent aux critères de sélection saisis et après un clic sur **Chercher**.



- Un clic sur le bouton de commande **Accepter** n'est valide qu'après avoir sélectionné une ligne dans le tableau par simple clic. Et cette action vous ouvre l'écran des données du fournisseur sélectionné.
- En constatant que votre fournisseur cherché ne s'y trouve pas ou que vous voulez ajouter un nouveau fournisseur, ouvrez l'écran des données du fournisseur vide pour l'ajouter par un clic sur le bouton de commande **Ajouter**.
- Un clic sur le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.

Ecran des données du fournisseur sélectionné

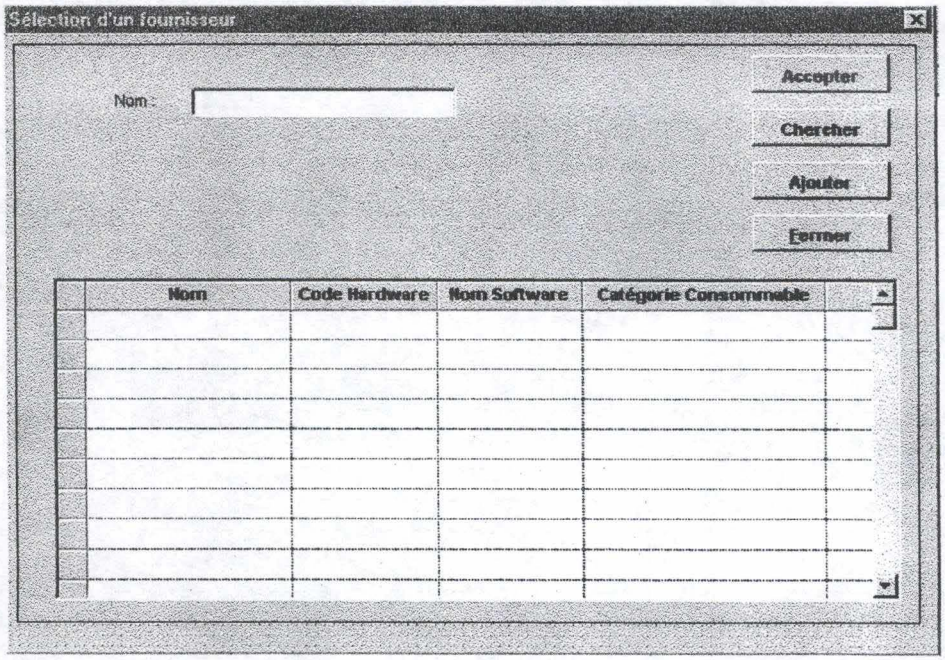
Si cet écran est ouvert en mode *AJOUT*, vous n'avez qu'à saisir les données et appuyer sur le bouton de commande **Confirmer** pour valider les saisies.
Si par contre, il est ouvert avec des données du fournisseur sélectionné à l'écran précédent, vous pouvez cliquer sur le bouton de commande **Editer** pour modifier vos données sauf l'identifiant.
Le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.



CONSOMMABLE

Ecran de Sélection d'un produit consommable

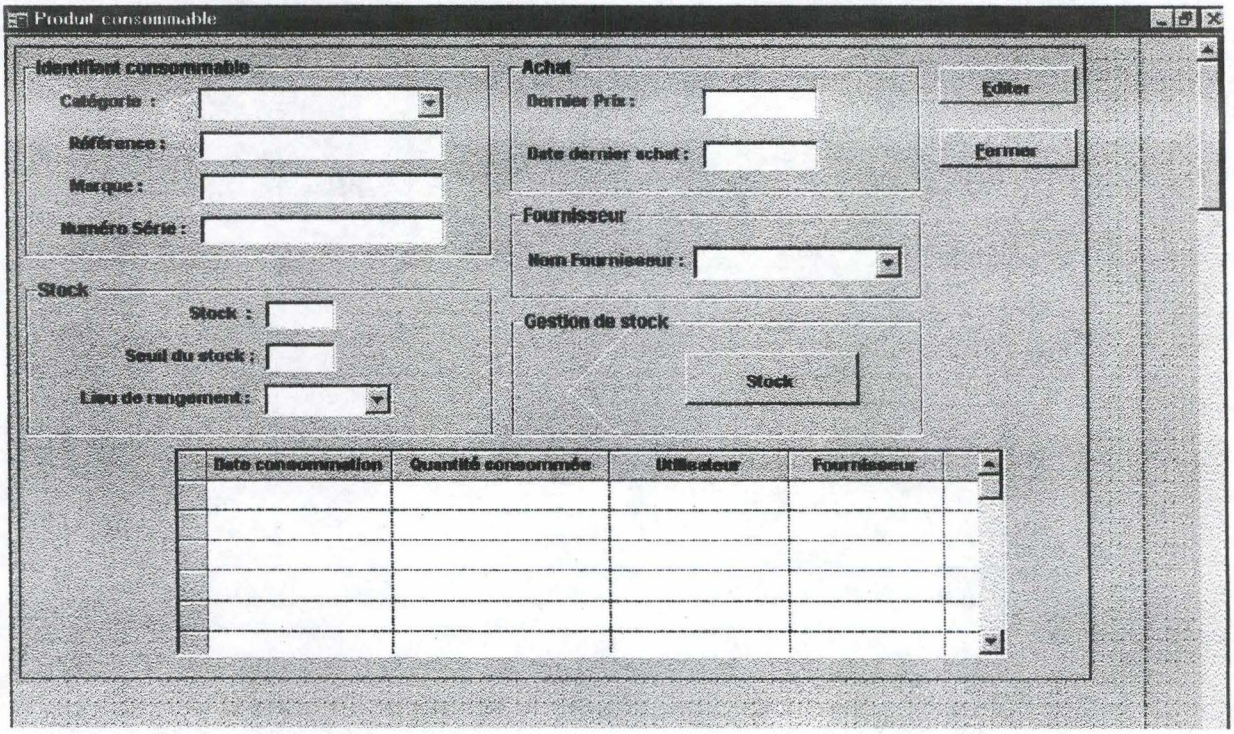
Cet écran permet de sélectionner un seul consommable parmi ceux affichés dans le tableau et qui répondent aux critères de sélection saisis et après un clic sur **Chercher**.



- Un clic sur le bouton de commande **Accepter** n'est valide qu'après avoir sélectionné une ligne dans le tableau par simple clic. Et cette action vous ouvre l'écran des données du consommable sélectionné.
- En constatant que votre consommable cherché ne s'y trouve pas ou que vous voulez ajouter un nouveau consommable, ouvrez l'écran des données du fournisseur vide pour l'ajouter par un clic sur le bouton de commande **Ajouter**.
- Un clic sur le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.

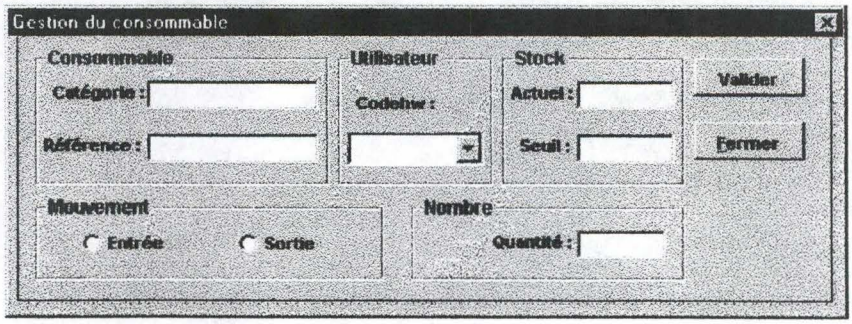
Ecran des données d'un produit consommable sélectionné

Si cet écran est ouvert en mode *AJOUT*, vous n'avez qu'à saisir les données et appuyer sur le bouton de commande **Confirmer** pour valider les saisies.
Si par contre, il est ouvert avec des données du consommable sélectionné à l'écran précédent, vous pouvez cliquer sur le bouton de commande **Editer** pour modifier vos données sauf les identifiants.
Le bouton de commande **Fermer** ferme cet écran et retourne à l'écran principal.



Un clic sur le bouton de commande **Stock** vous permet de faire la gestion de stock en ouvrant son écran.

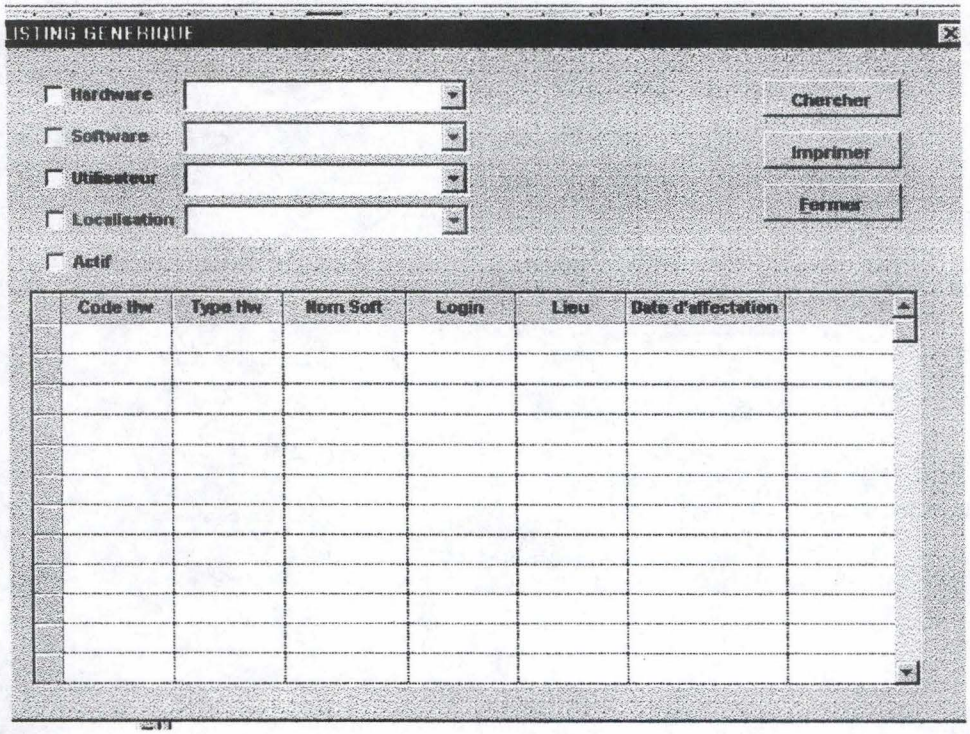
Ecran de gestion de stock du produit consommable sélectionné



Un clic sur le bouton de commande **Valider** vous permet d'accepter le mouvement sur le stock que vous venez de faire.

LISTING

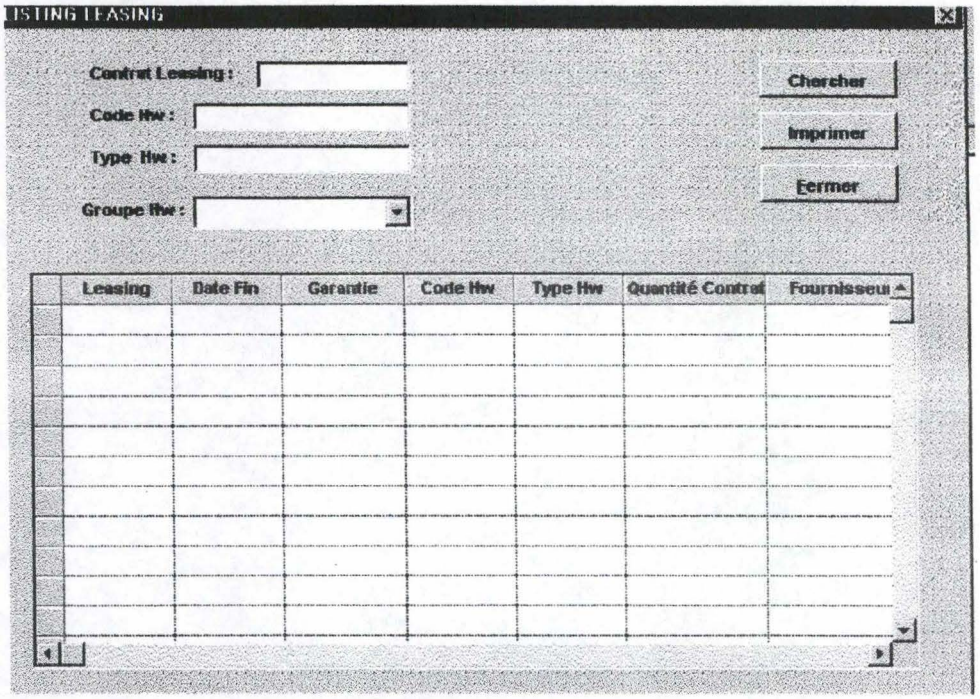
Ecran de listing générique (non fonctionnel)



Cet écran permet de visualiser les données de la base de données qui répondent aux critères de sélection saisis et après un clic sur le bouton de commande **Chercher**.

Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer ce tableau des données.

Ecran de listing de leasing de Hardware



Cet écran permet de visualiser les données de la base de données qui répondent aux critères de sélection saisis et après un clic sur le bouton de commande **Chercher**.

Un clic sur le bouton de commande **Imprimer** (non fonctionnel) vous permettrait d'imprimer ce tableau des données.