METHODE D'ANALYSE STRUCTUREE POUR LA MISE AU POINT DE LOGICIELS

Patrick Séchet
(Informatique)
(UR 502, Cadres Spatiaux de l'Indépendance Alimentaire)

EMBRAPA
Empresa Brasileira de Pesquisa Agropecuaria
SuperCenter Venâncio 2.000
70333 - Brasília, DF (Brésil)

RESUME - L'évolution actuelle des micro-ordinateurs, de plus en plus puissants et de moins en moins chers, conduit à une diffusion sans cesse croissante de l'informatique, qui envahit rapidement tous les secteurs de l'activité humaine. L'ORSTOM n'échappe nullement à ce phénomène, bien au contraire, de telle sorte que l'on y voit se développer des logiciels dans les domaines scientifiques les plus variés.

Au fur et à mesure que l'informatique étend son champ d'application, elle s'attaque nécessairement à des tâches de plus en plus complexes. Les réalisations artisanales actuelles, fait d'équipes restreintes (traditionnellement des chercheurs isolés), dont l'activité se situe essentiellement au niveau de la phase de programmation, devront progressivement faire la place à des développements de systèmes plus sophistiqués, mieux intégrés, susceptibles de satisfaire à des exigences de fiabilité, disponibilité et pérennité.

Etant entendu que les décisions majeures sont prises dans les étapes initiales, un transfert des efforts de réalisation vers les phases de conception et spécification (encore appelées respectivement analyse fonctionnelle et analyse organique) est indispensable, d'autant plus que la plupart des erreurs qui conduisent à un disfonctionnement du logiciel sont commises lors de ces étapes.

L'adoption d'une méthodologie rigoureuse pour la conception d'applications informatiques, quelles qu'elles soient, constitue un moyen efficace pour améliorer la qualité finale de celles-ci, en agissant essentiellement sur la communication, à plusieurs niveaux :

- au niveau du dialogue entre le concepteur (analyste) qui doit présenter un modèle logique de l'application (qu'il ne connaît généralement pas), et qui lui est décrite par le commanditaire (utilisateur), normalement peu soucieux de cette modélisation conceptuelle ;

- au niveau du dialogue entre l'analyste et le(s) programmeur(s), en fournissant une spécification précise de tous les objets qui seront manipulés dans l'étape suivante de programmation, ce qui se traduira par une diminution du nombre d'erreurs de cette phase ;

- au niveau du dialogue entre analystes, qui peuvent s'appuyer sur un langage commun pour débattre des aspects de représentation, spécifiques à l'application.

La méthode d'analyse structurée de systèmes, introduite par Chris Gane et Trish Sarson en 1977, fournit un ensemble de techniques de représentation graphique pour construire un modèle logique du système. Le schéma est développé par raffinements successifs depuis la vision globale de l'application, jusqu'à la représentation des fonctions élémentaires.

L'outil principal de modélisation est le diagramme de flux de données, qui fournit une vision graphique des relations entre les divers composants de l'application, permettant alors à l'analyste, à l'utilisateur et au programmeur de percevoir les diverses parties du système à partir d'une même structure modulaire.

L'analyse fonctionnelle élaborée dans la première étape de cette méthodologie conduit naturellement à une participation active de l'utilisateur lors de la définition des flux de données, prélude à la construction du dictionnaire de variables. A l'issue de cette phase, un document (le projet logique) est produit et va constituer la base de travail de l'étape suivante de spécification.

L'approbation, par l'utilisateur, de ce document fige la version du système à mettre au point : l'adéquation du produit dépend directement de la bonne compréhension du projet logique par l'utilisateur, d'où l'importance de cette participation.

Il est évident que l'adoption d'une telle méthodologie est un préalable indispensable à l'utilisation de techniques nouvelles relevant du génie logiciel. Celles-ci sont susceptibles d'augmenter substantiellement la productivité de la phase de programmation, goulot d'étranglement traditionnel pour la mise au point des applications informatiques, en offrant un environnement plus approprié et mieux intégré avec les autres phases.

L'utilisation d'une méthode unique pour la conception formelle de systèmes informatiques à l'Institut, est aussi de nature à modifier profondément l'approche de l'outil informatique par les scientifiques, comme le démontre les premières expériences qui ont été tentées.

Elle devra offrir une alternative au mode de prototypage qui s'est généralisé jusqu'à maintenant, favorisé par la carence chronique d'informaticiens, et aider à la mise au point de logiciels spécialisés, dont l'utilisation systématique et rationnelle par

l'ensemble du personnel scientifique est indispensable pour garantir le développement de notre recherche scientifique.

INTRODUCTION

L'essor spectaculaire de l'informatique, principalement depuis l'apparition des microprocesseurs il n'y a guère plus de dix ans, a eu pour conséquence une diversification considérable de son champ d'application, de telle sorte que le logiciel est devenu un constituant d'un nombre de plus en plus important de produits industriels et/ou de services.

Un institut de recherche comme l'ORSTOM est somme toute une entreprise comme une autre, dont la finalité est de produire des connaissances et de l'information en général : comme telle, elle investit des moyens relativement importants dans des opérations de logiciels qui accompagnent ses actions scientifiques de génération de connaissances.

La production de logiciels est ainsi une activité qui doit se développer considérablement dans les années qui viennent à l'Institut, compte tenu de l'étendue des disciplines couvertes et de leur spécificité d'une part, des conditions particulières d'intervention et de réalisation de ces activités d'autre part.

Au fur et à mesure que l'outil informatique se généralise et atteint progressivement tous nos secteurs d'activité, les logiciels à élaborer deviennent de plus en plus importants, non seulement parce que l'on attaque naturellement des tâches à automatiser de plus en plus complexes, mais aussi parce que les utilisateurs potentiels, mieux informés, deviennent de plus en plus exigeants sur la fiabilité et la convivialité des produits qui leur sont destinés.

En se restreignant à des activités purement scientifiques, on peut classer *grosso modo* les logiciels à développer en deux grands pôles : les produits destinés à la capitalisation d'information (génération de bases de données) sans privilégier une application particulière de façon à n'en compromettre aucune, et ceux dont la finalité est l'exploitation des données dans l'optique d'interprétation de l'information correspondante. A priori, la première catégorie requiert normalement des services plus professionnels que la seconde, eu égard essentiellement à son caractère plus général.

L'introduction de l'informatique pour une activité déterminée à l'ORSTOM dépend normalement de l'initiative d'un chercheur et, en conséquence, du niveau de connaissances que celui-ci acquiert sur les techniques de traitement de données disponibles et applicables dans le domaine thématique qui l'intéresse. Cette phase constitue donc une première étape, dont l'acquis principal est de montrer la voie en mettant en évidence les données susceptibles de

faire l'objet d'un traitement informatisé et les fonctions qui peuvent être automatisées. Elle se caractérise par une faible productivité due à des tâtonnements et à une évolution progressive du logiciel produit, de toutes façons très personnalisé (il est notoire qu'un logiciel réalisé dans ces conditions survit rarement au départ de celui qui l'a conçu).

Cette évolution s'explique simplement par le fait qu'une fois satisfaits les besoins fondamentaux des utilisateurs, leurs imaginations travaillent et définissent de nouvelles nécessités. Cette phase peut donc être assimilée à une étape d'élaboration de prototypes successifs, pour laquelle l'utilisation de langages de quatrième génération est adéquate, car plus efficace, plus dynamique et partant mieux appropriée que le passage par une spécification formelle, qui conduit à des systèmes beaucoup plus longs à mettre au point.

L'examen de quelques expériences de développement de logiciels, déjà conclues ou encore en cours à l'ORSTOM, dénote une très faible productivité globale : il est bien sûr difficile de la chiffrer et de se livrer à des comparaisons, eu égard à la grande diversité des applications développées. On sait néanmoins que dans la plupart des cas un système requiert plusieurs années pour sa mise au point.

Par ailleurs, au moment où *"qualité"* est devenu un mot d'ordre pour toute activité économique, la qualité du logiciel doit retenir l'attention des concepteurs et projettistes.

L'amélioration de la productivité dans la mise au point de logiciels, le souci de la qualité et le besoin de maîtriser la complexité sans cesse croissante des systèmes à développer exigent que l'ORSTOM se donne les moyens d'acquérir le métier du développement de logiciel. Ce savoir-faire se caractérise essentiellement par l'adoption et l'application de méthodes pour la réalisation de toutes les activités nécessaires à l'élaboration de *"produit-logiciels"*.

La figure 1 présente les différentes étapes du développement d'un logiciel, selon la vision de Tom de Marco, l'un des fondateurs de l'analyse structurée (De Marco, T. 1978). Cet article a pour objectif de présenter une méthodologie susceptible d'être appliquée dans l'étape de conception d'un logiciel et les bénéfices que l'on peut espérer tirer de son adoption. Adaptée à partir des concepts avancés par l'école nord-américaine (Gane, C. et Sarson, T. 1979), cette méthode a été appliquée depuis deux ans par l'équipe franco-brésilienne du programme SISGEO pour la réalisation de logiciels dans des domaines aussi divers que climat, sols, végétation, documents cartographiques, images de satellites, etc..

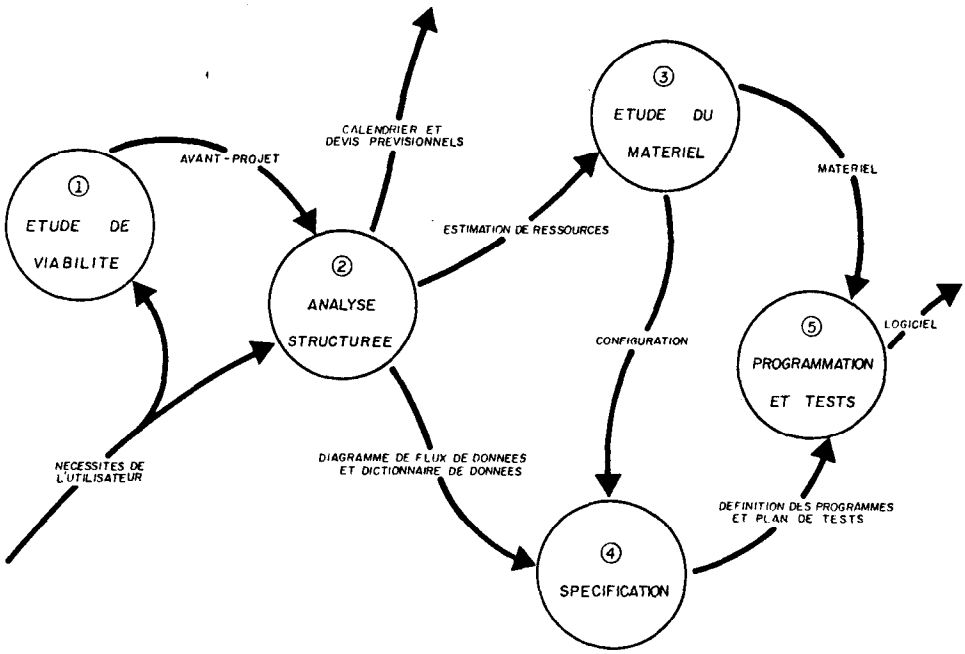


Fig. 1 - Etapes du développement d'un logiciel, vues par Tom de Marco (1978)

1. AVANTAGES

La définition et l'emploi d'une méthodologie répond à l'objectif de rendre plus homogène, donc plus efficace et rationnel, le développement des applications informatiques quel qu'elles soient. En ce qui concerne la seule partie de conception, l'adoption d'un ensemble de méthodes, normes et conventions répond aux préoccupations suivantes :

- *permettre le dialogue entre l'analyste et l'utilisateur.* Le succès d'une réalisation informatique est étroitement lié au degré d'intervention de l'utilisateur, à qui elle est destinée, dans son développement. Par conséquent, le souci majeur de la méthodologie de conception doit être de favoriser la compréhension qu'a un utilisateur des spécifications fonctionnelles établies par l'analyste. Elle doit en effet fournir un cadre apte à *définir clairement et enregistrer les nécessités informatiques de l'utilisateur.* On peut observer que dans la situation actuelle à l'ORSTOM, l'accès à l'informatique est réservé à des chercheurs qui ont consenti un effort particulier dans ce domaine : de ce fait, nombre de systèmes importants ne peuvent être mis au point, faute d'une connaissance technique suffisante des chercheurs intéressés. Dans la mesure où l'on dispose de quelques professionnels maîtrisant les techniques de développement de logiciel, l'adoption d'une méthodologie pour la spécification fonctionnelle devra *aider les scientifiques non initiés à l'informatique à accéder aux logiciels spécifiques dont ils ont besoin,* en jouant le rôle de responsable scientifique de l'application à mettre au point ;

- *permettre la documentation de l'application durant le développement.* Traditionnellement, la documentation de systèmes constitue le cauchemar de nos développeurs, de telle sorte que l'absence d'une documentation adéquate reste l'obstacle majeur à la valorisation des logiciels produits à l'ORSTOM. La plupart du temps elle est totalement inexistante ou, ce qui n'est guère mieux, elle se résume à l'insertion de quelques commentaires dans des programmes Basic ou Fortran, restreignant automatiquement le public atteint aux programmeurs dans ces langages. Dans les meilleurs cas, c'est à dire limités à une poignée de logiciels produits à l'Institut, un manuel d'utilisation est rédigé à posteriori, sans autre règle pré-établie que l'analogie avec les exemples de manuels qui accompagnent les progiciels du commerce. Il convient de noter que cet effort, louable au demeurant, reste incomplet dans la mesure où ce manuel ne sera d'aucun secours pour une éventuelle maintenance (évolution) postérieure. De la méthode d'analyse fonctionnelle résulte un ensemble de documents réunis dans le projet logique, qui constitue la base de la documentation du système : son

grand avantage est d'être produit à un moment où il peut servir de référence et de point de départ pour les phases suivantes (figure 1) ;

- *permettre le dialogue entre l'analyste et le programmeur.* En l'absence de tout document de spécification fonctionnelle ou organique, l'analyste est condamné à devenir à la fois le projettiste et le programmeur, ce qui l'oblige à assumer l'intégralité du développement du système. Cette situation s'est beaucoup répétée par le passé, de telle sorte que bon nombre d'analystes sont devenus esclaves de leur produit (*"analyste d'un seul système"*), faute d'avoir su partager les diverses tâches. Avec la complexité croissante des applications à développer, dans la plupart des cas on ne peut plus concevoir qu'un logiciel soit développé par une seule personne, de telle façon qu'une méthode rigoureuse est indispensable pour transmettre l'étape de conception (quelles fonctions seront automatisées ?) à l'étape de spécification (comment le seront-elles ?) ;

- *retarder au maximum la phase de programmation.* Tant que durent les premières phases de développement d'un logiciel (analyse fonctionnelle et analyse organique), l'impact d'une modification du projet est moins sensible, dans la mesure où aucune ligne de code n'a encore été écrite. Les échanges entre l'analyste et l'utilisateur pendant la phase de conception, puis entre ce dernier et le projettiste durant la phase de spécification, permettent d'examiner *"en profondeur"* les choix faits par les professionnels de l'informatique dans le but de satisfaire aux exigences du système. L'éventualité d'avoir une transformation importante à réaliser dans les phases postérieures du développement en est donc d'autant moins probable. Dans cette optique il est recommandable de ne pas commencer la phase de spécification tant que le projet logique n'a pas été dûment approuvé par toutes les personnes concernées. Avec cette méthode, le nombre d'erreurs fondamentales de programmation tend à diminuer sensiblement, ce qui compense normalement le temps passé aux spécifications ;

- d'une façon tout à fait semblable, le fait de suivre une méthodologie bien définie pour la conception du logiciel devra *permettre l'acquisition d'une configuration matérielle basée sur des critères rationnels et objectifs,* évitant ainsi toute précipitation pour l'achat d'un équipement, situation que l'on rencontre couramment ;

- *permettre de définir clairement ce qu'un logiciel fera.* Le principal objectif du projet logique est d'identifier les fonctions et processus que le logiciel à développer devra automatiser. Il est alors facile de savoir ce que le logiciel fera, et ce qu'il ne fera pas, et selon quelle logique, cette dernière étant également dans l'étape de conception par l'analyste ;

- la décomposition du logiciel en diverses fonctions ayant des objectifs distincts *permet le développement modulaire d'un système tout en conservant la vision d'ensemble* fournie par le projet logique. En effet, la complexité des systèmes constitue souvent un facteur qui empêche, ou pour le moins freine, leur réalisation : la structuration en processus, fréquemment indépendants, qui est réalisée au cours de l'étape de conception permet d'envisager une mise au point progressive, fonction après fonction, pouvant conduire à des versions successives de plus en plus élaborées et complètes du logiciel ;

- *fournir un argument pour suspendre toute modification pendant la réalisation.* Il est en effet souvent difficile de refuser une modification demandée par l'utilisateur pendant la réalisation du logiciel, même si celle-ci remet en question certains des choix déjà réalisés. Une fois le projet logique adopté, on peut renvoyer toute modification structurelle à une version postérieure du système ;

- *standardiser la conception des logiciels et en favoriser la valorisation.* Dans la mesure où l'on adopte une méthodologie et où l'on s'astreint à produire un document obéissant à des règles pré-établies, on permet une dépersonnalisation des systèmes élaborés et une normalisation favorable à la valorisation des produits mis au point. Dans cette optique, un manuel de normes et règles pourrait être élaboré de manière à *fournir une méthodologie pour la documentation des logiciels,* indispensable à leur institutionnalisation ;

- *produire des systèmes moins dépendants d'une réalisation physique.* Cet aspect constitue certainement un des éléments les plus positifs apporté par l'adoption d'une méthode de conception spécifique. Pour accompagner la dynamique actuelle de l'informatique, la plupart des applications développées ont eu à subir un certain nombre de migrations, de manière à s'adapter à de nouvelles configurations matérielles. Dans ces conditions, le projet logique établi à l'issue de la phase de conception est conservé, dans la mesure où il ne fait aucune référence à une configuration matérielle déterminée : il peut ainsi être regardé comme le document fondamental de l'application ;

- *fournir une voie pour l'utilisation de techniques de génie logiciel.* A l'heure actuelle le génie logiciel constitue, avec l'intelligence artificielle, l'un des domaines les plus effervescents de la recherche en informatique. Son but est d'apporter un soutien aux activités de développement de systèmes, en automatisant un maximum des tâches afférentes à cette activité. Un certain nombre de produits commerciaux sont déjà disponibles, dans ce domaine : aide à la conception, aide à la spécification, aide à la programmation ; tous ont pour base une méthodologie théorique existante (analyse structurée et variantes, projet structuré et variantes, etc.).

L'adoption d'une méthode est donc un préalable indispensable à l'utilisation des techniques de génie logiciel, susceptibles d'apporter un gain de productivité et de qualité significatifs lors de la mise au point des applications informatiques (Bourgeois, J. 1984).

D'autres avantages secondaires devraient également résulter de l'adoption d'une méthode de conception : certains sont évidents, comme le fait de *faciliter le dialogue entre les développeurs,* en leur permettant une communication plus technique grâce au langage commun qui est créé. Malgré la grande diversité des thèmes abordés à l'ORSTOM, chacun pourra facilement profiter de l'expérience de ses collègues, et faire bénéficier ceux-ci de sa propre expérience. D'autres avantages sont plus subtils, comme celui *d'autoriser un suivi de l'état d'avancement du projet,* alors que, traditionnellement, il est toujours très difficile de savoir où en est la mise au point d'une application, ni même d'avoir une prévision raisonnable de la conclusion d'une étape déterminée.

En se limitant à la seule phase de conception, on retiendra utilement tout l'intérêt que l'on peut tirer de l'élaboration d'un projet logique qui s'applique à définir exactement ce que le système fera, en faisant totale abstraction de toutes préoccupations liées à l'implémentation physique (restrictions de matériel, organisation physique de l'information, conditions d'accès, etc.), et sans compromettre les options qui pourront être prises ultérieurement en accord avec l'environnement matériel choisi ou disponible.

2. METHODE D'ANALYSE STRUCTUREE

2.1. La philosophie

La méthode de conception de systèmes qui est résumée dans cet article est adaptée de la version due à Chris Gane et Trish Sarson, qui a été publiée en 1979 dans l'ouvrage *"Structured System Analysis : tools and techniques"*. Celle-ci fait elle-même partie d'une école de méthodologies semblables (Yourdon, E. & Constantine, L.L., 1979), (De Marco, T., 1978), (Warnier, D.J. 1985), etc..

L'idée essentielle est de permettre la représentation graphique des fonctions d'un logiciel, et des données sur lesquelles ces fonctions agissent, de façon suffisamment simple et claire pour servir de véhicule de communication entre toutes les parties intéressées par la réalisation d'un système. Le diagramme correspondant peut alors être considéré comme un modèle logique du système à développer.

De cette façon, le problème principal (absence de langage commun entre l'analyste et l'utilisateur) est résolu, sans que l'analyste ait besoin de tout connaître de la spécialité dont relève le logiciel (il n'aura à poser des questions que sur les informations qui

doivent être manipulées, et qui sont représentées sur le diagramme), et sans que l'utilisateur n'ait plus à apprendre de l'informatique que les quelques symboles utilisés dans cette représentation graphique.

Une autre particularité de ces diagrammes est qu'ils permettent une approche descendante *"top-down"* du système, en autorisant une décomposition successive de chaque fonction mise en évidence dans une première représentation, pour fournir un nouveau diagramme de niveau plus détaillé.

Une fois les diagrammes élaborés, les entrée-sorties de données qu'ils mettent en oeuvre sont décrites et détaillées, de manière à construire le *"dictionnaire de données"*. Celui-ci constitue le second produit essentiel de l'étape de conception : il s'agit d'une structure adéquate pour recevoir tout le détail descriptif des données et de la logique des fonctions qui les transforment.

2.2. Les diagrammes de flux de données

Le diagramme de flux de données (DFD) est une maquette du système à développer : il doit donc représenter, en plus des données concernées (*flux*, ou flots) et des transformations qui leur sont appliquées (fonctions et processus), les personnes qui sont à l'origine, ou à qui sont destinées ces informations : ce sont les entités externes. Bien entendu, lors de l'analyse on est amené à identifier des étapes entre les processus pour lesquelles il est nécessaire de définir le stockage de certains éléments de données : le terme *"dépôt de données"* est utilisé pour représenter le concept logique de lieu de stockage.

Par conséquent, au total quatre symboles sont nécessaires pour construire le modèle logique du système (figure 2a). Un carré, dont les deux côtés en haut et à gauche sont représentés par des traits épais, symbolise une entité externe. Les flux de données sont représentés au moyen d'une flèche, de préférence horizontale ou verticale, dont la pointe indique la direction du flux. Les fonctions figurent sous la forme d'un rectangle debout, avec les coins arrondis : elles sont identifiées par un numéro situé dans la partie supérieure du rectangle. Enfin, le dépôt de données est symbolisé par une paire de lignes parallèles horizontales reliées à une seule des extrémités, et est identifié par un D majuscule suivi d'un numéro de séquence.

Chaque symbole sur le diagramme de flux de données reçoit un nom et une description, lesquels doivent être choisis de façon à être suggestifs pour l'utilisateur :

Fig. 2a – Les diagrammes de flux de données sont construits avec seulement quatre types de symboles

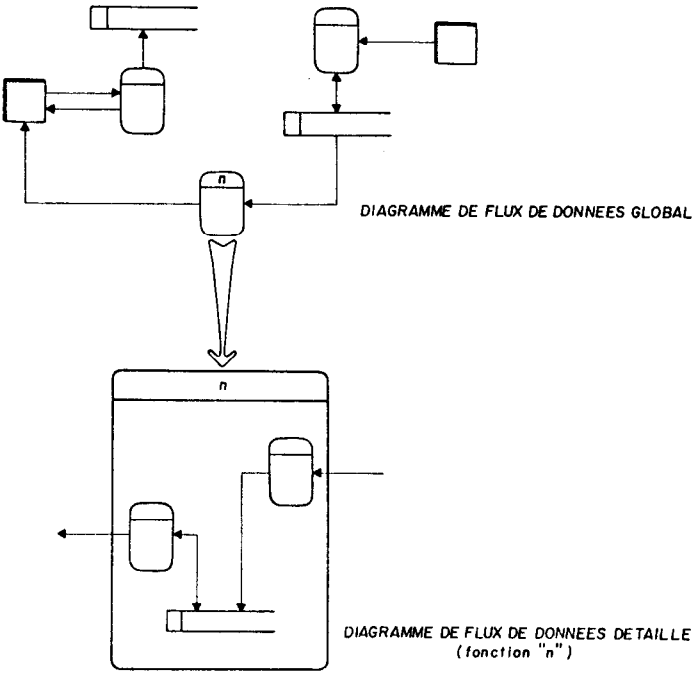


Fig. 2b - Chaque fonction du diagramme de flux de données global peut être "explosée" en un nouveau DFD

- *l'entité externe* est normalement une catégorie de personnes qui se communiquent avec le système : on utilise alors un terme générique (Hydrologue, Pédologue, par exemple). Il peut aussi s'agir d'un autre système qui reçoit ou fournit des informations au logiciel en cours d'analyse ;

- *le flux de données* doit être identifié de forme globale par une description qui caractérise le mieux possible son contenu. Par exemple : Sollicitation de calculs, Détails de jaugeages, Données d'identification, Caractéristiques de transaction, etc.. En règle générale, un même flux de données peut intervenir plusieurs fois sur les DFD, par exemple à l'entrée d'une fonction "Mettre à jour" et à la sortie de cette même fonction, en ayant le dépôt de données comme destination. On conviendra alors de ne faire figurer qu'une seule fois sa définition, de manière à éviter le risque de répétition dans la phase ultérieure d'analyse de données (Cf. 2.3) ;

- *la fonction* reçoit une description résumée qui se présente sous la forme d'une sentence impérative : verbe à l'infinitif + objet. Par exemple : Calculer les débits journaliers, Mettre à jour les caractéristiques, Emettre bulletin mensuel, Générer vecteur régional. On évitera les verbes comme "traiter" ou "réviser" qui montrent une connaissance encore insuffisante de la fonction en question, ce qui laisse supposer qu'une nouvelle décomposition est nécessaire. Par ailleurs, "ordonner", "classer" ou "trier" doivent également être évités, car ils représentent des fonctions sans valeur logique (simple réorganisation physique d'un fichier, qui n'a d'objet qu'après un choix de séquence ou d'organisation). La même réflexion s'applique pour les processus de simples copies et récupérations de sécurité (*"back-up/restore"*). Certaines des fonctions mises en évidence sur le diagramme peuvent être décrites comme "manuelles", lorsque le processus exercé (type Préparer les données, Extraire, etc.) est mal défini et, partant, non passible d'une automatisation, du moins dans l'état actuel des connaissances que l'on en a. On préférera toutefois remplacer ce type de fonction si possible par une entité externe (type : Coordinateur de transcription, Préparateur de cartes à digitaliser, etc.) ;

- *le dépôt de données* est identifié simplement par le nom qui caractérise son contenu : Tables, Etalonnages, herbier, Unités cartographiques, etc.. On évitera les termes inutiles comme Fichier de .., Cadastre, (Tables) du système.

Pour ne pas compliquer le diagramme de flux de données avec des lignes qui se croisent, la même entité externe ou le même dépôt de données peuvent être dessinés plus d'une fois sur le même diagramme : les deux (ou plus) symboles par entité externe sont identifiés par une ligne inclinée dans le coin inférieur droit du carré, tandis que les deux (ou plus) symboles par dépôt de données reçoivent une ligne verticale additionnelle à gauche.

Lors de l'élaboration d'un diagramme de flux de données initial, l'analyste doit se préoccuper de montrer le cadre général du système : il est donc indispensable de renvoyer la description de certains détails (conditions d'erreur ou d'exception, par exemple) à un niveau inférieur. De cette façon, chacune des fonctions illustrées sur le DFD de niveau 0 (encore appelé diagramme de contexte) peut être explosée sur un diagramme de flux de données de niveau inférieur (niveau 1), plus détaillé (figure 2b).

Si nécessaire chaque processus détecté à la suite de la décomposition d'une fonction peut être de nouveau subdivisé pour présenter un niveau supérieur de détails. Cette décomposition n'est toutefois généralement pas indispensable : il existe un certain nombre de règles, tant pour l'élaboration des DFD que pour en traiter les explosions successives.

2.3. Le dictionnaire de données

A mesure que l'on augmente les détails de l'analyse fonctionnelle, qui a pour résultat les DFD, on est amené à préciser la logique des fonctions, les détails des flux de données, le contenu des dépôts de données, etc.. Le dictionnaire des données constitue la structure adéquate pour recevoir les détails descriptifs des éléments des diagrammes de flux de données. Il est important d'observer que pendant toute cette phase on continue à se maintenir au niveau logique, en évitant de dessiner un format de bordereau, d'esquisser un gabarit de rapport d'éditer un *"lay-out"* de fichier!

Le dictionnaire des données du système peut être élaboré dès que les DFD sont considérés comme satisfaisants : l'étape correspondante est appelée analyse des données, et consiste à reprendre chaque diagramme et décrire successivement tous les symboles rencontrés dans la représentation graphique. On est alors conduit à définir avec précision :

- *les entités externes.* Il y a normalement peu d'entités externes pour un logiciel donné : en général on pourra se satisfaire de la liste de leur noms (tels qu'ils apparaissent sur les DFD) et de la description correspondante (normalement donnée par l'utilisateur), classée par ordre alphabétique. Dans le cas où une entité externe représente un autre système, il pourra être nécessaire de préciser son environnement sur le plan opérationnel et s'il existe un processus d'interface déjà prévu ;

- *les fonctions* sont classées conformément à leur numéro d'identification sur les DFD. En plus du numéro d'identification et de la sentence qui définit la fonction sur le diagramme, le dictionnaire de données devra contenir, pour chaque fonction, une description succincte et une spécification, aussi précise que possible, de la logique interne du processus. Celle-ci

pourra être exprimée sous la forme de langage structuré, tables ou arbres de décision. Pour éviter une description logique très importante dans le cas des fonctions les plus complexes, on pourra faire référence à des algorithmes pour lesquels un sous-chapitre spécial est ouvert dans le projet logique. Enfin, on conviendra de placer systématiquement, comme première clause de la description fonctionnelle, la nécessité éventuelle d'exécuter antérieurement une autre fonction, de façon à traduire l'existence d'une séquence logique entre deux processus ;

- *les flux de données.* Il s'agit de structures de données possédant une origine et une destination. Dans la mesure où un flux entre ou sort par rapport à une fonction, on lui attribue le code de la fonction suivit de lettre E ou S. En effet, il ne peut y avoir de flux de données entre deux entités externes, entre deux dépôts de données ou entre une entité externe et un dépôt de données sans intervention d'une fonction. Il est important de noter dans le dictionnaire, pour chaque flux, sa périodicité et une estimation du volume qui intervient à chaque exécution de la fonction, de façon à fournir une orientation utile pour la spécification et le projet physique. La description du contenu du flux de données est réalisée en détaillant la liste des éléments de données qui composent ce flux, chaque élément de données étant lui-même défini dans le dictionnaire de données sous la rubrique *"champs"*.

Tous les *champs* qui apparaissent dans les flux de données, mais aussi dans l'expression de la logique des fonctions doivent être décrits : toutefois, les champs de contrôle, c'est à dire ceux qui sont seulement utilisés pour les nécessités internes du système n'y figurent pas. Il est intéressant de prévoir un formulaire spécifique pour recevoir les nombreuses caractéristiques de chaque champ, de façon à faciliter l'élaboration du dictionnaire de variables. Ces caractéristiques peuvent être regroupées selon les catégories suivantes : identification, pseudonymes, structure, origine, stockage, domaine de valeurs, taille et représentation, codification, validation.

L'identification d'un champ est constituée par un *symbole* mnémonique, choisit suffisamment court pour ne pas compromettre l'utilisation de tel ou tel langage au moment de la programmation (huit lettres pourrait être un maximum). Le choix de ce symbole en temps de conception, et son appartenance au dictionnaire de variables est une garantie évidente d'homogénéité postérieure.

La *description*, aussi précise que possible, est normalement obtenue auprès de l'utilisateur, ainsi que le *nom d'usage courant*, optionnel, destiné à conserver comme pseudonyme la désignation usuelle traditionnelle dans la discipline (par exemple : TX et TM pour les températures maximales et minimales, sont des symboles consacrés par les climatologistes).

La *structure* d'un champ est destinée à maintenir, dans le dictionnaire de variables, une information sur les champs composés et la relation entre les divers éléments d'une même structure. Elle pourra être représentée par un ensemble comme XX.Y, où XX représente le numéro de séquence d'un champ composé dans un groupe (flux ou dépôt) et Y le numéro de séquence d'un sous-champ dans ce champ. Par exemple :

```
DATE    01
JOUR    01.1
MOIS    01.2
ANNEE   01.3
```

L'origine d'un champ peut être spécifiée, à ce stade du développement, par le code du flux de données, ou de l'algorithme, qui lui donne naissance. Toutefois, dans la phase postérieure de spécification, il sera plus intéressant d'indiquer le code du formulaire (ou de l'écran) pour les champs qui proviennent d'un flux de données.

Le *stockage* est noté à partir du code d'identification du dépôt de données qui contient ce champ : de la même façon, en temps de projet physique, on pourra indiquer le code mnémonique du fichier correspondant.

Comme un même champ peut figurer dans plusieurs flux et être contenu dans plus d'un dépôt de données, il est indispensable de prévoir plusieurs occurrences, tant pour l'origine que pour le stockage.

Le domaine des valeurs possibles est implicitement défini par le *type* du champ, numérique ou alphanumérique. Un *intervalle de valeurs* peut normalement être donné pour les champs numériques : le fait de préciser les valeurs minimale et maximale d'une variable en permettra une vérification plus rigoureuse. Dans certains cas il est nécessaire de préciser une valeur (ou un état) par *défaut*. Bien entendu, *l'unité de mesure* doit également être notée en temps de projet logique, en accord avec les préférences de l'utilisateur : dans le cas où le logiciel prévoit la conversion d'unités de mesure pour une variable déterminée, on utilisera deux champs différents.

La taille d'un champ dont on connaît déjà le type sera simplement représentée, sans faire référence au matériel, par un *masque d'édition*.

Dans le cas où le champ possède plusieurs *occurrences* (tableau), on en notera le nombre pendant la phase de conception.

La codification d'un champ sera représentée simplement dans le dictionnaire de variables par le numéro d'identification de la *table* utilisée pour décodifier le champ en question.

La validation d'un champ fait l'objet d'une attention particulière lors du montage du dictionnaire de variables. Elle peut s'exprimer par l'ensemble des caractéristiques suivantes :

- *obligatoire.* Il s'agit d'un booléen qui signale si le champ correspondant doit être présent impérativement, ou non ;
- *validation spéciale.* Elle s'exprime sous la forme du code de l'algorithme de validation de ce champ. Par exemple, pour une longitude, le sous-champ hémisphère ne peut être que Est ou Ouest ;
- *cohérence.* Elle indique l'existence d'un algorithme que la valeur de ce champ doit respecter, en combinaison avec un ou plusieurs autres champs. Par exemple, en chimie des sols les divers résultats de l'analyse granulométrique de la terre fine (sable grossier, sable fin, silte et argile) doivent totaliser 100% pour un échantillon déterminé. Le code de l'algorithme en question devra bien sûr figurer pour chacun des champs qui y interviennent ;
- *erreur.* Cette colonne est utilisée pour signaler le cas d'un champ qui devra être accepté, même s'il a été considéré erroné par la critique. Ce cas intervient fréquemment pour des données historiques (climatologie, par exemple), lorsqu'il n'y a pas de possibilité de retrouver la valeur réelle, l'erreur étant alors tout au plus signalée ;
- *digit de vérification.* On signale l'existence d'un digit de contrôle pour le champ considéré.

Dans le sous-chapitre *algorithmes* sont regroupées toutes les formules de calculs mathématiques (généralement utilisées pour obtenir un élément de donnée en fonction d'un ou plusieurs autres), ainsi que les expressions en langage structuré et autres tables de décision, utilisées dans le système. Chaque algorithme reçoit un code (A, suivi d'un numéro de séquence à deux chiffres), une définition et la description mathématique de sa logique. Pour chaque algorithme il est essentiel de traiter tous les cas d'exception avec l'utilisateur, rencontrés lorsque l'un ou l'autre des arguments est inconnu (par exemple).

Les *dépôts de données*, enfin, constituent également des structures de données : ces structures sont cette fois statiques, alors que les flux de données étaient des structures dynamiques. Le contenu d'un dépôt de données sera donc décrit en fournissant la relation des noms de champs qui composent cette structure.

On précisera de plus le code et le nom du dépôt de données, ainsi qu'une description succincte de son utilité. A ce stade de la conception on pourra faire une observation sur la clé primaire, voire sur l'organisation physique : toutefois, la clé peut ne pas être unique dans la mesure où aucune normalisation n'a encore été faite.

Pour terminer, il convient de faire trois remarques importantes au sujet du dictionnaire de données :

- en premier lieu, en suivant scrupuleusement les règles dictées pour son élaboration, on garantit que tous les éléments nécessaires à la réalisation du système sont décrits, et en un seul endroit. La logique est trouvée dans la partie des fonctions, les entrée-sorties dans la partie sur les flux de données, les caractéristiques des données dans la partie des champs et les nécessités de stockage dans la partie sur les dépôts de données. Le projettiste va donc pouvoir y rencontrer toutes les spécifications dont il a besoin pour dessiner les fichiers, créer les écrans, élaborer les gabarits des états, etc., qui constituent les tâches du projet physique ;

- on a pris le plus grand soin, dans toute l'analyse des données, de ne jamais aborder le moindre aspect de la réalisation concrète, de manière à ne pas (trop) induire une solution physique particulière. Par conséquent, le projettiste reste entièrement libre de ses choix quant à l'organisation des fichiers, la structuration des tâches, les supports pour les entrée-sorties, etc.. De plus, la partie projet logique reste totalement indépendante de la version implémentée qui en sera faite dans un environnement matériel déterminé ;

- la partie dictionnaire de variables proprement dite peut être automatisée, compte tenu du nombre élevé de variables que certains systèmes peuvent atteindre (cent variables est un nombre très fréquent, cinq cents n'est pas si rare), et surtout de l'intérêt que l'on peut en tirer. En effet, on peut imaginer de nombreuses applications de ce dictionnaire : rapports totaux ou partiels, complets ou résumés, selon diverses classifications ; vérifications de cohérence et intégrité ; génération automatique de *"books"* ou *"includes"*, etc.. On notera cependant que l'essentiel de la valeur d'un dictionnaire de variables provient de ce qu'il constitue un dépôt central d'information sur le système, pour l'analyste, le projettiste et les programmeurs qui y travaillent. Ce dernier élément doit

être pris en considération lors du choix éventuel d'un logiciel d'administration de dictionnaire, en particulier en ce qui concerne la configuration le supportant (ressource distribuée ou non).

2.4. Le reste du projet logique

Pour compléter la documentation de l'étape de conception, un certain nombre d'éléments doivent être précisés, et font l'objet d'autant de chapitres du projet logique correspondant. En prenant exemple sur la méthodologie adoptée par la division de développement de systèmes de l'EMBRAPA, et appliquée par l'équipe de SISGEO, le projet logique pourrait avoir le sommaire suivant :

1. Introduction
2. Modèle d'entités/relations
3. Diagrammes de flux de données
4. Dictionnaire de données
5. Tactique de réalisation
6. Estimation de ressources
7. Glossaire
8. Signatures

Le premier chapitre, introduction, a pour objet de décrire succinctement le contexte général de l'application, essentiellement en termes de présentation de ce qui motive son développement. Les sous-chapitres suivants constituent cette introduction : historique (brève narration des faits qui ont conduit à la conception du système), objectifs, en termes généraux, du système et (justificatifs)/bénéfices attendus de sa mise au point.

Le chapitre suivant, modèles d'entités/relations, est constitué d'un diagramme schématisant ce modèle et de la liste alphabétique des entités (nom, description et identification du dépôt de données correspondant). Ce chapitre est normalement rédigé à la fin de la phase de conception, car il repose sur la normalisation des dépôts de données qui interviennent dans le logiciel, jusqu'à la troisième forme normale.

La première forme normale correspond à l'élimination des groupes répétés dans une structure de données déterminée. La deuxième correspond au regroupement de tous les champs qui dépendent d'une même clé tandis que la troisième forme normale suppose l'élimination des dépendances fonctionnelles entre champs distincts de la clé primaire, ce qui revient à isoler les tables de conversion qui expriment ces dépendances.

Le modèle d'entités/relations constitue une vision logique qui se traduit dans le projet physique par le *lay-out* de la base de données. Toutefois, ce dernier incorpore déjà certaines restrictions imposées par l'environnement de *"hardware et software"*.

Les chapitres 3 et 4 ont été amplement discutés dans les parties précédentes de cet article (§2.2 et §2.3). Le cinquième chapitre, tactique de réalisation, a pour but de fournir un calendrier prévisionnel pour la mise en place effective du logiciel : l'élaboration d'un diagramme de structure du système, mettant en évidence les principales fonctions indépendantes (sous-systèmes), sera utile pour la compréhension de la décomposition en étapes distinctes. Le second sous-chapitre, sur les restrictions, doit présenter toutes les limitations susceptibles de freiner, voire empêcher, l'élaboration du logiciel, ou encore en restreindre l'utilisation. Par exemple, ce pourra être une indisponibilité de personnel, l'absence de matériel adéquat, des restrictions administratives ou d'accès aux informations, etc.. Le dernier sous-chapitre doit aborder le mode d'implémentation, en spécifiant essentiellement si la réalisation doit être modulaire, quelles sont les fonctions prioritaires, quel est le calendrier prévisionnel.

Le sixième chapitre, estimation de ressources, poursuit l'objectif de déterminer, dès la phase de conception, la demande de ressources nécessaires à la mise en oeuvre du système. Il contient donc un certain nombre d'éléments de nature à fixer la configuration minimale et la configuration recommandée pour le produit en question. Dans le cas d'un logiciel destiné à être exécuté sur un micro-ordinateur, les caractéristiques suivantes devraient y figurer : type de CPU, taille de mémoire vive utilisée, nécessité éventuelle d'un coprocesseur arithmétique, nombre de drives, type de carte vidéo, type d'impression (sur 80 ou 132 colonnes), nécessité de périphérique spécifique comme traceur de courbes ou table à digitaliser, capacité de stockage minimale (disquette ou disque dur), système d'exploitation (version minimale) et autres logiciels nécessaires à l'utilisation de ce produit.

De la même façon, une estimation des ressources humaines (temps d'opération mensuel, par exemple) et des ressources financières requises pour le fonctionnement normal du système devrait être fournie dans ce chapitre.

Le septième chapitre, glossaire, a pour but de définir clairement chacun des termes utilisés et qui n'apparaissent dans aucun des éléments décrits dans le dictionnaire de données. Il se compose normalement d'une partie thématique (termes introduits par le spécialiste) et d'une partie informatique (termes employés par l'analyste). La connaissance des expressions et des concepts contenus dans le glossaire enrichit la propre compréhension du système.

La dernière partie est destinée à recevoir la signature des personnes responsables pour l'approbation du projet logique. Il est en effet indispensable que ce document soit approuvé par les deux parties, et que chacun soit conscient de ce que la mise en route des étapes suivantes de spécification et implémentation signifie en

termes de bénéfices apportés par l'automatisation, mais aussi d'efforts, de ressources impliquées, de délais à attendre, etc.. C'est donc une sorte de contrat pour la mise au point du logiciel que l'utilisateur signe avec l'analyste responsable de la conception, au nom de la totalité de l'équipe de développement.

3. EXEMPLE D'APPLICATION

Pour illustrer la façon d'utiliser la méthode de conception qui a été présentée dans cet article, on s'appuiera sur la production récente d'un logiciel d'administration de dictionnaire, PCDICO.

3.1. Enoncé

L'énoncé du problème peut être extrait d'un rapport de mission effectuée en mars 1987 à Manaus, auprès de Françoise Grenand, linguiste, responsable scientifique du projet :

... Il s'agit de la manipulation sur micro-ordinateur d'un dictionnaire de plus de deux mille mots. Un premier système rudimentaire avait été réalisé avec le concours d'un programmeur brésilien sur micro-ordinateur CP/M, de telle sorte que le fichier a été entièrement saisi. Toutefois les ressources d'administration de données (tris, mise à jour, édition de champs) sont précaires et ne permettent pas de réaliser de manière satisfaisante les corrections nécessaires, d'autant plus que le fichier (1,2 Mégaoctets) doit être réparti sur 06 disquettes.

La migration de cette application sur PC-compatible, avec disque dur, a été décidée et, à cette occasion, une nouvelle version du système sera mise au point et comportera des ressources complètes d'édition de champ pour l'inclusion et la mise à jour, une structure et une organisation de fichier plus appropriées, des ressources de tri plus performantes. Le système correspondant devra être programmé à Brasília, et prêt pour la mi-mai 1987. ...

3.2. Diagrammes de flux de données

Les diagrammes de flux de données qui ont été établis sont particulièrement simples. Au niveau 0 , diagramme de contexte de la figure 3a, deux fonctions sont représentées, l'une de mise à jour du dictionnaire accessible par le linguiste, l'autre des consultations accessibles par les utilisateurs.

Fig. 3a – Diagramme de contexte de PCDICO

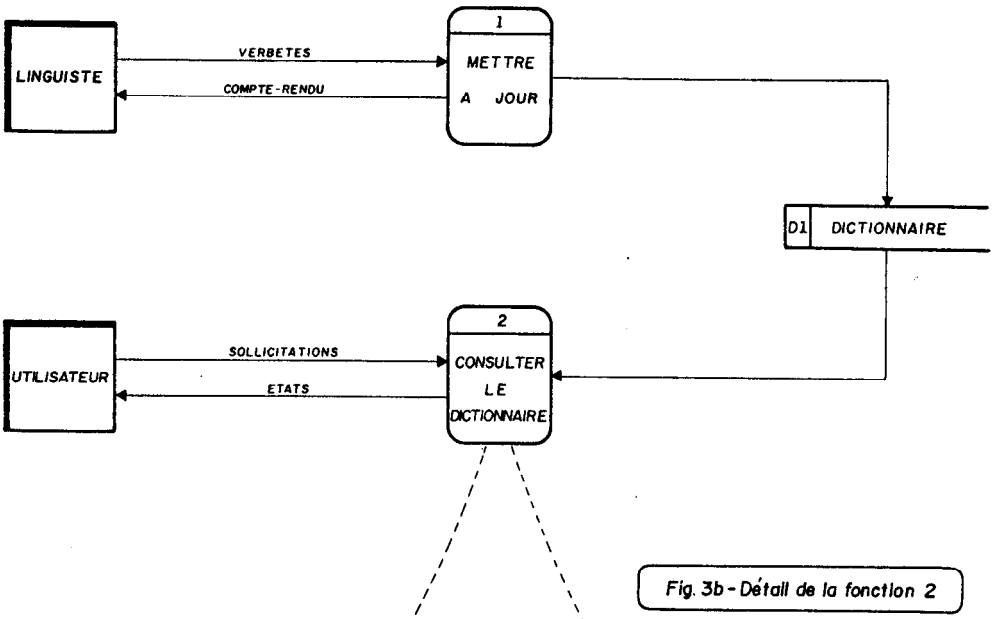
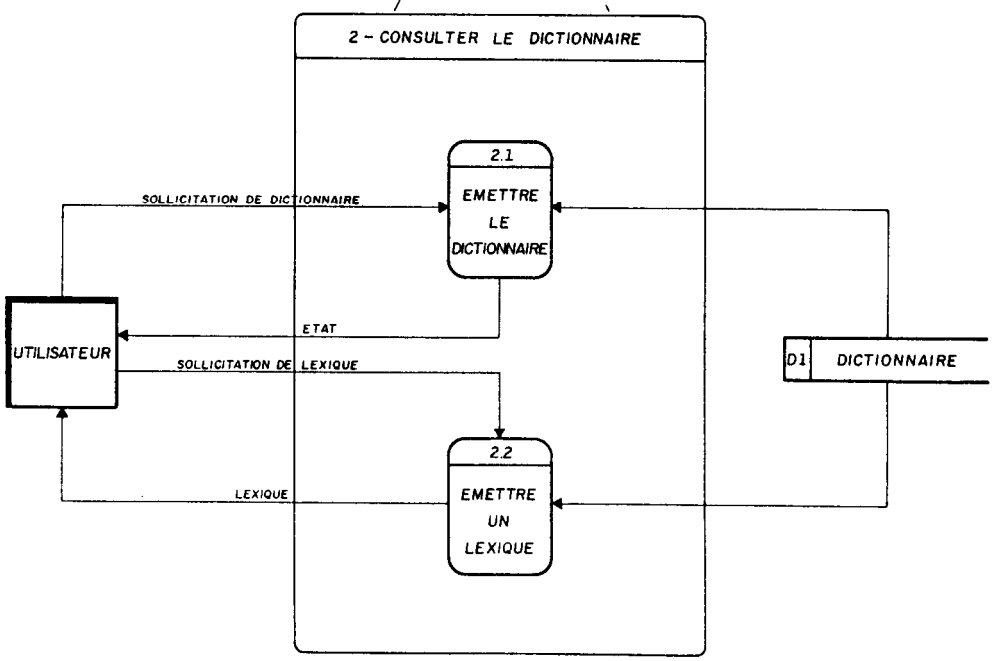


Fig. 3b – Détail de la fonction 2



A ce niveau, deux entités externes, un dépôt de données (le dictionnaire) et quatre flux de données sont mis en évidence. Pour faire apparaître deux types différents de sortie : état du dictionnaire (préparation à l'édition) et lexiques, la fonction *"Consulter"* a été éclatée et les flux de données correspondants précisés (figure 3b).

3.3. Dictionnaire des données

Le dictionnaire des données peut alors être élaboré.

a) définition des entités externes :

Linguiste - expert de la langue, responsable pour la constitution, l'administration et la maintenance du dictionnaire ;

Utilisateur - toute personne intéressée par la consultation et l'utilisation du dictionnaire, y compris l'éditeur et le propre linguiste.

b) définition des fonctions :

1 Mettre à jour : processus responsable pour la saisie, l'actualisation et la correction des données du dictionnaire. Celui-ci devra prendre en considération le problème de l'accentuation, c'est à dire manipuler les caractères accentués et le ç, représentés, par convention, sous la forme de deux caractères consécutifs (par exemple : 'e = é), de façon à conserver un ordre lexicographique plus naturel que l'ordre ASCII (ç juste après le c, en particulier).

2 Consulter le Dictionnaire : plusieurs types de consultation du dictionnaire seront autorisées : liste complète, liste spéciale pour l'édition, liste sélective des mots du dictionnaire, lexiques. Le résultat pourra être transmis sur écran ou sur papier.

2.1 Emettre le dictionnaire : production d'une liste des *"verbetes"* complètes, comme document préparatoire à l'édition. La sélection des verbetes selon divers critères est autorisée, pour permettre des recherches spécifiques.

2.2 Emettre un lexique : production d'un état simplifié, du type mot-sens (ou sens-mot).

c) définition des flux de données :

Verbetes (1.E) : ce sont les données qui altéreront le dictionnaire. Elles sont organisées en "verbetes".

- volume : il existe près de 2200 verbetes pour le *"pequeno dicionario de lingua geral"* ;
- périodicité : éventuelle ;

- champs : (attributs de verbete)
  Mot (vocable de la "Lingua Geral")
  ClasseMot (classe grammaticale du mot)
  Sens (en portugais, signification du mot)
  ClasseSens (classe grammaticale du sens)
  Exemple (exemple d'utilisation du mot)
  Etymologie (texte sur l'origine du mot)

Compte-rendu (1.S) : il s'agit d'un message sur une grille d'écran, qui confirme l'action effectivement réalisée sur le dictionnaire pour une transaction déterminée.

- volume : une grille d'écran par verbete altérée, insérée ou exclue ;
- périodicité : éventuelle ;
- champs : données de la verbete
  (voir 1.E) et message explicatif.

Sollicitation de dictionnaire (2.1E) : elle est constituée par une grille d'écran qui précisera les critères utilisés pour l'édition du dictionnaire (forme, support, sélection).

- volume : une grille d'écran par sollicitation ;
- périodicité : éventuelle ;
- champs : (diverses options pour l'émission)
  Forme (complet ou simplifié)
  Critère (champ composé)
  Condition de préfixe
  Condition de suffixe
  Condition de classe grammaticale
  Condition de mot
  Condition de longueur (de mot)
  Périphérique (écran ou imprimante).

Etat du dictionnaire (2.1S) : écran ou sortie d'imprimante contenant le résultat du processus d'édition d'états.

- volume : d'une page (ou un écran) à 250 pages (ou 500 écrans) par sollicitation. Dépend de la demande et du contenu actuel du dictionnaire ;
- périodicité : éventuelle ;
- champs : attributs de la verbete
  (voir 1.E) et message donnant les statistiques de l'édition.

Sollicitation de lexique (2.2E) : elle est constituée par une grille d'écran qui précisera les critères utilisés pour l'édition du lexique.

- volume : une grille d'écran par sollicitation ;
- périodicité : éventuelle ;
- champs : (diverses options pour l'émission)
  Forme (complet ou simplifié)
  Périphérique (écran ou imprimante).
  Classement (lingua geral ou portugais)

Lexique (2.2S) : écran ou sortie d'imprimante contenant le résultat du processus d'édition de lexiques.

- volume : d'une page (ou un écran) à 70 pages (ou 200 écrans) par sollicitation. Dépend de la sollicitation et du contenu actuel du dictionnaire ;
- périodicité : éventuelle ;
- champs : Statistiques de l'édition
  et, par verbete :
  Mot
  ClasseMot (éventuellement)
  Sens
  ClasseSens (éventuellement).

d) définition des champs :

Il s'agit de la première esquisse du dictionnaire de variables du système PCDICO, construit à partir des données identifiées dans le projet logique. Pour chaque variable on indique la structure (afin d'identifier les champs composés), un intervalle de validation, un masque d'édition et l'origine, sous la forme d'une nomenclature de flux (à remplacer par un numéro d'écran ou de formulaire au cours de l'étape de spécification). Le tableau 1 présente le dictionnaire de variables construit pour cette version du logiciel.

e) algorithmes :

- A01 : Si Prefixe1 est non blanc, et Prefixe2 est blanc alors Prefixe2 = Prefixe1 ;
- A02 : Si Suffixe1 est non blanc, et Suffixe2 est blanc alors Suffixe2 = Suffixe1.

f) dépôt de données :

La structure logique de l'information stockée et manipulée par le système PCDICO est représentée sous la forme d'un diagramme de relations entre entités (chapitre 2) :

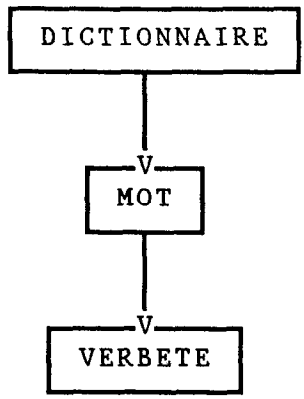
Tableau 1 : La phase de conception produit une première ébauche du dictionnaire de variables.

SYMBOLE	D E S C R I P T I O N	Stru-cture	Inter valle	Masque	Ori-gine	Cohé-rence
Mot	Mot de la lingua geral	1.01		X(36)	1.E	*
ClasseMot	Classe grammaticale du mot	2.01	T01	X	1.E	
Sens	Signification du mot (port.)	3.01	-	X(36)	1.E	
ClasseSens	Classe grammaticale du sens	4.01	T01	X	1.E	
Exemple	Texte, exemple d'util. du mot	5.01	-		1.E	
Ligne(Ex.)	Ligne du texte exemple	5.02		X(78)		
Etymolo.	Texte, etymologie du mot	6.01	-		1.E	
Ligne(Et.)	Ligne du texte etymologie	6.02		X(78)		
Critere	Critere de sélection d'édition	7.01			2.E	
Mot1	Début de sélection sur le mot	7.02	-	X(36)		
Mot2	Fin de sélection sur le mot	7.03	-	X(36)		
Prefixe1	Debut de sélection de prefixe	7.04	-	X(10)	A01	
Prefixe2	Fin de sélection de prefixe	7.05	-	X(10)	A01	
Suffixe1	Debut de sélection de suffixe	7.06	-	X(10)	A02	
Suffixe2	Fin de sélection de suffixe	7.07	-	X(10)	A02	
Classe	Classe grammat. sélectionnée	7.08	T01	X		
TailleMot	Taille choisie pour le mot	7.09	01 à 36	99		
FormeEtat	Forme de la sortie diction.	8.01	C.S	X	2.E	
Destinat.	Périphérique de sortie	9.01	E.I	X	2.E	
Ordre	Ordre de classement de sortie	10.01	L.P	X	2.E	

* dans le cas où le mot se présente avec des accents, le nombre de caractères utilisables est ramené à 36 moins le nombre d'accents ;

T01 désigne la table des classes grammaticales présentée ci-après (elle peut varier d'une langue à l'autre) :

code :	A	signification :	ADVERBIO
	C		CONJUNCAO
	D		SUBST.DEP.
	F		SUFIXO
	G		PART.GRAM.
	J		ADJETIVO
	N		NUMERAL
	P		PRONOME
	R		PREPOSICAO
	S		SUBST.
	V		VERBO



On appelle *"Dictionnaire"* l'ensemble des relations connues entre tous les vocables décrits de la *lingua geral* (appelés *"mot"*) et les significations correspondantes en portugais, appelées *"sens"*. On convient de désigner sous le nom de *"verbete"* le registre d'un sens pour un mot déterminé : le dictionnaire peut alors être considéré, indifféremment, comme une collection de verbetes, ou une collection de mots (regroupant toutes les verbetes qui ne sont que plusieurs sens attribués au même mot).

3.4. Commentaires

Pour compléter le projet logique du logiciel PCDICO, il suffit d'établir une tactique de réalisation (chapitre 5) et de faire une estimation de la configuration minimale pour opérer le système (chapitre 6).

Bien que particulièrement simple, cet exemple appelle quelques remarques :

- on remarquera que le projet logique est volontairement maintenu à un niveau conceptuel très général, en évitant de représenter des détails susceptibles de figer certains choix physiques. Par exemple, on pourra constater que l'on n'a pas précisé de quelle façon la table des classes grammaticales doit être manipulée (une gestion complète pouvant faire l'objet d'une fonction importante), ou encore on n'a pris aucune décision sur la sécurité des fichiers (fonction de copie/récupération), qui pourra donc être intégrée, ou non, au progiciel développé ;
- lorsque l'on définit deux entités externes ayant des prérogatives différentes, comme c'est le cas entre le linguiste et l'utilisateur, on sous-entend qu'un contrôle d'accès devra être implémenté, sous une forme ou une autre ;
- la description de la logique des fonctions n'est faite que lorsqu'elle est réellement nécessaire. Il eût été peu rationnel, par exemple, de décrire le détail du processus de mise à jour du fi-

chier dictionnaire, même à niveau logique, dans la mesure où le responsable de l'implémentation sait parfaitement de quoi il s'agit ! Il faut néanmoins tempérer cette observation, dans la mesure où cette logique n'apparaît qu'à cet endroit, l'utilisateur pouvant y trouver une utilité.

Ce logiciel a été programmé durant le mois de mai 1987 et une première version implantée début juin. Quelques modifications de détail ont ensuite été réalisées, de sorte que c'est une version 1.3 qui a finalement été exploitée en vraie grandeur, à partir du mois d'août pour préparer la publication du dictionnaire de *"lingua geral"*.

Le développement d'une nouvelle version, incluant en particulier la définition et la gestion complète d'un alphabet, sur l'écran comme sur l'imprimante, est envisagée dans un futur proche.

CONCLUSION

La méthode qui a été décrite est utilisée au sein du Département d'Informatique de l'EMBRAPA, depuis la fin 1984. Initialement, elle a été employée pour la réalisation de systèmes administratifs : comptabilité, patrimoine, etc..

Son application à des systèmes de soutien à la recherche, et donc de caractère scientifique, est une innovation due à l'équipe EMBRAPA/ORSTOM du programme SISGEO. Après les quatre premiers projets logiques (climat, végétation, documents cartographiques et images de satellites) déjà conclus, trois autres sont en cours de réalisation avec la même philosophie : sols, données socio-économiques et données phytosanitaires.

Parallèlement, la même méthodologie est utilisée dans le domaine génétique par d'autres équipes, pour des applications sensiblement différentes. L'expérience purement ORSTOM, acquise avec PCDICO, est en cours de renouvellement, avec un système très particulier, et plus complexe, destiné à rendre opérationnelle sur PC-compatible une méthode de synthèse et critique de données, mise au point par un chercheur de l'Institut.

La généralisation progressive de cette méthode, ou de toute autre proposition susceptible d'aboutir à la réalisation de modèles logiques pour les progiciels spécifiquement développés, pourrait constituer un guide précieux pour ce genre d'activités, pour l'instant plutôt artisanales. Dans cette optique, un cours destiné aux développeurs pourrait être préparé, et un guide rédigé et proposé comme recommandation de documentation de systèmes.

D'ores et déjà, il existe plusieurs produit-logiciels disponibles pour assister les professionnels dans la réalisation des travaux afférents à cette phase du développement d'un progiciel : ceux-ci donnent une nouvelle dimension à cette étape, principalement en permettant l'intégration de ses résultats pour la spécification et la codification. On peut raisonnablement en attendre des gains spectaculaires de productivité et qualité.

REFERENCES BIBLIOGRAPHIQUES

AFNOR : *Recommandation de Plan Qualité Logiciel.* Fascicule de documentation de la Normalisation Française, Z 67-130, 1987.

BOURGEOIS, Jacques : Ateliers de génie logiciel : état de l'art et perspectives. In : *Revue du Génie Logiciel*, 1, 1984. Agence de l'Informatique, Paris.

DE MARCO, Tom : *Structured Analysis and System Specification.* Yourdon Inc., New York, 1978.

EMBRAPA : *Manual de normas e procedimentos para o desenvolvimento de sistemas de informaçäo.* EMBRAPA/DIN, Brasília, 1985. Diffusion restreinte.

GANE, Chris : *Rapid System Development using structured analysis and relational technology.* IBPI Instituto Brasileiro de Pesquisa em Informatica, Rio de Janeiro, 1987.

GANE, Chris & SARSON, Trish : *Structured Systems Analysis : tools and techniques.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1979. 256p.

MARTIN, James : Desenvolvimento de aplicaçoes sem programadores. In : *Cadernos de informatica*, 4, 1981, Compucenter, Sao Paulo.

WARNIER, Jean-Dominique : *LCS Logica de construçäo de sistemas.* Datamec/Editora Campus Ltda, Rio de Janeiro, 1985. 191p.

YOURDON, Edward & CONSTANTINE, Larry : *Structured Design.* Yourdon Press, New York, 1978.