# Conception et réalisation d'un service reparti sur réseau pour la consultation de bases d'informations

Dominique Voboril

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE DE GRENOBLE (C.U.E.F.A.)

---

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR C.N.A.M.

en

INFORMATIQUE

par

Dominique VOBORIL

---

# CONCEPTION ET REALISATION D'UN SERVICE RÉPARTI SUR RESEAU POUR LA CONSULTATION DE BASES D'INFORMATIONS

Les travaux relatifs au présent mémoire ont été effectués à l'Institut d'Informatique et de Mathématiques Appliquées de Grenoble (I.M.A.G.) sous la direction de Monsieur Pierre LAFORGUE.

Je remercie,

Monsieur Claude KAISER, Professeur au Conservatoire National des Arts et Métiers, qui me fait l'honneur de présider ce jury,

Monsieur Jacques COURTIN, Professeur à l'Université des Sciences sociales de Grenoble, qui m'a suivi durant une partie du cycle C du C.N.A.M.,

Monsieur Louis BOLLIET, Professeur Emérite à l'Université des Sciences sociales de Grenoble, qui a su me guider tout au long des cycles B et C du C.N.A.M.,

Monsieur Samer HAJ HOUSSAIN, Ingénieur au Centre de Recherche Bull à Saint Martin d'Hères, pour l'attention qu'il m'a témoigné, pour ses remarques pertinentes et constructives ainsi que pour ses encouragements efficients,

Monsieur Pierre LAFORGUE, Ingénieur Réseaux et responsable du Service d'Etudes Logiciel de l'I.M.A.G., qui m'a spontanément accueilli dans son service en me présentant ce mémoire,

Monsieur François PELLERIN, responsable du service "système développement" du Département Informatique de la Société Merlin-Gerin à Grenoble, pour la confiance qu'il m'accorde depuis plusieurs années ainsi que pour sa participation au jury,

toutes les personnes du Service d'Etudes Logiciel, du centre de recherche du groupe BULL pour leur collaboration efficace et du Laboratoire de Génie Informatique , plus particulièrement Madame Christine BAUMANN, Ingénieur d'Etudes, pour son étroite collaboration et son soutien moral pendant toute la durée de ce mémoire ainsi que pour l'attention constante qu'elle a porté à ce travail,

tous ceux qui m'ont permis de consacrer une année à ce mémoire; la société Merlin-Gerin en autorisant mon absence, Madame NUGHES du C.U.E.F.A. et Madame AUBIN du FONGECIF Grenoble qui m'ont soutenue dans toutes les démarches administratives du congé-formation.

# TABLE DES MATIERES

TABLE DES MATIERES ..... 1

SOMMAIRE ..... 4

INTRODUCTION ..... 5

CHAPITRE 1 - LE PROJET ET SON ENVIRONNEMENT

1. Le projet ..... 8
2. Environnement matériel et logiciel ..... 9
   2.1. I.M.A.G. ..... 9
   2.2. UNIX ..... 10

CHAPITRE 2 - GESTION DES PROJETS DE RECHERCHE À TRAVERS UNE BASE DE DONNÉES RELATIONNELLE

1. Introduction ..... 14
   1.1. Organisation des laboratoires ..... 14
   1.2. Le Système de Gestion de Base de Données Oracle ..... 15
2. Développement de l'application ..... 16
   2.1. Méthodologie ..... 16
      2.1.1. Expression des besoins et analyse des données ..... 16
      2.1.2. Le schéma conceptuel ..... 19
      2.1.3. Codification retenue pour l'environnement Oracle sous Unix ..... 27
      2.1.4. Définitions des relations de la base ..... 35

2.2. Création de la base et mise en œuvre de l'application ..... 37

2.2.1. Rassemblement des données ..... 37

2.2.2. Automatisation du formatage des fichiers ..... 39

2.2.3. ODL (Oracle Data Loader) ..... 42

2.3. Difficultés rencontrées ..... 44

**3. Conclusion de la première phase ..... 46**

**CHAPITRE 3 - COMMUNICATION ET ACCES À DISTANCE**

**1. Introduction ..... 48**

**2. Architecture de communication ..... 49**

2.1. Le modèle de référence OSI ..... 49

2.2. Le réseau local Ethernet et les protocoles TCP/IP ..... 53

2.3. Programmation de bas niveau (en termes de "socket") ..... 56

2.4. Choix d'un schéma d'appel de procédures à distance ..... 61

2.4.1. Le schéma de communication de type RPC ..... 61

2.4.2. RPC (Remote Procedure Call) ..... 64

**3. Mise en œuvre et problèmes ..... 68**

3.1. Etude du projet ..... 68

3.2. Transformations nécessaires à la communication ..... 73

3.3. Choix de l'interface procédurale de communication avec Oracle ..... 75

3.3.1. Le code HLI (High Level Interface) ..... 76

3.3.2. PRO*C ..... 77

3.3.3. L'interface ORACLE "SOUPE" ..... 79

3.4. Réalisation du prototype ..... 82

3.5. Difficultés rencontrées ..... 87

**CONCLUSION ..... 88**

## ANNEXES

ANNEXE 1 - Organisation d'une base de données Oracle ..... 93

ANNEXE 2 - Exemple d'un fichier de création de table ..... 96

ANNEXE 3 - Formulaire de saisie ..... 97

ANNEXE 4 - Les protocoles TCP/IP: routage et fragmentation ..... 100

ANNEXE 5 - Serveur de multiplication via les "sockets" ..... 105

ANNEXE 6 - Serveur de multiplication via RPC ..... 110

**BIBLIOGRAPHIE ..... 115**

**INDEX ..... 117**

# SOMMAIRE

Une base de données relationnelle centralisée sur un ordinateur connecté à un réseau hétérogène met en évidence le problème de communication entre processus distants.

Ce mémoire analyse les données nécessaires à la mise en place d'une base relationnelle Oracle et propose une codification de l'environnement de ce SGBD sous un système Unix. Sont présentés, les outils d'Oracle utilisés pour la création de la base de données.

Un serveur a été réalisé dans un réseau local Ethernet reliant des machines fonctionnant sous Unix en utilisant un mécanisme d'appel de procédures à distance dans le but d'accéder à toute base de données Oracle.

Cette réalisation a soulevé les problèmes d'interfaçage entre les protocoles généraux de communication (TCP/IP) et les outils d'interrogation locale d'une base de données Oracle.

# INTRODUCTION

Ce mémoire a été effectué à l'Institut d'Informatique et de Mathématiques Appliquées de Grenoble (IMAG) au Domaine Universitaire de Saint Martin d'Hères.

Ce groupement est basé principalement sur une organisation en équipes et laboratoires au sein desquels se distinguent des unités de recherche classées par activité scientifique. Le besoin de rassembler des informations sur ce groupement a engendré la proposition de ce mémoire.

Une partie du travail réalisé consiste à créer une base de données relationnelle formée de l'ensemble de ces informations. Cette base doit pouvoir être consultée facilement au sein de la communauté que représente l'IMAG, ce qui implique des accès à distance.

Dans le cadre de ce travail, le réseau hétérogène de l'IMAG est employé pour faire communiquer les ordinateurs du réseau avec l'ordinateur ou se trouve centralisée la base de données.

L'autre partie du travail consiste à mettre en œuvre un prototype permettant d'assurer un service réparti sur le réseau pour la consultation d'une base de données gérée par le système de gestion de bases de données Oracle. L'étude et la réalisation de ce projet sont basées sur les outils du S.G.B.D. Oracle et le mécanisme d'appel de procédures à distance RPC™ des protocoles TCP/IP.

Nous présentons tout d'abord, dans le premier chapitre, le projet et son environnement. Les aspects matériel et logiciel sont explicités de manière à introduire le vocabulaire nécessaire à la compréhension de ce mémoire.

---

™ RPC: Remote Procedure Call

Le deuxième chapitre traite tout ce qui est relatif à la mise en place de la base de données IMAG. Nous présentons le schéma conceptuel des données que nous avons établi pour cette base. Nous proposons une codification des objets Oracle et des fichiers Unix© nécessaire à un tel environnement. Nous détaillons ensuite les moyens mis œuvre pour regrouper les données de la base.

Le troisième chapitre de ce mémoire traite la partie communication. Nous détaillons le mécanisme d'appel de procédures à distance en précisant les problèmes de représentation des données entre machines diffférentes. Nous présentons un exemple de programmation de bas niveau. Le même exemple est repris avec le mécanisme d'appel de procédure distante RPC. Oracle, nécessite une interface de communication. Après une étude des interfaces proposées par ce S.G.B.D., il a été choisi l'interface SOUPE® développée dans le centre de recherche BULL. Nous expliquons ici les choix de réalisation de notre implantation.

Nous présentons, pour finir, les extensions ultérieures que nous jugeons nécessaires, notamment au niveau de l'interface utilisateur sur les machines clientes.

---

© Unix: marque déposée de Bell Laboratories
® SOUPE: Simple Oracle User Programming Extension

# CHAPITRE 1

# LE PROJET ET SON ENVIRONNEMENT

# 1. Le projet

L'équipe du service "système développement" de Merlin Gerin dont j'ai fait partie pendant deux années consécutives, m'a permis de me familiariser avec certains outils de développement d'application. Mes connaissances sur les bases de données et les réseaux, acquises en cours du soir et durant ces deux années, ont suggérées la réalisation d'un projet concernant ces deux domaines.

Ainsi, l'institut de l'Informatique et de Mathématiques Appliquées de Grenoble a besoin de rassembler les informations relatives à tous les projets de recherche à savoir, les thèmes de recherche, les personnes, les équipes et les laboratoires concernés par chacun de ces projets. Ces données sont regroupées dans une base de données relationnelle centralisée sur un ordinateur connecté au réseau de l'IMAG.

Toutes les machines du réseau doivent être capables, à terme, d'interroger la base de données à travers un serveur.

Le travail consiste à mettre en place la base de données IMAG puis à concevoir et réaliser le serveur qui permet à tout programme d'application (interface homme/machine) d'accéder à distance ou localement à la base de données.

# 2. Environnement matériel et logiciel

## 2.1. I.M.A.G.

L' I.M.A.G. (Informatique et Mathématiques Appliquées de Grenoble) est un Groupement de Recherche comprenant plus de cinq cents chercheurs et ingénieurs. La variété des thèmes de recherche, et donc des besoins en matériel, a conduit à une hétérogénéité importante au niveau matériel. Des protocoles fédérateurs ont permis d'intégrer environ 130 ordinateurs, stations et petits serveurs, la plupart sous Unix, une centaine de MacIntosh et quelques compatibles PC sur un réseau local multi-sites.

Les MacIntosh sont regroupés sur plusieurs réseaux AppleTalk. Sur chacun des ces réseaux, les MacIntosh dialoguent entre eux grâce aux protocoles spécifiques d' Apple. Chaque réseau AppleTalk est relié à Ethernet par une passerelle spécifique (cela permet une situation de travail sur un VAX à partir d'un MacIntosh).

L'ensemble des laboratoires est réparti sur deux sites principaux : l'un au centre ville à Grenoble, l'autre sur le domaine universitaire à Saint Martin-d'Hères. Sur chacun de ces deux sites est aménagé un réseau local Ethernet (plusieurs câbles coaxiaux Ethernet d'un débit de dix mégabits par seconde, reliés entre eux par des répéteurs). Ces deux sous réseaux sont connectés par une liaison spécialisée, louée aux PTT et gérée par le protocole X25.

Les échanges intra-réseaux et inter-réseaux utilisent TCP ou UDP au niveau de la couche transport représentée dans la quatrième couche du modèle ISO (cf. chapitre 3, paragraphe 2.1). Au niveau de la couche application, les services offerts aux utilisateurs, les plus généralement employés, sont le transfert de fichiers (FTP : File Transfert Protocol), l'accès à distance (TELNET : TErminaL NETwork protocol) et le courrier électronique (SMTP : Simple Mail Transfert Protocol). On trouve également les extensions des commandes standard d'Unix appelées les r-commandes (exemples: rcp: *remote copy*, rlogin: *remote login*, rsh: *remote shell*).

## 2.2. UNIX

Les différents systèmes Unix de l' I.M.A.G. appartiennent tous à l'un des deux standards: BSD ou System V. Nous avons, pour notre part, effectué des développements sur l'Unix System V du BULL SPS9 et sur l'Unix BSD 4.3 du VAX 11/785.

Les points marquants d'Unix sont son arborescence de désignation d'objets et ses primitives de communication entre processus, outre bien sûr, sa portabilité aisée qui en fait un standard disponible chez quasiment tous les constructeurs (parfois marginalement).

Un nom complet de fichier s'écrit comme une séquence de noms séparés par le caractère '/' correspondant à l'arborescence des répertoires et sous-répertoires.

**exemple:** /users/imag/voboril/fich1

Le nom commence par le caractère '/' appelé la racine qui contient le répertoire *users*. Celui-ci contient le répertoire *imag* qui contient à son tour le répertoire *voboril*. C'est dans cette structure de noms hiérarchique que se loge le fichier *fich1*. Les répertoires sont utilisés pour grouper ensemble les fichiers relatifs à un même projet. Au départ, chaque utilisateur a un répertoire, créé par l'administrateur du système, appelé le "*home directory* ".

Un interpréteur de commandes puissant permet aux utilisateurs individuels ou aux projets de personnaliser l'environnement pour adapter leur propre style en définissant leurs propres commandes. Un système de documentation permet l'accès aux commandes disponibles définies dans le manuel Unix.

Les outils d'Unix BSD pour la communication entre processus (locaux et distants, via le réseau) sont basés sur la notion de "sockets " (détaillée au chapitre 3, paragraphe 2.3). Unix System V offre des outils différents, mais beaucoup de constructeurs leur ont ajouté les "sockets". Nous avons donc pu utiliser ce mécanisme unique sur SPS9 comme sur le VAX. Toutefois, l'existence d'un mécanisme universel ne résout pas tous les problèmes. En particulier, il ne résout pas l'incompatibilité de la représentation y compris au niveau des matériels (cf. chapitre 3, paragraphe 2.4).

Unix intègre toute une panoplie d'outils forts appréciés des utilisateurs. Ceux-ci procurent un environnement complet de mise au point de logiciels, du type "boîte à outils" plutôt qu'un environnement intégré.

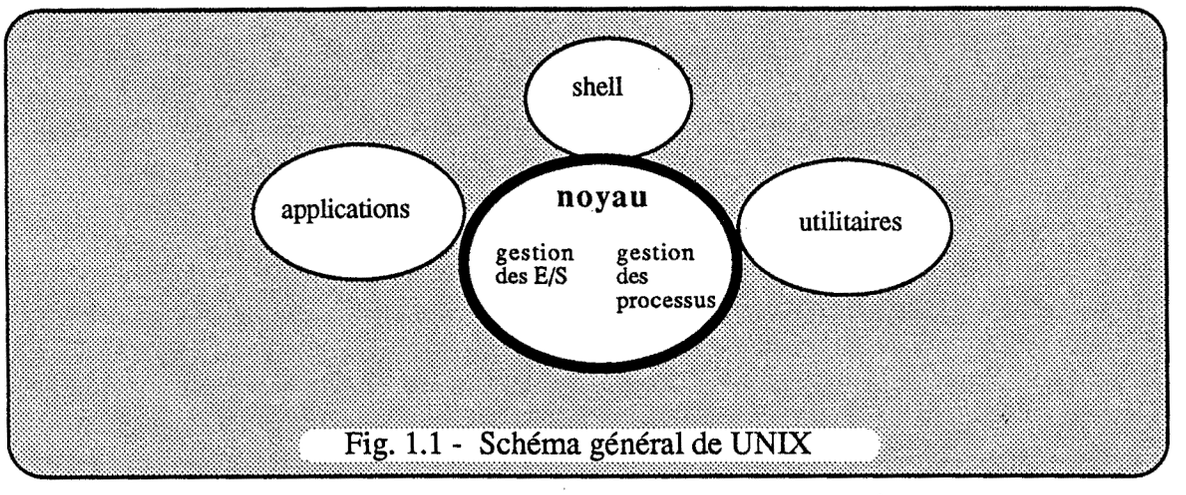



Fig. 1.1 - Schéma général de UNIX

Les principaux outils utilisés au cours de la réalisation du projet sont:

- **learn** : cours auto éducatifs sur plusieurs sujets (éditeur vi, langage C, ...).
- **shell** : interpréteur du langage de commandes (écriture de fichiers *"scripts"*),
- **cc** : compilateur langage C, éditeur de liens,
- **vi** : éditeur pleine page,
- **make** : outil d'aide au développement et à la maintenance des programmes (il mémorise les dépendances entre les fichiers et les séquences d'instructions permettant de synchroniser leur mise à jour),
- **dbx** : débogueur symbolique d'Unix BSD, il permet de déboguer des programmes sources et l'exécution de programmes sous Unix en mode pas à pas ou avec arrêts en des points particuliers.
- **awk** : outil d'analyse et de manipulation de chaînes de caractères dans des fichiers,

- **fonctions système,** notamment pour l'utilisation de sockets de communication entre processus,
- **outils réseaux,** notamment pour l'appel de procédures à distance (RPC) en environnement hétérogène (XDR).

La programmation a été réalisée en langage C.

# CHAPITRE 2

# GESTION DES PROJETS DE RECHERCHE A TRAVERS UNE BASE DE DONNEES RELATIONNELLE

# 1. Introduction

Dans un premier paragraphe, nous présentons l'organisation interne des laboratoires de l'IMAG afin d'introduire les relations de la base de données détaillée dans le chapitre "Développement de l'application".

Dans un second paragraphe, nous parlons du Système de Gestion de Base de Données Oracle et de ses outils utilisés pour la réalisation de notre application, la partie technique de ce SGBD étant annexée.

## 1.1. Organisation des laboratoires

L'existence de plusieurs personnes amenées à travailler sur un même projet a préconisé la formation d'équipes au sein de laboratoires spécifiques. Chaque laboratoire travaille sur des sujets particuliers appelés thèmes. Ce sont des critères de classement généraux (ex: *la robotique).*

Des critères plus précis interviennent lorsque l'on descend au sein des équipes et des projets; ce sont les sous-thèmes. Les sous-thèmes de chaque équipe d'un laboratoire forment l'ensemble des sous-thèmes de ce laboratoire.

Toute personne de l'IMAG a une fonction, un statut bien déterminé tel que, enseignant, chercheur ou thésard. Elle appartient à un organisme particulier comme le CNRS, l'INPG ... Elle peut avoir plusieurs adresses professionnelles si elle a plusieurs bureaux en des lieux géographiques différents.

Toutes ces informations sont rassemblées dans la base de données que nous étudions dans le paragraphe 2.

## 1.2. Le Système de Gestion de Base de Données Oracle

Oracle est basé sur le modèle relationnel.

Le modèle relationnel est apparu en 1970 avec les propositions de E.F. CODD pour un système de bases de données dont le modèle est constitué de relations. Ce modèle de données consiste à percevoir l'ensemble des données comme des tableaux organisés en lignes et colonnes [CHRI87].

Oracle est un système de gestion de bases de données relationnelles développé par la Société Oracle Corporation sur un grand nombre de systèmes, des ordinateurs personnels aux gros systèmes.

Le langage d'accès aux bases de données est SQL (Structured Query Language). C'est aujourd'hui le langage standard commun à la plupart des bases de données relationnelles. Il se découpe en deux parties:

1 - **L.D.D.** (Langage de Définition de Données) que nous avons utilisé pour décrire le schéma de la base de données,

2 - **L.M.D.** (Langage de Manipulation de Données) utilisé pour intégrer, modifier ou extraire des informations de la base de données. Il peut s'exécuter:
   - à travers un langage hôte (cobol,Fortran, PL/1, Assembleur, C dans notre cas)
   - de façon autonome.

Dans un programme, une requête SQL est réalisée sous la forme d'un appel de primitives Oracle. Une phase de ***précompilation*** traduit ces appels en appel de procédures externes.

L'annexe 1 détaille l'organisation d'une base de données Oracle. Elle donne également la liste des outils proposés par Oracle pour gérer une base de données.

# 2. Développement de l'application

Ce chapitre aborde la méthodologie de mise en œuvre en commençant par une étude des besoins des utilisateurs et des données existantes. Nous présentons ensuite le schéma conceptuel des données suivant le modèle Z d'Abrial [ABRI74]. La mise en place d'un environnement relationnel ne va pas sans spécifier une codification aussi bien pour les objets relationnels que pour les fichiers nécessaires à l'environnement d'Oracle. Cette étape nous a permis de définir les relations de la base.

Les données indispensables à la création physique de la base doivent être regroupées. Nous exposons le moyen employé pour rassembler ces données, la préparation plus ou moins automatique des fichiers de chargement de la base puis le chargement lui–même par l'outil ODL® .

Un dernier paragraphe détaille en partie, les difficultés rencontrées lors de la conception et du développement de cette application.

## 2.1. Méthodologie

Nous présentons ici, la démarche suivie lors de l'étude de l'application concernant la base de données IMAG, une codification jugée pratiquement indispensable pour un environnement Oracle sous Unix et la définition des relations de la base.

### 2.1.1. Expression des besoins et analyse des données

On considère dans un premier temps, les besoins exacts de l'utilisateur afin de fixer les buts de l'existence de l'application.

---

® ODL : Oracle Data Loader

On aborde ensuite les points relatifs à l'existant et la façon de stocker les informations dans la base relationnelle en fonction de certaines règles techniques.

**• Expressions des besoins**

Que veut l'utilisateur final ?

- il veut imprimer le rapport d'activité d'une équipe, ou des équipes travaillant sur un thème donné par exemple,

- il veut des informations sur un projet de recherche, les personnes relatives à ce projet, le nom et l'adresse du ou des laboratoires, etc..., sur papier ou à l'écran et suivant un certain format,

- il veut pouvoir modifier le rapport d'activité à l'écran d'une année sur l'autre.

Sont donc à prévoir, des imprimés et des images écrans comme résultats essentiels pour les utilisateurs.

Bien évidemment, ces données vont vivre, évoluer ou changer. Il faut donc prévoir des traitements en conséquence : mise à jour, création, suppression et le traitement principal : la consultation. Ces traitements seront guidés par des menus écran.

**• Etude des données existantes**

Les projets sont répartis dans une ou plusieurs équipes et dépendent d'un ou plusieurs thèmes et sous-thèmes. Les thèmes et sous-thèmes sont des rubriques qui permettent de classer les projets, les équipes, les laboratoires et par conséquent les personnes. Un rapport d'activité par équipe est diffusé chaque année.

Quelles sont les informations utiles pour la construction du modèle relationnel de la base ? Elles sont toutes indispensables puisque les données stockées dans la base doivent permettre de construire ces rapports d'activité en plus de la consultation offerte aux usagers.

Grâce à un important travail de synthèse, les thèmes et sous-thèmes existants dans ces rapports d'activité ont été regroupés sous la forme d'une liste qui, dans un premier temps, fut loin d'être exhaustive et définitive. En effet, certains thèmes se recoupaient, quelques sous-thèmes présentaient des redondances. Certaines incohérences ont fait surface.

L'élaboration d'une liste standard a mis fin à tous ces problèmes puisque chacun peut choisir au moins une rubrique spécifique à son activité.

L'organisation actuelle, en équipes et laboratoires, entraîne la création de ces deux entités dans la base. Un nom de laboratoire doit être unique, tout comme celui d'une équipe. La façon de représenter cette règle est de créer une clé primaire unique sur l'attribut qui représente le nom. Une des obligations se rapportant aux clés primaires indique que celles-ci ne peuvent jamais être mises à jour [CHRI81]. Cela veut dire que si un nom doit être modifié, il faut supprimer la ligne et la recréer ! Heureusement, il existe une solution plus simple; l'adjonction d'un attribut comme clé primaire.

On choisit un attribut numérique. Or, il se trouve qu'il existe déjà une numérotation de laboratoires et d'équipes au sein d'un logiciel développé pour la gestion du service reprographie de l' I.M.A.G. [REY84]. Nous avons essayé de nous rapprocher au maximum de cette numérotation afin de ne pas affecter deux numéros différents à une même équipe mais aussi surtout pour ne pas créer une autre représentation numérique pour des mêmes attributs.

Les entités "equipe" et "laboratoire" viennent d'être citées, mais il en est de même pour quelques autres. Notamment, l'entité "personne" doit être représentée de façon unique. Un attribut numérique sera également ajouté. Une remarque particulière découle de cet attribut; ce numéro obligatoirement unique représente une et une seule personne. Les personnes responsables du réseau veulent profiter de cette unicité pour modifier le numéro identificateur d'une personne sur toutes les machines du réseau.

En effet, actuellement, une personne peut avoir un compte sur le VAX et être représentée par un numéro n1 et avoir un compte sur le SPS9 et être représentée par un numéro n2. Ce procédé s'avère assez peu fiable et risque de provoquer très vite une saturation au niveau des chiffres.

Une seule contrainte nous est imposée: commencer la numérotation à 100. Par définition, chaque personne sera représentée dans l'entité "personne" de la base de données par un numéro à trois positions. La codification interne des personnes dans la base de données est donc une occasion à saisir pour uniformiser les identifications inter-machines.

### 2.1.2. Le schéma conceptuel

Le modèle conceptuel n'est qu'une représentation abstraite d'un ensemble de données et de liens entre des données. Il permet toutefois une analyse très complète de la situation et des relations à mettre en place.

Nous présentons les relations de la base d'après le modèle Z dans les lignes qui suivent.

laboratoire ---------- composition (1,n) / labod'équipe (1,1) ---------- équipe

laboratoire ---------- composition (1,n) / labodepersonne (1,1) ---------- personne

thème ---------- sous-thèmedethème (0,n) / thèmedesous-thème (1,n) ---------- sous-thème

équipe ---------- projetd'équipe (0,n) / équipedeprojet (1,n) ---------- projet

| | | |
|---|---|---|
| équipe | sous-thèmed'équipe (1,n)<br>équipedesous-thème (1,n) | sous–thème |
| personne | équipedepersonne (0,1)<br>personned'équipe (1,n) | équipe |
| personne | responsabled'équipe (0,1) | équipe |
| personne | responsabledelabo(0,1) | laboratoire |
| laboratoire | tutelledelaboratoire (1,n)<br>laboratoiredetutelle (1,n) | tutelle |
| laboratoire | adressedelaboratoire (1,1) | adresse |
| personne | thèmedepersonne (0,n)<br>personnedehème (0,n) | thème |
| personne | projetdepersonne (0,n)<br>personnedeprojet (1,n) | projet |
| personne | responsable (0,n) | projet |
| projet | sous-thèmedeprojet (1,n)<br>projetdesous-thème (0,n) | sous-thème |

projet ---------- ostdeprojet (1,n) ---------- ost

projet ---------- soutiendeprojet (0;n) ---------- soutien

personne ---------- adressedepersonne (1,n) ---------- adresse

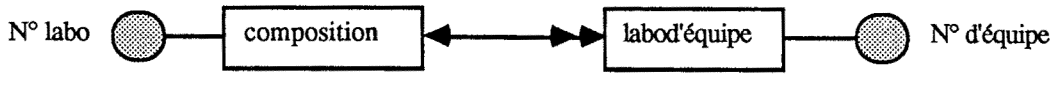
personne ---------- statutdepersonne (1,n) ---------- statut

L'élaboration du schéma a entraîné une multitude de questions comme: "combien de personnes peuvent être responsables d'un même projet ?" ou encore "dans un laboratoire, peut-il y avoir des personnes qui ne font pas partie d'une équipe ?" ...

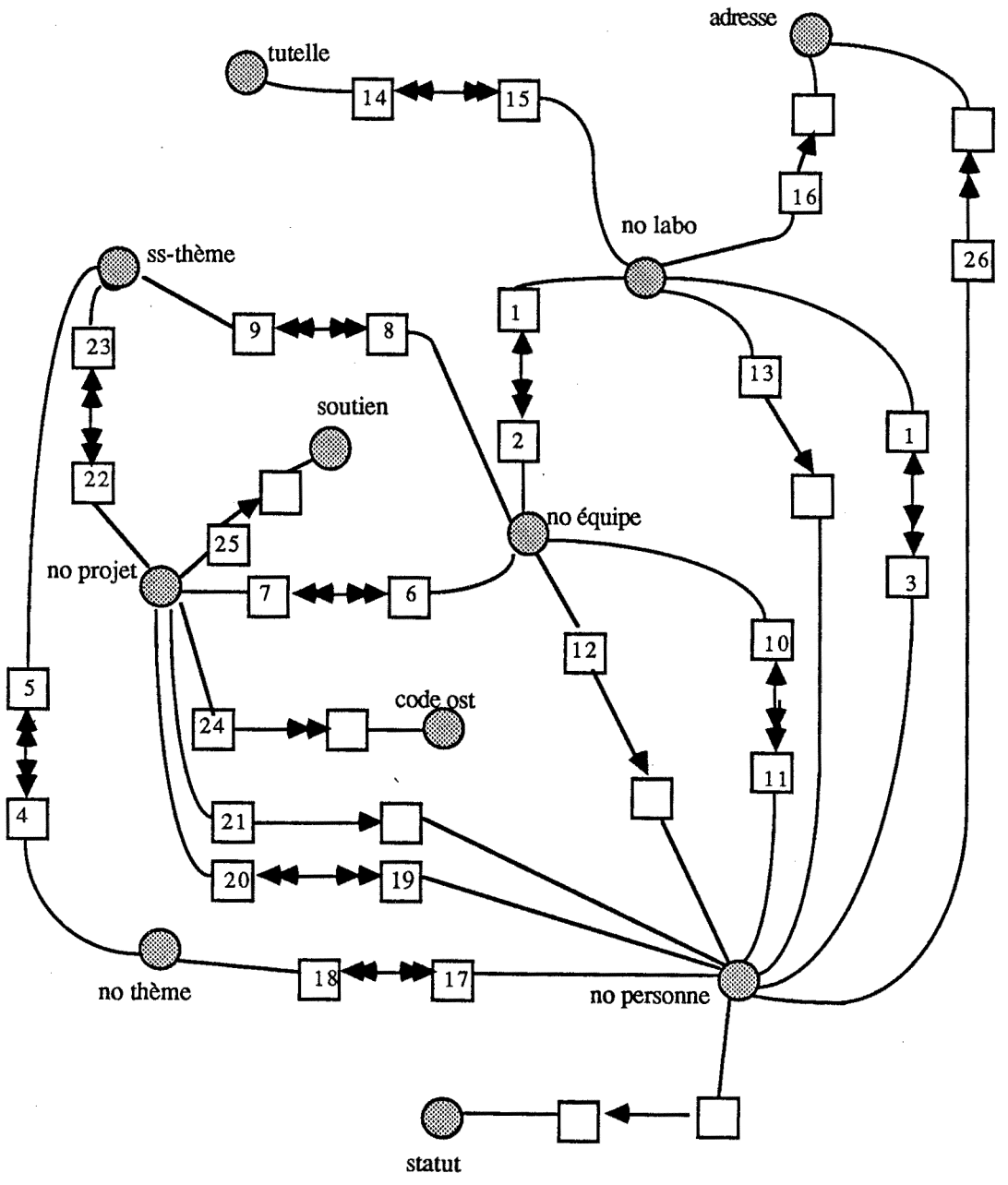
Les réponses sont toutes visibles à travers le schéma relationnel où chaque flèche du schéma permet d'écrire une réponse.

Une association binaire dans le schéma conceptuel est représentée sous la forme d'un dessin conventionnel dans lequel les ronds représentent les noms des ensembles d'entités et les carrés les noms des fonctions, une flèche simple indiquant une fonction monovaluée, alors qu'une flèche double, une fonction multivaluée.

**exemple:**




La fonction "composition" donne pour un numéro de laboratoire, l'ensemble des numéros d'équipe qui le compose, tandis que la fonction "labod'équipe" donne pour un numéro d'équipe le numéro du laboratoire. C'est ainsi que nous présentons le schéma conceptuel à l'aide de la figure 2.1.




1 - composition
2 - labod'équipe
3 - labodepersonne
4 - sous-thèmedethème
5 - thèmedesous-thème
6 - projetd'équipe
7 - équipedeprojet
8 - sous-thèmed'équipe
9 - équipedesous-thème
10 - équipedepersonne
11 - personned'équipe
12 - responsabled'équipe
13 - responsabledelabo
14 - labodetutelle
15 - tutelledelabo
16 - adressedelabo
17 - thèmedepersonne
18 - personnedethème
19 - projetdepersonne
20 - personnedeprojet
21 - responsabledeprojet
22 - sousthèmedeprojet
23 - projetdesous-thème
24 - ostdeprojet
25 - soutiendeprojet
26 - adressedepersonne
27 - statutdepersonne

Fig. 2.1 - Schéma conceptuel des données

Nous avons appliqué la décomposition des relations en respectant les formes normales. La normalisation (suivant les formes normales de CODD) entraîne une décomposition des relations [DELO83].

Rappelons en quoi consistent les formes normales à travers des exemples se rapportant à une partie de notre base.

**Normalisation** : décomposition des relations

**Première forme normale** : aucun attribut n'est un ensemble d'ensembles.
exemple: Relation PROJET

R0

| NOMPROJ | PERS | SOUTIEN |
|---|---|---|
| TIGRE | {p1,p2,p3} | {s1,s2} |
| ESPRIT | {p4} | {s1,s3,s4} |
| SOLEIL | {p1} | {ø} |

Fig. 2.2 - Relation non normalisée.

PERS et SOUTIEN sont des ensembles d'ensembles. La transformation en première forme normale est une opération simple comme le montre la figure ci-dessous:

R1

| NOMPROJ | PERS | SOUTIEN |
|---|---|---|
| TIGRE | p1 | s1 |
| TIGRE | p1 | s2 |
| TIGRE | p2 | s1 |
| TIGRE | p2 | s2 |
| TIGRE | p3 | s1 |
| TIGRE | p3 | s2 |
| ESPRIT | p4 | s1 |
| ESPRIT | p4 | s3 |
| ESPRIT | p4 | s4 |
| SOLEIL | p1 | - |

Fig. 2.3 - Représentation normalisée de la relation R0.

**Deuxième forme normale:** dépendances fonctionnelles

- la relation doit être en première forme normale,

- tout attribut n'appartenant pas à la clé est en dépendance fonctionnelle complète avec la clé.

**Troisième forme normale:** dépendances fonctionnelles transitives

- la relation doit être en deuxième forme normale,

- tout attribut n'appartenant pas à la clé n'est pas en dépendance fonctionnelle transitive avec la clé.

R

| IDPROJ | IDPERS | NOM |
|---|---|---|
| 0001 | 100 | BAUMANN |
| 0002 | 100 | BAUMANN |
| 0003 | 101 | SCOTT |
| 0004 | 100 | BAUMANN |
| 0005 | 100 | BAUMANN |

Fig. 2.4 - Relation en 2ème FN et non en 3ème FN

On a les dépendances : IDPROJ => IDPERS
IDPERS => NOM

Une meilleure représentation se substitue en deux relations :

R1

| IDPROJ | IDPERS |
|---|---|
| 0001 | 100 |
| 0002 | 100 |
| 0003 | 101 |
| 0004 | 100 |
| 0005 | 100 |

R2

| IDPERS | NOM |
|---|---|
| 100 | BAUMANN |
| 101 | SCOTT |

Fig. 2.5 - Passage 2ème FN en 3ème FN par décomposition

Inconvénients de la représentation en deuxième forme normale par rapport à la troisième forme normale:

. AJOUT : Supposons qu'une nouvelle personne (RICHARD) arrive sans avoir encore un projet. Le tuple { -, 102, RICHARD } ne peut s'ajouter à R car la clé n'est pas définie, mais le tuple { 102, RICHARD } peut s'ajouter à R2.

. SUPPRESSION : Si le projet 0003 se termine, sa suppression dans R entraîne la perte de l'existence de SCOTT, alors que ce n'est pas vrai dans R1 et R2.

. MISE-A-JOUR : Si l'on veut mettre à jour le nom d'une personne, une seule ligne suffit avec R1 et R2, mais il faut plusieurs mises à jour au niveau de R.

Ces exemples démontrent la nécessité de la normalisation pour éviter des incohérences lors de la maintenance d'une base de données relationnelle.

Chaque association, chaque cardinalité a été mise en clair, en français, comme par exemple *"une personne appartient toujours à un et un seul laboratoire"*. Nous avons regroupé ces contraintes dites ***contraintes d'intégrité*** pour en établir une liste.

**Liste des contraintes d'intégrité**

Une personne appartient toujours à un et à un seul laboratoire
Un laboratoire est composé d'une ou plusieurs personnes
Un laboratoire est composé d'une ou plusieurs équipes
Une équipe appartient à un et à un seul laboratoire
Une équipe comprend une ou plusieurs personnes
Une personne fait partie de 0 ou 1 seule équipe
Dans une équipe, il y aura de 0 à plusieurs projets
Un projet est conduit par une ou plusieurs équipes
Un projet est conduit par 1 et 1 seul responsable
Un projet est mené par une ou plusieurs personnes
Une personne travaille sur 0 à plusieurs projets

Un thème est subdivisé en 0 ou plusieurs sous-thèmes
Un sous-thème appartient à un ou plusieurs thèmes
Un sous-thème peut regrouper 0 ou plusieurs projets
Un projet peut être en relation avec un ou plusieurs sous-thèmes
Dans un laboratoire donné, sont conduits 0 à plusieurs projets
Un projet est conduit par un ou plusieurs laboratoires
Une équipe travaille sur un ou plusieurs sous-thèmes
Un sous-thème est étudié par une ou plusieurs équipes
Une personne s'intéresse à 0 ou plusieurs thèmes
Un laboratoire travaille sur un à plusieurs thèmes
Un thème intéresse 0 à plusieurs laboratoires
Un thème peut intéresser 0 à plusieurs équipes
Un thème peut regrouper un ou plusieurs projets
Un projet peut être en relation avec un ou plusieurs thèmes
Une équipe travaille sur un ou plusieurs thèmes
Un sous-thème peut intéresser 0 ou plusieurs personnes
Une personne s'intéresse à 0 ou plusieurs sous-thèmes

Cette liste a été envoyée à chaque équipe ainsi qu'une liste des thèmes et des sous-thèmes afin de les faire valider ou éventuellement modifier. Les critiques et réflexions qui en sont revenues ont permis de construire le shéma présenté en figure 2.1.

Pour mettre en œuvre le modèle conceptuel des données, nous avons ressenti le besoin d'un ensemble de normes de ***codification***. Cette codification pourra être appliquée lors de développement ultérieur dans le cas d'un environnement Oracle sous Unix.

### 2.1.3. Codification retenue pour l'environnement Oracle sous Unix

Les expériences vécues dans les environnements de SGBD relationnels (notamment à Merlin Gerin) montrent qu'une étude préalable des objets relationnels et des fichiers systèmes s'avèrent indispensables. Ils sont la source d'une bonne analyse et d'un gain de temps considérable pour la mise en œuvre de l'application. Il est clair que cette démarche est une étape initiale au développement d'applications sous un SGBD relationnel.

C'est le rôle que doivent tenir les personnes d'une cellule "système-développement". Il faut également mettre en place des moyens ou des outils pour faire respecter ces normes.

Cette étude nous a donc conduit à instaurer une codification des objets Oracle et des fichiers Unix indispensables à l'environnement d'une base de données Oracle. Aucune application n'ayant été développée avec Oracle, nous sommes partis de zéro pour mettre au point la codification.

Il a été choisi de structurer la plupart des noms en fonction d'une seule racine: **le nom de la table.** Nous avons conservé une certaine homogénéité avec Oracle pour les suffixes des fichiers et les préfixes des objets.

Nous présentons ici le détail de la codification choisie pour les objets Oracle:

**DATABASE** : la seule contrainte du nom de cet objet sera sa taille d'un maximum de 8 caractères. Ce nom doit être unique.

**PARTITION** : p_xxxxxxn où
- p signifie "partition ",
- xxxxxx indique le nom caractéristique de la partition.
- n précise un numéro d'ordre, le numéro de la partition dans la base.

Il est préférable, pour une meilleure homogénéité, de retrouver ce nom dans une partie de celui de la base.

**SPACE DEFINITION :** sp_nomtable où

- sp signifie "space definition ",
- nomtable représente le nom de la table pour laquelle le *space definition* est créé. Une seule table par *space definition* : cette règle est choisie pour simplifier la gestion de l'environnement mais aussi pour des question de performances.

exemple: sp_equipe

**CLUSTER :** cl_nomcluster où

- cl signifie cluster.

Un cluster peut contenir une ou plusieurs tables. Il sert à améliorer les performances d'accès sur les tables.

**TABLE :** nomtable

Il doit commencer par une lettre et être mnémonique. Sa taille est d'un maximum de 7 caractères. Il sert de racine aux noms des fichiers Unix qui seront suffixés; la taille maximum du nom de la table est donc dépendante de la taille maximum des fichiers Unix, soit 14 caractères. Une table appartient à son créateur (c'est à dire, l'utilisateur Oracle qui l'aura créée). Le nom de la table est alors implicitement qualifié par le nom de son créateur.

exemple: scott.equipe

**INDEX :** nomtable i n où

- nomtable représente le nom de la table sur laquelle est créé l'index ,
- i signifie "index",
- n indique un numéro d'ordre (vaut 1 si un seul index est créé sur la table).

exemple: equipei1

**VUE MERE :** nomtable v où

- nomtable est le nom de la table initiale qui sert à créer la vue ,
- v signifie "vue"
- taille: 8 caractères maximum. Le programme d'application ne doit travailler qu'au travers de vue pour se parer de toute modifications de la table.

**COLONNE :** nomcolonne

Le nombre maximum de caractères possibles pour nommer la colonne est de 30.

Il est préférable de le rendre le plus court possible pour une manipulation plus aisée en programmation. Une entité donnée doit avoir le même nom dans toutes les tables ainsi que le même format quelque soit le projet. A l'inverse, il ne faut pas utiliser deux noms identiques pour des significations différentes.

Le chargement des données dans les tables Oracle peut se faire ligne à ligne avec l'ordre SQL "INSERT", mais ceci est long et fastidieux à préparer surtout dans le cas de l'initialisation des tables d'une base. Il existe une façon plus pratique de charger une table : en utilisant ODL®, qui, à partir d'un fichier de données formaté, va créer les segments données et index, s'il y a lieu, dans la table chargée.

Pour exécuter ce travail, il faut un fichier de contrôle indiquant à ODL les définitions des champs en entrée et en sortie. On ne peut charger qu'une seule table par fichier de contrôle. Nous avons donc créé un fichier de contrôle par table à charger avec ODL. Il s'intègre dans la syntaxe d'ODL telle qu'elle est définie ici:

**ODL fichier_contrôle fichier_log user/password**

® ODL: Oracle Data Loader

Une codification rigoureuse évite de se perdre dans des recherches de noms inutiles. Ainsi, nous avons adopté les conventions définies dans les lignes suivantes pour les noms des fichiers Unix.

**FICHIER DE CONTROLE :** nomtable.odl

Ce fichier contient les ordres de chargement d'une table. Il précise le nom du fichier qui contient les données à charger dans la table, les définitions du format d'entrée et de sortie (table Oracle). Il doit se trouver dans le sous-répertoire appelé ODL.

exemple: equipe.odl

**FICHIER LOG :** nomtable.log

La syntaxe de la commande ODL indique que la présence de ce fichier est obligatoire. Il est créé lors de chaque commande ODL pour donner le résultat du chargement de la table. On y trouve notamment, le nombre d'enregistrements lus en entrée et le nombre d'enregistrements chargés.

exemple: equipe.log

Le chargement consécutif de plusieurs tables risque d'entraîner une "pollution" du répertoire dans lequel la commande ODL est effectuée. Nous avons prévu un petit fichier de commandes Unix pour faire le ménage de ces fichiers résultats.

**FICHIER DE DONNEES :** fnomtable

Ce fichier contient les données à charger dans la table *nomtable* par l'intermédiaire d'ODL. Son nom est spécifié dans le fichier de contrôle associé. Tous les fichiers de données sont rassemblés dans le répertoire appelé FICHBD.

exemple: fequipe

L'environnement ODL ne suffit pas à la gestion de celui d'Oracle. En effet, les tables sont créées à l'aide du langage SQL et chargées via ODL. La codification mise en place consiste à construire un fichier par table pour la création de tout ce qui la concerne, à savoir son "space definition", la table elle même (sans les données), ses index, ses commentaires

de colonnes et les autorisations d'accès [ORA3]. Nous avons choisi pour ce fichier, une structure bien définie détaillée ci-dessous.

- suppression des index
- suppression de la table
- suppression du space definition

- création du space definition
- création de la table dans son space definition
- affectation de commentaires sur les colonnes de la table
- création d'index

- donner l'autorisation de lecture sur la table, à tout le monde

**- structure d'un fichier de création pour une relation -**

Un exemple de ce type de structure est donné en Annexe 2.

Les lignes de ce fichier sont exécutées sous UFI (si version 4 d'Oracle) ou sous SQL*PLUS (en version 5). Le propriétaire de la table est alors l'utilisateur connecté à Oracle qui exécute le fichier de création par la commande START [ORA1].

Sa syntaxe est décrite ci-après:

**FICHIER DE CREATION:** crt nomtable.sql

Ce type de fichier joue un rôle important dans l'environnement de la base de données. Il est exécuté sous SQL*PLUS avec la commande START. Le propriétaire de la table est alors l'utilisateur connecté à Oracle qui lance le fichier. Il existe un fichier de ce type par table. Tous ces fichiers sont regroupés au sein d'un même répertoire: ENVIR.

exemple : **crtequipe.sql**

**START crtequipe** (le suffixe 'sql' est ajouté automatiquement).

D'autres fichiers ont un suffixe "sql". Ils peuvent contenir des requêtes SQL particulières telles que donner l'autorisation à un utilisateur de faire quelque chose sur une table ou sur une partie de table. Ce dernier type de fichier peut faire partie de la famille des outils maison

construits à base de langage SQL. Il est fortement conseillé de garder des traces de ce qui a été fait en ajoutant, en commentaire, la date à laquelle l'opération a été effectuée. Chaque ligne exécutée sera mise en commentaire avant l'exécution d'une autre requête de même ordre (cf. exemple pour FICHIER OUTILS).
Ces deux types de fichiers seront stockés dans le répertoire QUERY. Voici leur description:

**FICHIER REQUETES :** xxxxx **q** nn.**sql** où

- xxxxx est un nom mnémonique se rapportant à la nature de l'opération faite sur la ou les tables spécifiées dans la requête (maximum 7 caractères).
- q signifie "query",
- nn correspond au numéro d'ordre général pour un même préfixe xxxxxq.

exemple: majpersq01.sql

**FICHIER OUTILS :** xxx_yyy.**sql** où

- xxx représente trois lettres significatives indiquant le type de l'outil,
- yyy interprète un maximum de dix caractères alphanumériques indiquant de façon plus détaillée le rôle de l'outil.

exemple: grt_imagrech.sql où 'grt' signifie 'grant';
Cet ordre a pour but de donner les autorisations sur les tables de la base nommée "imagrech".
exemple de contenu avec mise en commentaire des ordres déjà exécutés :

```
≠ autorisation donnée le 8 Août 1988
≠ grant select on table pers to public;
≠ autorisations données le 9 Février 1989
grant update on table labo to baumann;
grant update (col1, col2) on table proj to user1;
```

Chaque utilisateur d'Oracle possède un répertoire (son *"home directory"*) au sein d'un répertoire commun. Sous chacun de ces répertoires utilisateurs, on trouvera des répertoires spécifiques à chaque ensemble de fichiers:

**ENVIR :** contient les fichiers de création des tables codés:
**crt** nomtable **. sql**

**QUERY :** contient les requêtes SQL et les outils
**xxxxxqnn.sql** et **xxx_yyy.sql**

**SRC :** contient les programmes sources.

**BIN :** contient les programmes exécutables et les fichiers de commandes Unix.

**ODL :** contient les fichiers de contrôle destinés au chargement des tables
nomtable**.odl**

**FICHBD :** contient tous les fichiers de données initiales en vue de charger les tables de la base.
**bd**xxx, **fich**nomtable, **f**nomtable

Comme dans beaucoup d'applications, des évolutions peuvent se produire, entraînant alors des modifications dans les programmes exécutés par les utilisateurs finaux. Nous qualifierons ces programmes d'opérationnels. Afin de ne pas gêner les usagers de l'application, les modifications devront se faire dans un environnement de test.

Une fois le programme mis au point, le code exécutable est migré vers le répertoire de l'environnement opérationnel, soit le répertoire nommé ***usr/local.*** On entend par *"migration"*, le transfert vers la destination et la suppression à l'origine.

Un programme exécutable peut ainsi se trouver à deux endroits différents s'il est en cours d'évolution.

Le schéma suivant illustre l'organisation des répertoires de notre environnement utilisateur sur l'ordinateur SPS9 (CASTOR):

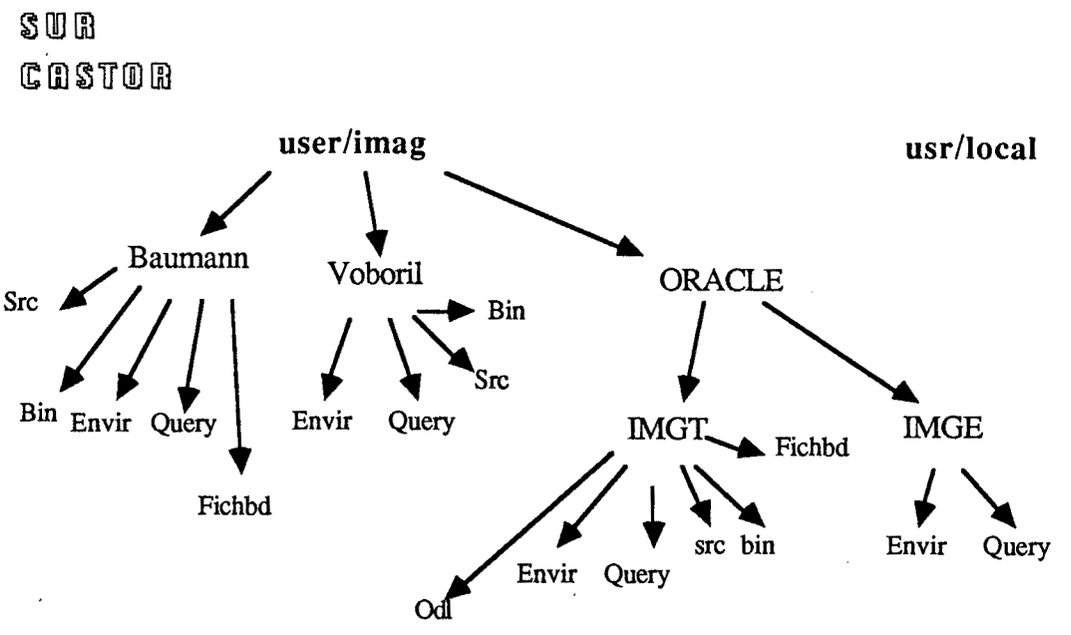



Fig. 2.6 - Organisation des répertoires sur CASTOR

On constate l'existence de deux répertoires de travail (Baumann et Voboril) et un répertoire contenant deux sous-répertoires. Le sous-répertoire IMGT correspond à l'environnement de test ("IMG" comme "IMAG" et "T" comme "test"), et le sous-répertoire IMGR à l'environnement d'exploitation ("E" comme exploitation).

Le répertoire IMGE ne contient aucun sous-répertoire prévu pour les programmes. Le source des programmes reste en environnement de test et le code exécutable est transféré de l'environnement de test vers le répertoire commun "usr/local".

### 2.1.4. Définitions des relations de la base

Le détail des relations , leurs attributs et leur taille sont donnés par les figures 2.7 et 2.8.

| NOM TABLE | NOM ATTRIBUT | TAILLE ATTRIBUT | COMMENTAIRE ATTRIBUT | LIBELLE TABLE | NOMBRE DE LIGNES |
|---|---|---|---|---|---|
| labo | idlabo | 2 | n° du laboratoire | table des laboratoires | 10 |
| | siglabo | 15 | son sigle | | |
| | lblabo | 15 | son libellé | | |
| | lieu | 15 | son lieu géographique | | |
| | telabo | 10 | son n° de téléphone | | |
| equipe | idlabo | 2 | n° du laboratoire | table des équipes | 100 |
| | idequip | 2 | n° de l'équipe | | |
| | lbequip | 80 | libellé de l'équipe | | |
| | idresp | 3 | n° du responsable de l'équipe | | |
| pers | idpers | 3 | n° de personne | table des personnes | 1000 |
| | nom | 30 | nom de la personne | | |
| | prenom | 20 | prénom de la personne | | |
| | statut | 1 | code statut (situation) | | |
| | appt | 5 | appartenance | | |
| | idlabo | 2 | n° labo de la personne | | |
| | idequip | 2 | n° d'équipe de la pers. | | |
| proj | idproj | 4 | n° de projet | table des projets | 300 |
| | sigproj | 15 | son sigle | | |
| | lbproj | 50 | son libellé | | |
| | datefinproj | 8 | date prévue fin projet | | |
| | idresp | 3 | son n° du responsable | | |
| | idprog | 3 | son n° de programme | | |
| | raproj | 65535 | rapport d'activité | | |
| them | idth | 4 | n° du thème | table des thèmes | 40 |
| | lbth | 60 | libellé du thème | | |
| ssthem | idssth | 4 | n° du sous-thème | table des sous-thèmes | 300 |
| | lbssth | 80 | libellé du sous-thème | | |

Fig.2.7 - Détail des relations de la base des Projets de l'IMAG (partie 1)

| NOM TABLE | NOM ATTRIBUT | TAILLE ATTRIBUT | COMMENTAIRE ATTRIBUT | LIBELLE TABLE | NOMBRE DE LIGNES |
|---|---|---|---|---|---|
| soutien | idsout<br>lbsout | 3<br>40 | n° du soutien<br>libellé du soutien | table des soutiens | 15 |
| statut | statut<br>lbstatut | 1<br>30 | code statut<br>libellé du statut | table des statut | ? |
| prog | idprog<br>lbprog | 3<br>50 | n° du programme<br>libellé du programme | table des programmes | ? |
| appten | appt<br>lbappt | 5<br>60 | code appartenance<br>libellé appartenance | table des appartenances | ? |
| adresse | lieu<br>adr1<br>adr2<br>adr3 | 15<br>15<br>25<br>25 | lieu géographique<br>adresse partie 1<br>adresse partie 2<br>adresse partie 3 | table des adresses | 5 |
| eqproj | idlabo<br>idequip<br>idproj | 2<br>2<br>4 | n° de laboratoire<br>n° d'équipe<br>n° de projet | table des projets pour chaque équipe | 500 |
| persth | idpers<br>idth<br>idssth | 3<br>4<br>4 | n° de personne<br>n° de thème<br>n° de sous-thème | table des thèmes et sous-thèmes par personne | 4000 |
| thssth | idth<br>idssth | 4<br>4 | n° de thème<br>n° de sous-thème | table des thèmes et de leurs sous-thèmes | ? |
| perspro | idpers<br>idproj | 3<br>4 | n° de personne<br>n° de projet | table des projets pour chaque pers. | 4700 |
| eqssth | idlabo<br>idequip<br>idssth<br>idth | 2<br>2<br>4<br>4 | n° de laboratoire<br>n° d'équipe<br>n° de sous-thème<br>n° de thème | table des thèmes et sous-thèmes par équipe | 3200 |
| prossth | idproj<br>idssth<br>idth | 4<br>4<br>4 | n° de projet<br>n° de sous-thème<br>n° de thème | table des thèmes et sous-thèmes par projet | 1800 |
| tutelle | idlabo<br>appt | 2<br>5 | n° de laboratoire<br>code appartenance | table des tutelles par labo | 30 |
| projsou | idproj<br>idsout | 4<br>3 | n° de projet<br>n° de soutien | table des soutiens par projet | 1800 |
| persadr | idpers<br>lieu<br>batimt<br>bureau<br>telpers | 3<br>15<br>10<br>10<br>8 | n° de personne<br>lieu géographique<br>batiment de la personne<br>son bureau<br>son n° de téléphone | table des adresses pour chaque personne | 1500 |
| projost | idproj<br>idost | 4<br>3 | n° de projet<br>code OST du projet | table des ost par projet | 1000 |

Fig. 2.8 - Détail des relations de la Base des Projets de l'IMAG (partie 2)

Cette démarche a débouché sur l'implantation d'une base prototype préalable, nécessaire pour aborder la réalisation des outils distants décrits dans le chapitre 3.

## 2.2. Création de la base et mise en œuvre de l'application

Nous présentons ici, les moyens mis en œuvre pour créer les données dans la base, à savoir, le regroupement des informations provenant de chaque équipe, le traitement de ces données soit automatiquement soit manuellement pour préparer la création de la base, l'outil de chargement ODL proposé par Oracle, et le chargement proprement dit.

### 2.2.1. Rassemblement des données

Comment regrouper toutes les informations nécessaires à la création de la base ? Les derniers rapports d'activité diffusés par les équipes de recherche procurent certaines informations, mais celles-ci ne sont plus forcément à jour. Le problème réside également dans le fait qu'elles sont sur papier, ce qui signifie qu'il faudrait les ressaisir ! La présentation des rapports d'activité n'est pas uniforme. Nous avons donc demandé aux chefs d'équipe de chaque laboratoire de remplir une sorte de formulaire. Sous la forme d'un fichier, il a été envoyé à chacun par messagerie électronique.

Le squelette du formulaire de saisie est décrit en Annexe 3. Ce document, une fois renseigné, nous a permis de travailler sur des données récentes.

Il y avait une solution plus conviviale et plus fiable de récupérer ces informations ; en écrivant une petite transaction. Un dialogue écran se serait engagé avec le chef d'équipe et les réponses vérifiées auraient été stockées directement dans un fichier formaté en vue d'être exploité pour le chargement de la base.

Ce procédé impliquait un système homogène pour chacun. Or, tout le monde ne travaille pas sur la même machine ni sur le même système d'exploitation. De plus, certains préfèrent saisir du texte sur un MacIntosh. Chacun des fichiers type (formulaires) a donc été

complété sur des matériels différents et renvoyés par messagerie électronique sur le serveur principal de l' IMAG.

La figure 2.9 illustre le cheminement de ces formulaires.

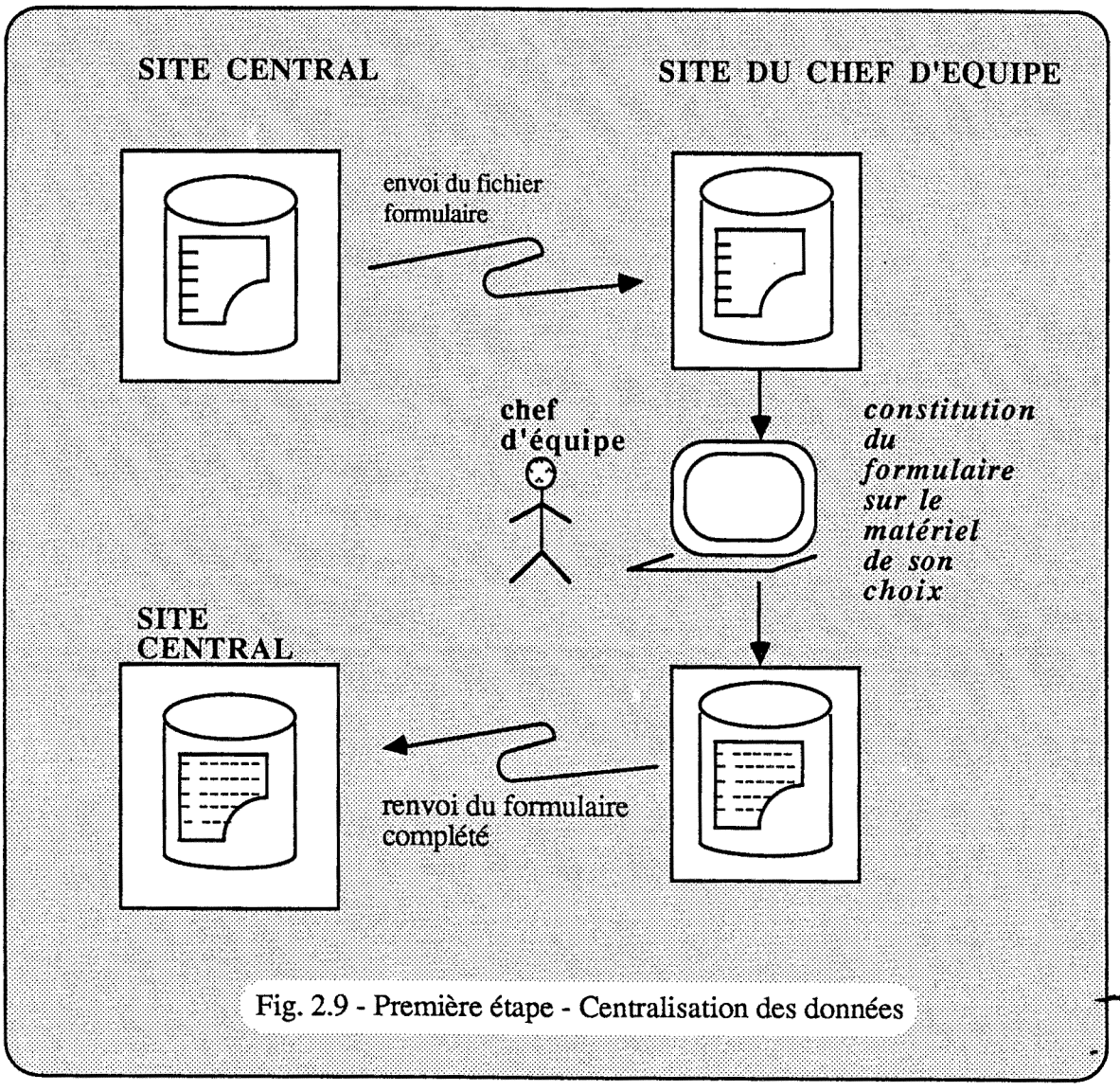



Fig. 2.9 - Première étape - Centralisation des données

Tous ces fichiers types passent dans une "moulinette" d'extraction pour donner les fichiers préformatés exigés au chargement des tables de la base.

### 2.2.2. Automatisation du formatage des fichiers

Une grande majorité des informations nécessaires au chargement de la base peut être extraite automatiquement dans des fichiers préformatés. On entend par "préformaté", un fichier dont les champs correspondent exactement aux attributs de la table pour lequel il est destiné.

Deux sortes de fichiers préformatés sont créés pour le chargement de la base:

- les fichiers créés manuellement à partir des données difficiles à extraire par programme ou d'informations non présentes dans le fichier formulaire (comme par exemple, les adresses des personnes ou des laboratoires).

- les fichiers créés par programme pour extraire et rassembler un maximum d'informations automatiquement.

Huit fichiers préformatés sont créés par trois programmes écrits en langage C qui font la majeure partie du travail. Ces fichiers sont tous préfixés par "fich" (exemple: "fichpers").

Un premier programme extrait toutes les informations primordiales à la création des deux tables suivantes: *pers* et *perspro*. Son rôle est aussi d'affecter un numéro, par ordre d'arrivée, à chaque personne, en commençant à 100 (cela pour des raisons déjà citées précédemment dans le paragraphe 2.1.2 de ce même chapitre).

Un deuxième programme parcourt le fichier formulaire pour rassembler des informations en quatre fichiers préformatés destinés à charger les tables nommées ci-après: *proj, eqproj, projsou, projost;* elles sont toutes liées aux projets. Ce programme affecte un numéro à chaque projet rencontré.

Un troisième programme permet la création de deux autres fichiers préformatés qui sont relatifs aux tables *eqssth* et *prossth*.

Ces huit fichiers sont préformatés avec un caractère spécial entre chaque champ, ceci pour éviter des erreurs en tapant les espaces obligatoires pour les informations de longueur variable dans un champ fixe.

**exemple:** **le fichier préformaté "fichpers"**

106:DUPONT:GILBERT:C:12345:04:01
107:DE LA PAGE:ANNE-CECILE:67895:02:01

L'utilisation de l'outil Unix 'AWK' va permettre de formater le fichier pour donner des longueurs de champs fixes. Ainsi, l'exemple précédent donne le fichier "fpers":

**exemple:** **le fichier formaté "fpers"**

| | | |
|---|---|---|
| 106DUPONT | GILBERT | C123450401 |
| 107DE LA PAGE | ANNE-CECILE | B678950201 |

Toute table de la base sera initialisée avec un fichier de ce type dont chaque champ correspond à un attribut de la table.

Chaque fichier créé par AWK est préfixé avec la seule lettre f. La description de la taille des zones et du caractère spécial sont définies dans un fichier particulier précisé à l'exécution de AWK [BOUR82].

Ainsi, la figure 2.10 montre le traitement effectué sur chacun des fichiers formulaires complétés.

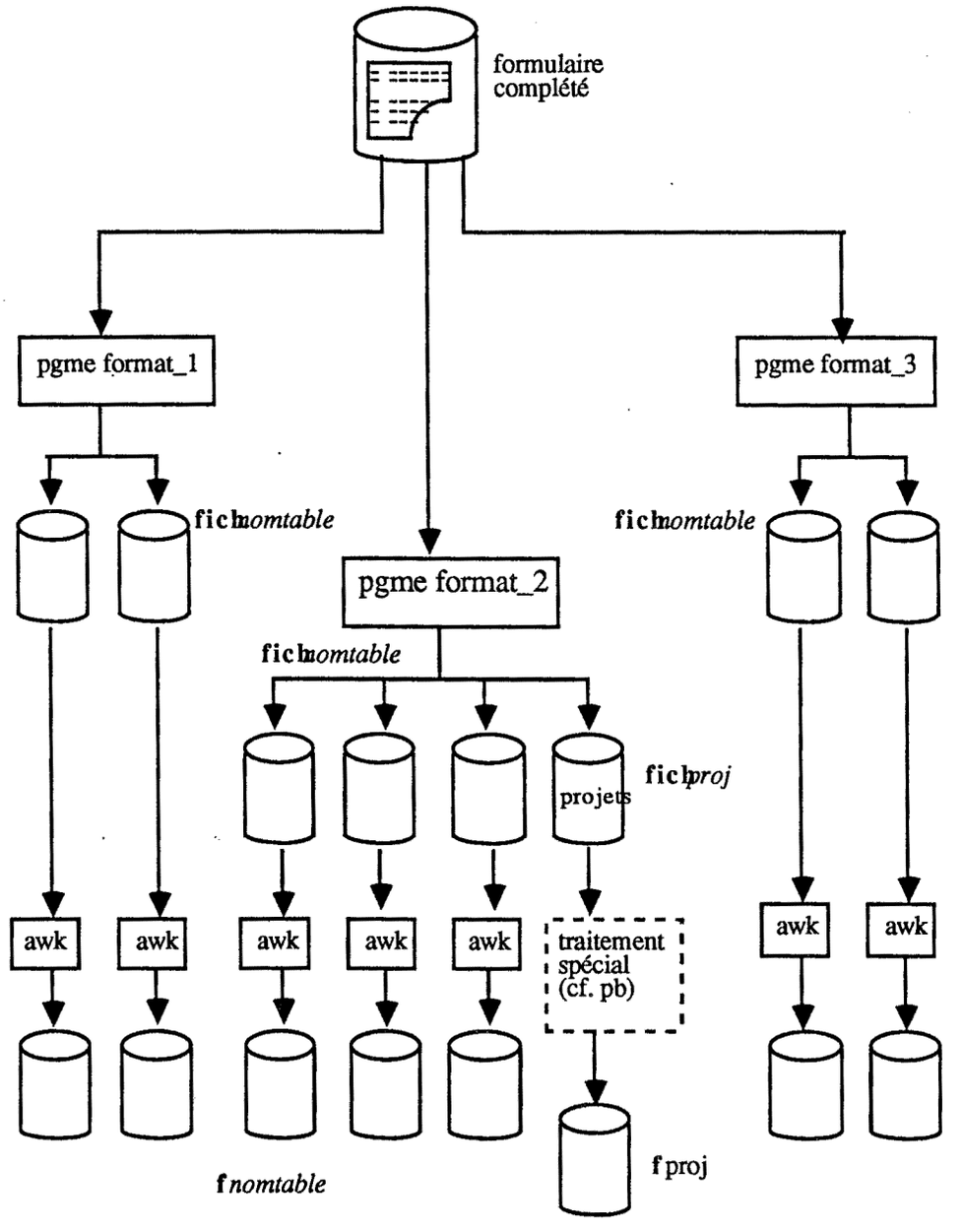



Fig. 2.10 - Deuxième étape - Découpage des données en fichiers formatés

Après cette étape, une fois tous les fichiers formatés créés, le chargement proprement dit peut commencer avec ODL.

### 2.2.3. ODL (Oracle Data Loader)

ODL est l'outil d'Oracle conçu pour le chargement initial d'une table de la base. Une fois les fichiers de données formatés (à un champ correspond un attribut), ODL peut s'exécuter.

ODL a besoin d'un fichier de contrôle que l'on pourrait comparer à un fichier de commandes dont le langage est celui d'ODL (proche de SQL). Ce fichier, dont la syntaxe est détaillée dans [ORA1], est divisé en trois parties. Voici un exemple de fichier de contrôle pour la table *equipe*.

```
define record e-equipe as
        e-equip(char(4)),
        e-idlabo(char(2)),
        e-lbequip(char(80)),
        e-idresp(char(3));

define source file
from fequipe
        length 90
        containing e-equipe;

FOR EACH RECORD
        insert into equipe (idequip, idlabo, lbequip, idresp)
        values (e-idequip, e-idlabo, e-lbequip, e-idresp)
NEXT RECORD
```

Fig. 2.11 - Fichier contrôle de la table 'equipe' (pour ODL)

Les deux premiers ordres correspondent à des déclarations du fichier formaté en entrée et le dernier ordre est celui du traitement. Ce dernier indique que chaque ligne du fichier d'entrée doit être chargée dans une ligne de la table. On peut ainsi ne renseigner que certaines colonnes de la table pour leur valeur initiale, si certains attributs ne peuvent pas être renseignés lors du chargement.

## Notes concernant ODL

Certaines remarques sont à retenir.

- un fichier de contrôle ne peut charger qu'une seule table, d'où la codification de son nom : **nomtable.odl**.

- le paramètre **length** doit être égal à la somme des longueurs des champs du fichier d'entrée plus un caractère, le caractère de fin de ligne (NL).

- le fichier de données formaté doit être trié sur la clé si la table possède un index (par la commande Unix SORT [BOUR82]).

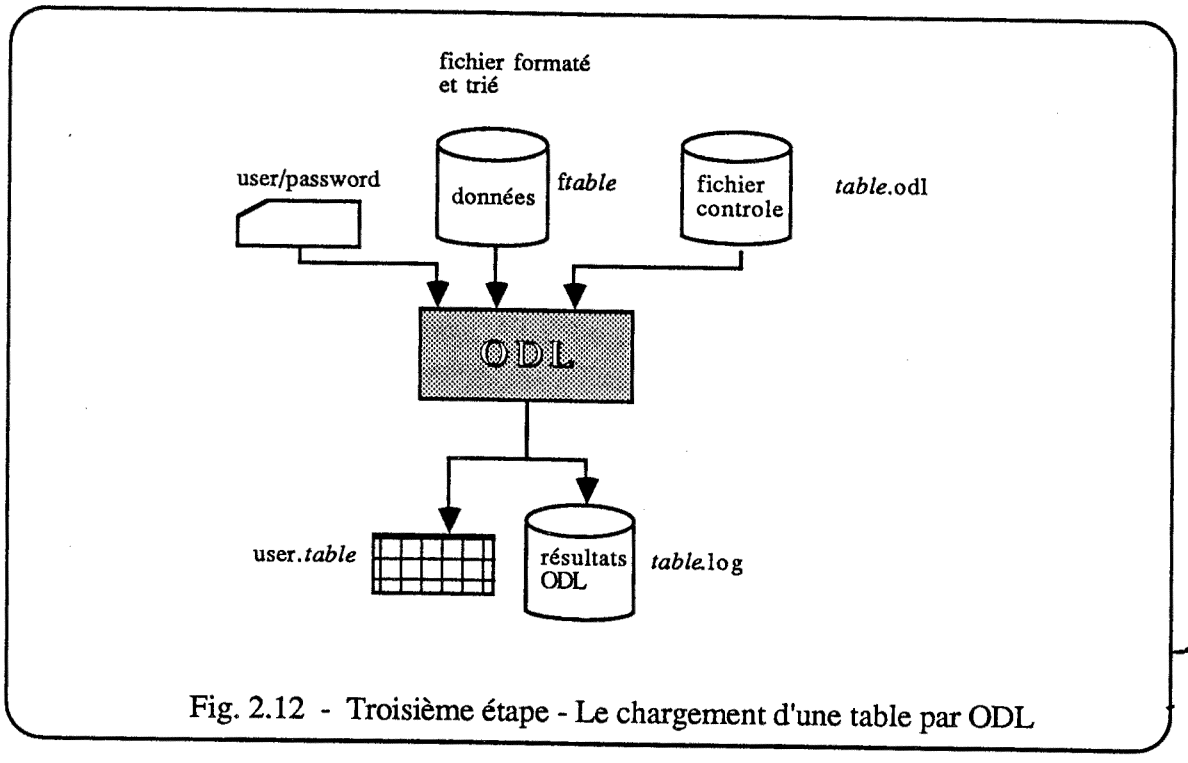



Fig. 2.12 - Troisième étape - Le chargement d'une table par ODL

Le fichier résultat est systématiquement créé à chaque commande ODL (la précision de son nom dans la commande est obligatoire).

La commande de chargement d'une table étant répétitive, un fichier exécutable (commandes Unix) a permis de rendre cette manipulation moins fastidieuse. Seuls le nom

de la commande et celui de la table sont à donner. Ce côté pratique est possible grâce à la codification choisie.

## 2.3. Difficultés rencontrées

La table *proj* est constituée de plusieurs attributs dont le dernier est de type LONG. Celui-ci correspond à du texte composé de phrases et de paragraphes, et donc contenant des changements de ligne (caractère NL). Plusieurs tests sur des colonnes de type LONG nous ont amené à la conclusion suivante: ODL génère des caractères fin de ligne indésirables lorsqu'il y en a déjà dans le texte. Si le texte n'en contient pas, ODL se comporte normalement.

Solution adoptée: le texte LONG est chargé dans la table avec un caractère spécial à la place du caractère fin de ligne. L'opération inverse est effectuée lors de la lecture de ce texte. Un petit programme fait ce travail.

La commande AWK est très pratique pour le formatage des fichiers mais les champs trop longs ne sont pas acceptés. Nous avons donc écrit un programme, toujours en C, pour formater le fichier de données de la table des projets.

Ces deux problèmes techniques modifient quelque peu l'enchaînement des événements de la deuxième étape en ce qui concerne les projets, c'est ce que nous pouvons constater à l'aide de la figure 2.13.

Ces deux exemples correspondent à deux des difficultés techniques que nous avons eu à résoudre. Nous avons été confrontés à un problème plus gênant en raison du niveau de développement du système de gestion de base de données sur le matériel BULL.

Le SGBD Oracle fut installé définitivement sur notre SPS9 avec tous ses outils (précompilateur C, SQL*FORMS, ...) en fin d'été 88, mais il s'agissait d'une version "BETA TEST" suivie de versions ultérieures également "BETA TEST", c'est à dire en test chez le client, ce qui n'a pas simplifié la résolution des problèmes techniques rencontrés. Le système Oracle standard de BULL était la version 4, version trop incomplète pour notre

travail. Nous avons donc découvert les fonctions des outils propres à la version 5, et buté sur leur implémentation parfois incorrecte.

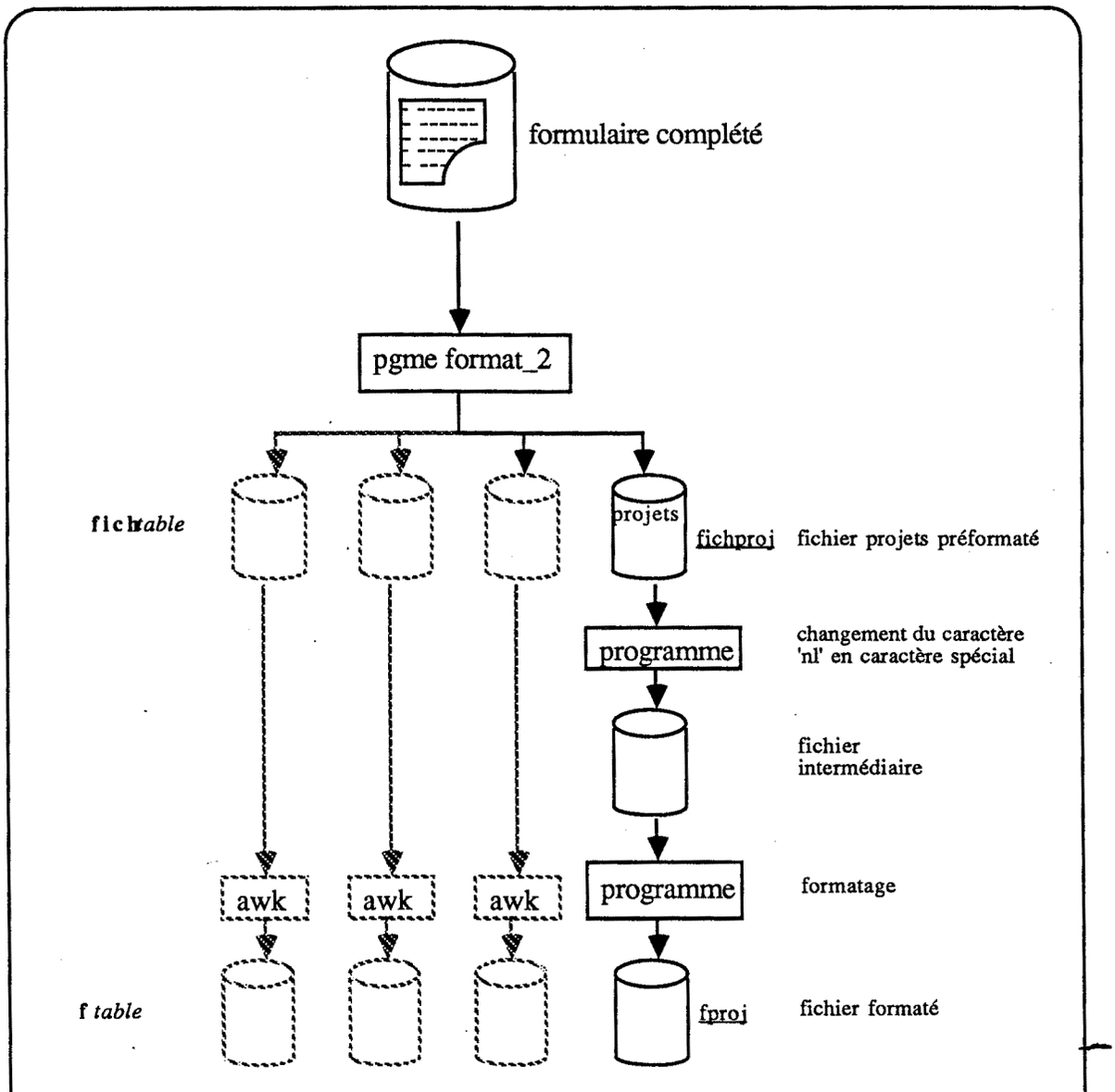



Fig. 2.13 - Deuxième étape pour le fichier formaté de la table des projets

Les problèmes liés aux installations d'Oracle ont engendré une navigation importante entre les différents répertoires pour rechercher, étudier et comparer le rôle de chaque module et pour évaluer si leur état en permettait l'utilisation.

# 3. Conclusion de la première phase

Au début de la période du mémoire, Oracle n'était pas disponible. Or, il devenait inéluctable de voir fonctionner des requêtes SQL sur quelques tables d'essais afin d'avoir une idée plus concrète du procédé. C'est dans le département informatique de la Société Merlin-Gerin que nous avons pu mettre en œuvre un échantillon démonstratif sous le SGBD relationnel DB2 (Database 2 d'IBM). Ces tests furent encourageants et très satisfaisants, préparant la réalisation effectuée ensuite à l' IMAG.

Cette application complète nous a entraîné dans un travail intéressant de communication et de regroupement d'informations lors de contacts avec les personnes des laboratoires.

L'environnement de la recherche et celui de l'industrie sont deux mondes totalement différents. Il n'est pas simple d'obtenir des informations dans un environnement qui évolue. La collaboration entre les équipes est basée sur la disponibilité des gens et sur leur gentillesse. Ainsi, nous avons pu disposer de la documentation Oracle pendant toute la durée de ce mémoire. La possibilité de la consulter à volonté fût un atout primordial pour le déroulement des opérations. Cette remarque paraît triviale. En réalité, nous avons constaté que dans un environnement de laboratoires de recherche, il est rare de disposer de documentation sur les outils dépassant les simples manuels accessibles "en ligne".

Cette application de base de données a permis de mettre en œuvre les principes de travail, relatifs au relationnel, que j'ai acquis auparavant en cours ou lors de mon expérience chez Merlin Gerin avec le SGBD DB2.

# CHAPITRE 3

# COMMUNICATION ET ACCES A DISTANCE

# 1. Introduction

Pour communiquer entre elles, les machines doivent utiliser des langages communs exprimés sous la forme de protocoles. La normalisation dans les réseaux a pour but d'assurer une cohérence à un niveau donné de protocole, en fonction des impératifs des différents partenaires impliqués, à savoir les utilisateurs, les constructeurs et les organismes de télécommunications.

Les travaux de normalisation ont débuté dans les années 70, mais les premiers résultats sont apparus au début des années 80.

L'ISO (International Standard Organisation) à élaboré les normes en matière d'interconnexion de systèmes informatiques que nous présentons ci-aprés.

Avant que ces normes se donnent bien à la réalisation d'outils de communication complets par les différents partenaires impliqués, et notamment par tous les constructeurs, deux possibilités sont offertes:

- l'utilisation de protocoles spécifiques d'un constructeur, par exemple SNA d'IBM, DNA de DEC, DSA de BULL, etc..., avec des "passerelles" reliant les différents mondes,

- l'utilisation d'un protocole standard que l'on peut appeler, par abus de terme, "norme de fait", bien que ses spécifications n'émanent pas d'un organisme officiel de normalisation : TCP/IP. Les "passerelles" ne sont alors plus nécessaires, si ce n'est au simple niveau de l'utilisation de médias physiques différents.

Nous avons ainsi utilisé TCP/IP; nous présentons toutefois le modèle de l'ISO car sa terminologie est une nécessité pour exposer clairement toute application concernant les réseaux.

Nous présentons également le schéma d'appel de procédures distantes à savoir la programmation de bas niveau ("sockets") et un outil de plus haut niveau de type RPC.

Les pages suivantes dévoilent la mise en œuvre du projet et les difficultés abordées lors de sa réalisation.

# 2. Architecture de communication

Les différentes couches du modèle OSI introduisent la présentation de la famille des protocoles TCP/IP. Nous présentons ici, une programation de bas niveau en termes de "sockets" permettant à des processus locaux ou distants de communiquer entre eux. Nous étudions un schéma d'appel de procédure à distance et détaillons l'outil retenu.

## 2.1. Le modèle de référence OSI

ISO a défini:

- un système capable de coopérer et de communiquer avec d'autres systèmes. Il est formé de sept couches, c'est le modèle OSI (Open System Interconnection),
- un ensemble de normes spécifiant les services offerts au niveau d'une couche donnée.

Les couches du modèle sont numérotées de un à sept, cet ordre représentant un niveau d'abstraction de plus en plus élevé.

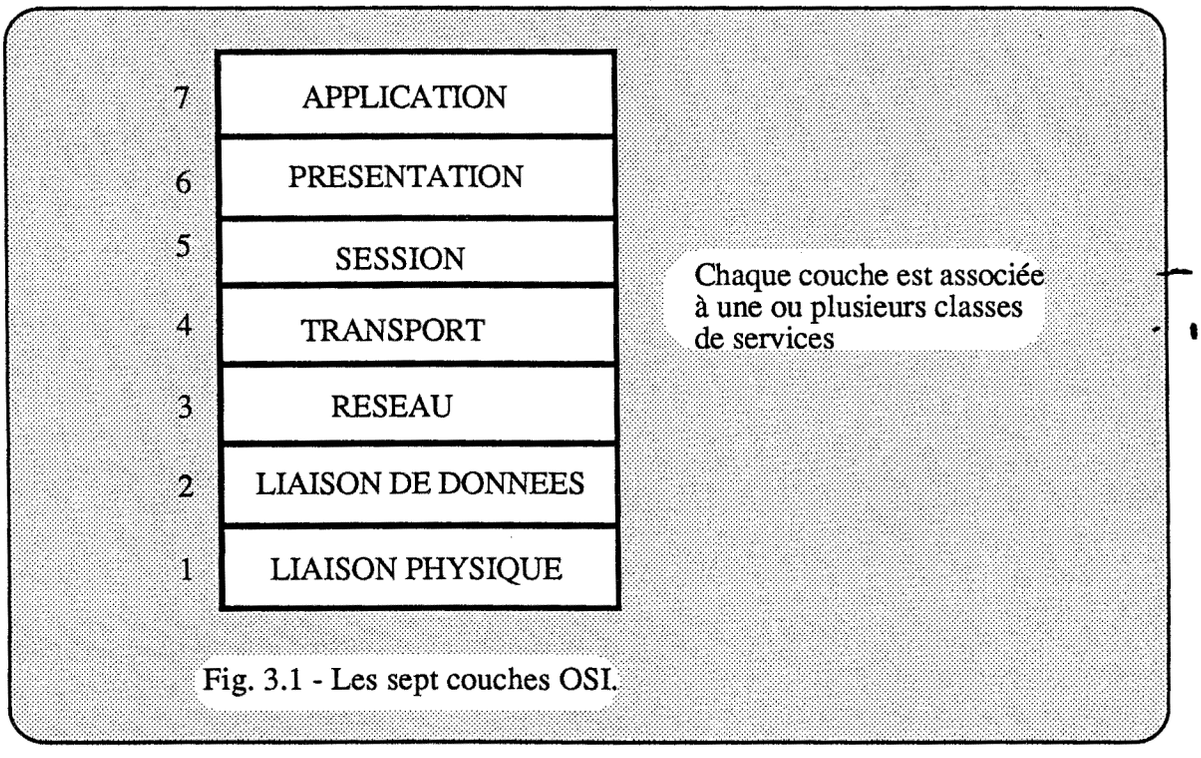



Fig. 3.1 - Les sept couches OSI.

Voyons le rôle attribué à chacune de ces couches.

**• La couche de liaison physique**

Elle fournit les procédures et les fonctions mécaniques et électriques nécessaires pour établir, maintenir et libérer des connexions physiques entre entités de liaisons de données.

Cette couche assure la transmission des données. Elle définit notamment le codage physique d'un bit sur le support physique. L'unité de donnée, à ce niveau, est le bit..

**• La couche liaison de données**

Elle fournit les procédures et les moyens fonctionnels nécessaires à l'établissement, au maintien et à la libération des connexions de liaison de données entre deux systèmes communicants et adjacents (c'est à dire sans passerelles). Une connexion de données est bâtie sur une ou plusieurs connexions physiques.

L'unité de donnée est la trame: ensemble d'informations de contrôle (numérotation, code de détection d'erreurs, champ d'acquittement ...) et de données binaires.

Cette couche assure le contrôle de flux, la détection et le contrôle des erreurs de transmission.

**• La couche réseau**

Elle est responsable de l'acheminement des données d'un nœud origine à un nœud destination, ces données pouvant traverser plusieurs nœuds intermédiaires. Elle se charge de prendre la distance la plus courte et sans encombrement.

L'unité de donnée est le paquet.

Cette couche offre deux types de services: un service en mode connexion (ou circuit virtuel) et un service en mode sans connexion (ou datagramme).

Un service en mode connexion peut être comparé au service téléphonique: une communication est établie en composant le numéro de l'appelé, puis s'instaure le dialogue, qui se termine en raccrochant le combiné téléphonique.

Un service en mode sans connexion correspond plutôt au service offert par la poste: chaque lettre est véhiculée de façon indépendante; l'adresse figure sur chaque enveloppe. Il peut arriver aux P.T.T de perdre une lettre, c'est à l'utilisateur d'effectuer son propre contrôle. De plus, deux lettres n'arrivent pas forcément dans l'ordre dans lequel elles ont été expédiées.

Un protocole en mode sans connexion présente les avantages d'être simple à mettre en œuvre et d'obtenir des performances élevées; en contrepartie, sa fiabilité est légère.

Un protocole en mode connexion offre une communication plus fiable car il permet de maintenir la séquence des données, de préallouer les ressources dans les différents nœuds en réduisant les risques de congestion et de réaliser un contrôle de flux. C'est ici un contrôle de bout en bout, alors que le contrôle effectué au niveau liaison s'opérait entre machines adjacentes. Un nœud intermédiaire peut acquitter un paquet sans arriver à le transmettre au nœud suivant. Dans ce cas, une retransmission s'impose à partir du nœud source.

Les inconvénients d'un mode connexion sont la complexité de mise en œuvre, une consommation de ressources élevée et une augmentation des délais de transmission.

**• La couche transport**

Cette couche doit assurer que toutes les données qu'elle reçoit arrivent correctement à destination. C'est le dernier niveau traitant le transfert de données et il doit, à ce titre, résoudre tous les problèmes de transmission.

La couche transport doit améliorer, si besoin est, le service fourni par la couche réseau. Elle a de plus, l'objectif d'optimiser l'emploi des ressources de transmissions disponibles.

L'unité de donnée est le message.

**• La couche session**

Elle établit la communication entre deux processus et non entre deux machines. Cette couche est responsable de la mise en place et du contrôle du dialogue entre tâches distantes pour les synchroniser. Ce service est nécessaire, notamment à la cohérence des données transmises: elle assure que tous les messages d'une transaction sont traités correctement, sinon la transmission est recommencée.

**• La couche présentation**

Elle assure une compréhension syntaxique entre les utilisateurs, en gérant les formats de données à échanger et en effectuant les transformations nécessaires sur les structures de données pour les rendre compréhensibles entre matériels hétérogènes.

Cette couche a pour but de procurer à l'utilisateur une totale indépendance vis-à-vis :

- du matériel : taille des entiers, ordre des bits, taille des caractères, ...

- du langage : type record ou array de Pascal, type structure ou string de C, ...

- du compilateur : la représentation d'un "packed array of boolean" Pascal correspond, suivant les compilateurs, soit à un bit, soit à un octet, soit à un mot par booléen, ...

Cette couche peut assurer d'autres services comme des transformations cryptographiques pour assurer la sécurité et la confidentialité de la transmission, des compressions de textes pour optimiser la masse des informations échangées.

**• La couche application**

Elle consiste en un ensemble d'éléments de services. Ces éléments sont classés en deux catégories :

- les services de transfert d'informations indépendantes de la nature de l'application,

- les services de transfert d'informations spécifiques (transferts de fichiers, accès à une base de données) et les services particuliers demandés par une application.

Les efforts de standardisation à ce niveau ont permis la définition de plusieurs entités application (éléments de services spécifiques et protocoles associés). Les seules applications normalisées sont le transfert de fichier, le terminal virtuel, l'annuaire, et la messagerie.

Il faut noter la différence entre le modèle OSI des sept couches avec la normalisation des classes de services OSI dans chacune de ces couches.

Tous les protocoles "constructeurs" peuvent être présentés, avec le minimum de distorsion nécessaire ..., comme découpés selon un modèle en couches prétendument "OSI". Un tel découpage ne garantit évidemment pas par lui-même l'intéropérabilité avec les systèmes analogues des autres constructeurs.

## 2.2. Le réseau local Ethernet et les protocoles TCP/IP

Dans le réseau Ethernet, ou plus exactement réseau "ISO 8802.3", chaque utilisateur peut disposer d'un ordinateur individuel fournissant un certain nombre de services; le réseau lui permet en outre, d'accéder à des services communs, partagés entre tous ou partie des utilisateurs. Il est ainsi possible d'équilibrer les investissements entre les serveurs (matériels et logiciels fournissant les services communs) et les ordinateurs individuels, en conservant les avantages propres à chacun : grande capacité, fonctions spécialisées pour les services communs, partage des ressources, facilité d'accès et d'utilisation, autonomie de fonctionnement pour les postes de travail individuels [CORN81].

La figure 3.2 schématise l'organisation générale du réseau Ethernet.

La voie de communication est unique. Elle est constituée d'un câble (en diverses technologies : coaxial fin ou gros, paires torsadées, fibre optique) partagé selon le principe de la diffusion avec retransmission en cas de conflit.

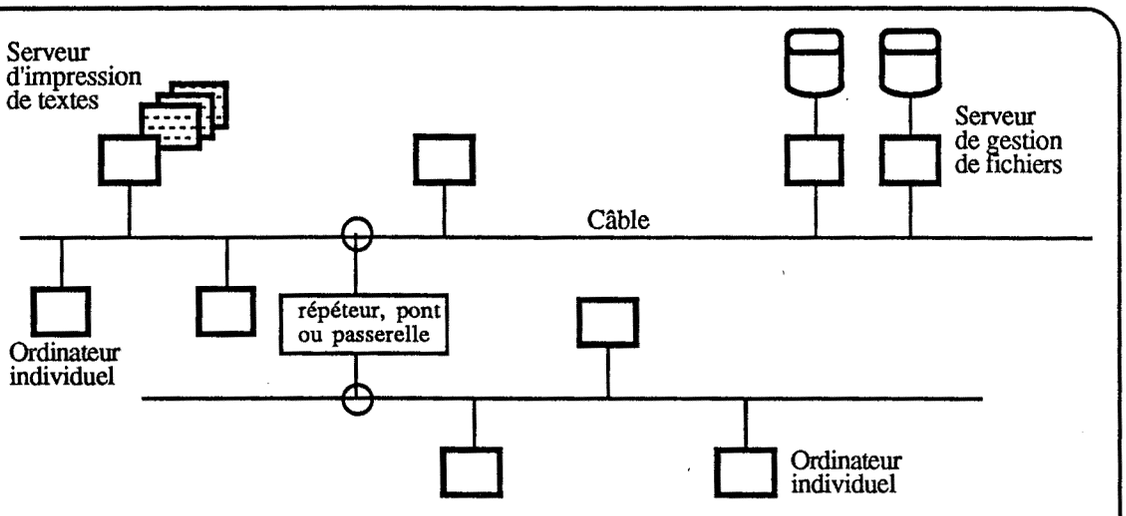



Fig. 3.2 - Schéma général du réseau local Ethernet.

Les caractéristiques du câble limitent sa longueur, mais plusieurs câbles peuvent être reliés entre eux par des dispositifs situés dans les couches 1 ("répéteur"), 2 ("pont"), 3 ("routeur") et 4 ou plus ("passerelle"). Les messages qui transitent sur cette voie sont organisés en paquets comportant les informations suivantes :

- adresse Ethernet de la station émettrice,
- adresse Ethernet de la station réceptrice ou adresse de diffusion,
- données de service des protocoles de routage et de transport,
- données à transmettre.

Chaque station est connectée au câble au moyen d'un contrôleur de communication et d'un transmetteur ("transceiver"). Les transmetteurs transforment les signaux électriques du câble en signaux admissibles par le contrôleur.

TCP/IP désignent un ensemble de protocoles concernant les couches 3 à 7 du modèle OSI. La figure 3.3 schématise la hiérarchie des protocoles TCP/IP.

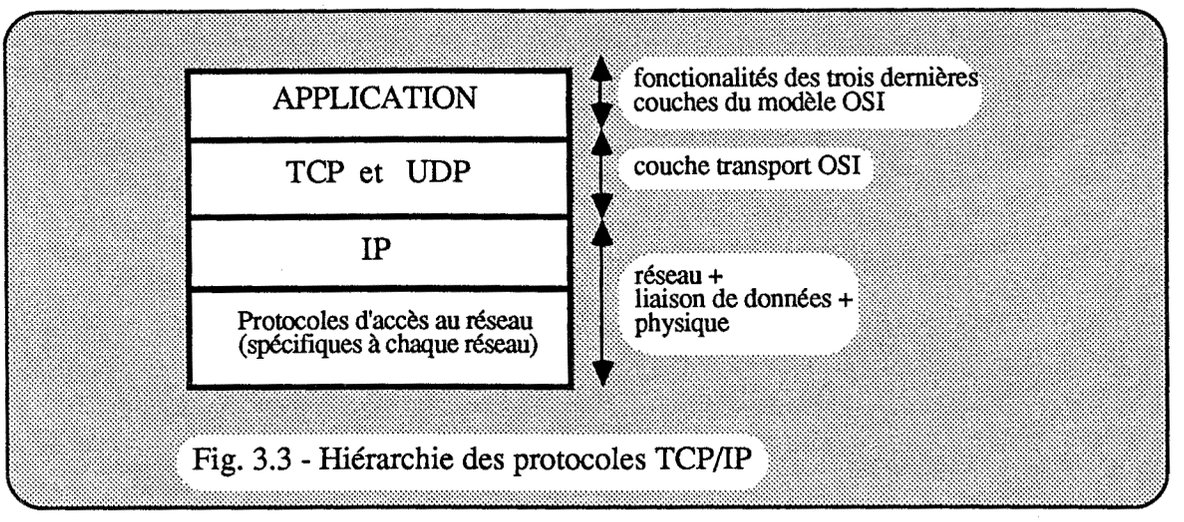



Fig. 3.3 - Hiérarchie des protocoles TCP/IP

## Les protocoles de transport TCP et UDP

On distingue deux classes de transport: UDP (User Datagram Protocol) et TCP (Transmission Control Protocol).

### • Le protocole UDP.

La transmission se fait par datagramme. Un datagramme est un ensemble de données envoyé comme un simple message (le début et la fin sont connus).

UDP travaille en mode non connecté. La communication se fait soit en mode point à point (un seul destinataire) soit en diffusion (plusieurs destinataires).

La transmission n'est pas fiable: il n'y a aucun moyen de savoir si le message est arrivé à destination.

• **Le protocole TCP.**

La transmission est orientée flots d'octets. Le nombre d'octets en émission peut être différent du nombre d'octets en réception. Le contrôle de flux est basé sur un système de fenêtrage en émission et en réception.

TCP travaille en mode connecté. La communication se fait en mode point à point.

La transmission est fiable, un octet envoyé arrive toujours à destination.

**Le protocole de réseau IP (Internet Protocol)**

IP est le protocole de réseau sur lequel s'appuient TCP et UDP pour transmettre leurs segments. Ce protocole remplit deux grandes fonctions :

. le routage,

. la fragmentation.

IP est responsable de l'arrivée des datagrammes à la destination. Celle-ci est indiquée par l'adresse destination.

Le ***routage*** est la fonction qui permet de définir le chemin nécessaire pour une communication de bout en bout.

La ***fragmentation*** est l'opération qui permet d'adapter la taille des paquets élémentaires ("fragments") aux contraintes spécifiques des médias de communication.

Nous précisons ces deux notions en Annexe 4.

## 2.3. Programmation de bas niveau (en termes de "socket")

Jusqu'à maintenant, nous avons décrit la façon dont un flot de données est divisé en datagrammes puis envoyé à un autre ordinateur, et enfin rassemblé en un flot de données. Pour compléter cet état de communication, il est indispensable d'établir une connexion à un ordinateur et de contrôler la transmission.

L'interface, entre les protocoles de communication et les applications au niveau utilisateur, est assurée par une couche de logiciel appelée "sockets". Une traduction approximative de ***"socket"*** pourrait être ***"port"***.

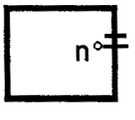
Un des aspects de la répartition peut être mis en évidence dans le cas de n'importe quel réseau local: la relation ***"client-serveur"***. Pour chaque service réseau, un ou plusieurs ordinateurs "serveur" sont à l'écoute des requêtes des ordinateurs "client". Un ordinateur joue, bien entendu, souvent les deux rôles de serveur et de client pour des services différents, voire pour un même service mais entre partenaires - programmes ou utilisateurs - différents. L'asymétrie de la relation client-serveur n'implique aucune asymétrie quant aux transferts. La relation met en œuvre un dialogue entre deux machines: la communication est toujours bidirectionnelle, sauf pour les serveurs de diffusion.

L'ordinateur client et l'ordinateur serveur communiquent en échangeant des messages à travers le réseau. Le client a l'initiative de l'échange qui se déroule selon des protocoles définis dépendant de l'application.

La gestion du partage (priorité, exclusion mutuelle, mise en attente et réveil des clients) est entièrement à la charge du serveur.

Des primitives définies par l'Université de Berkeley exercent ces fonctions qui permettent d'établir une communication de processus à processus, que ces processus soient ou non exécutés par une même machine:

La primitive de création : fonction **socket**

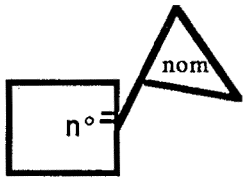



no = socket (...)

Elle a pour effet d'ouvrir un port en affectant un numéro au processus qui exécute cette primitive.

Il existe les mêmes primitives pour gérer un "pipe" (communication locale entre deux processus selon le schéma "producteur-consommateur") et pour gérer une communication entre un processus d'une machine et un processus d'une autre.

La primitive de désignation : fonction booléenne **bind**




y = bind(no,nom)

Elle a pour effet de donner à un port *"no"*, un nom *"nom"* connu dans tout le système et de rendre une valeur booléenne en fonction de l'établissement de la connexion.

L'envoi et la réception de la chaîne de caractères.

On distingue deux stratégies possibles pour la communication:

- *la communication avec établissement de connexion* (nous sommes dans le cas de TCP), les messages ne spécifient pas d'adresse et les ordres de lecture/écriture sont simplement Read et Write,

- *la communication sans connexion* (nous sommes dans le cas de UDP), les messages doivent spécifier l'adresse de la machine. Les ordres de lecture/écriture sont alors "Read from" et "Send to" .

- **La communication avec établissement de connexion**

Elle fait intervenir les primitives *listen, accept et connect.*

La communication en mode connexion permet un schéma "clients-serveur" par l'intermédiaire de l'instruction "fork" dans le programme serveur. Cette instruction duplique le processus serveur et son contexte dans un processus appelé "fils". Le serveur lancé acquiert un nouveau "socket" et c'est lui qui répond à la demande de connexion du client en lui donnant l'adresse de ce nouveau "socket".

L'exécution des primitives *listen* et *accept* dans un processus P1 a pour effet de mettre celui-ci en attente des demandes d'ouverture de connexion effectuées par des processus distants au moyen de la primitive *connect*.

Dans l'exemple de la figure 3.4, le processus P1 joue le rôle du serveur et le processus P2, le rôle du client.

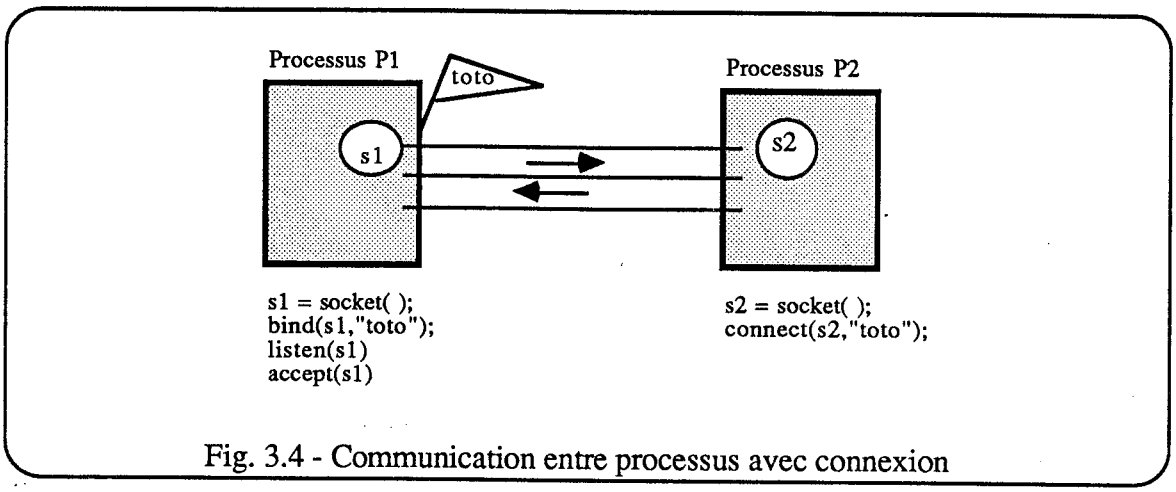



Fig. 3.4 - Communication entre processus avec connexion

Les ordres de lecture/écriture sont implicites et ne nécessitent pas la précision des adresses de machine.

- **La communication sans connexion**

Dans le mode sans connexion, nous avons toujours la présence du serveur, mais les requêtes sont satisfaites individuellement sans établissement de dialogue (une requête, une réponse). Les ordres "Read from" et "Send to" permettent le transfert des messages entre les machines.

Dans l'exemple de la figure 3.5, chacun des processus ouvre un port en lui affectant une étiquette. Les processus P2 et P3 envoient respectivement les messages m2 et m3 au processus P1 dont l'étiquette doit être connue des processus clients.

Le processus serveur (P1) ne connait l'étiquette d'un client que par les renseignements donnés dans l'en-tête du message arrivé. C'est donc par l'intermédiaire de l'en-tête que le processus serveur peut répondre à un client.

La synchronisation du dialogue reste à la charge des processus communicants. Les protocoles de communication sont libres et doivent être discutés entre les processus.

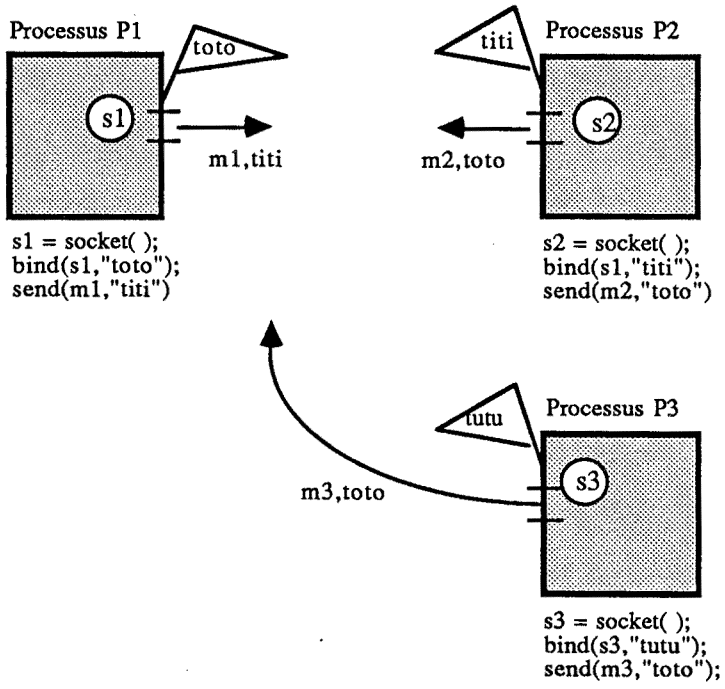



Fig. 3.5 - Communication sans connexion entre processus.

Les principales différences entre des deux modes de connexion sont la fiabilité de la transmission et la nature de l'information transférée. Le mode sans connexion ne connaît pas la primitive "connect" donc il travaille sans établissement de dialogue.

Nous avons utilisé le mécanisme des "sockets", très puissant et aussi de très bas niveau, surtout dans un but de formation, afin de comprendre en détail, le fonctionnement des communications inter-processus.

Nous donnons un exemple de programmation avec le mécanisme des "sockets" en Annexe 5. Il représente un serveur de multiplication. Cet exemple sera repris plus loin avec l'outil RPC® pour en établir une comparaison.

® RPC: Remote Procedure Call

Pour la réalisation effectuée, nous avons eu recours à un outil de plus haut niveau, basé lui même sur les "sockets", l'appel de procédures distantes RPC.

## 2.4. Choix d'un schéma d'appel de procédures à distance

L'appel de procédure à distance ou Remote Procedure Call (RPC) correspond à l'extension de l'appel de procédure locale dans un environnment de machines interconnectées par un réseau.

Nous étudions ici, le schéma de communication de type RPC et nous présentons ensuite le mécanisme de l'outil retenu.

### 2.4.1. Le schéma de communication de type RPC

Nous venons de présenter la programmation à l'aide de "sockets". Voyons ce que nous apporte un outil de type RPC. La communication par RPC a des avantages:

- elle exige moins de programmation,
- elle est plus facile à vérifier, ou du moins, elle est moins génératrice d'erreurs.

En contrepartie, le schéma de communication est bien entendu plus figé. En l'occurrence, ce schéma répondait à nos spécifications.

Une procédure définie sur un serveur doit pouvoir être appelée comme si elle était définie localement. Or, le code des procédures est uniquement accessible chez le serveur.

Le système de communication est synchrone. Le serveur peut devenir client d'un autre serveur.

La figure 3.6 illustre l'aspect général du rôle de l'outil. Dans l'appel d'une procédure distante comme dans le cas d'un appel local, l'environnement du processus appelant est suspendu. Les paramètres sont passés, via le réseau, à l'environnment de l'appelé et la procédure désirée est exécutée. Au retour, la procédure appelée transmet ses résultats à l'environnement de l'appelant qui est alors réveillé [RPC86].

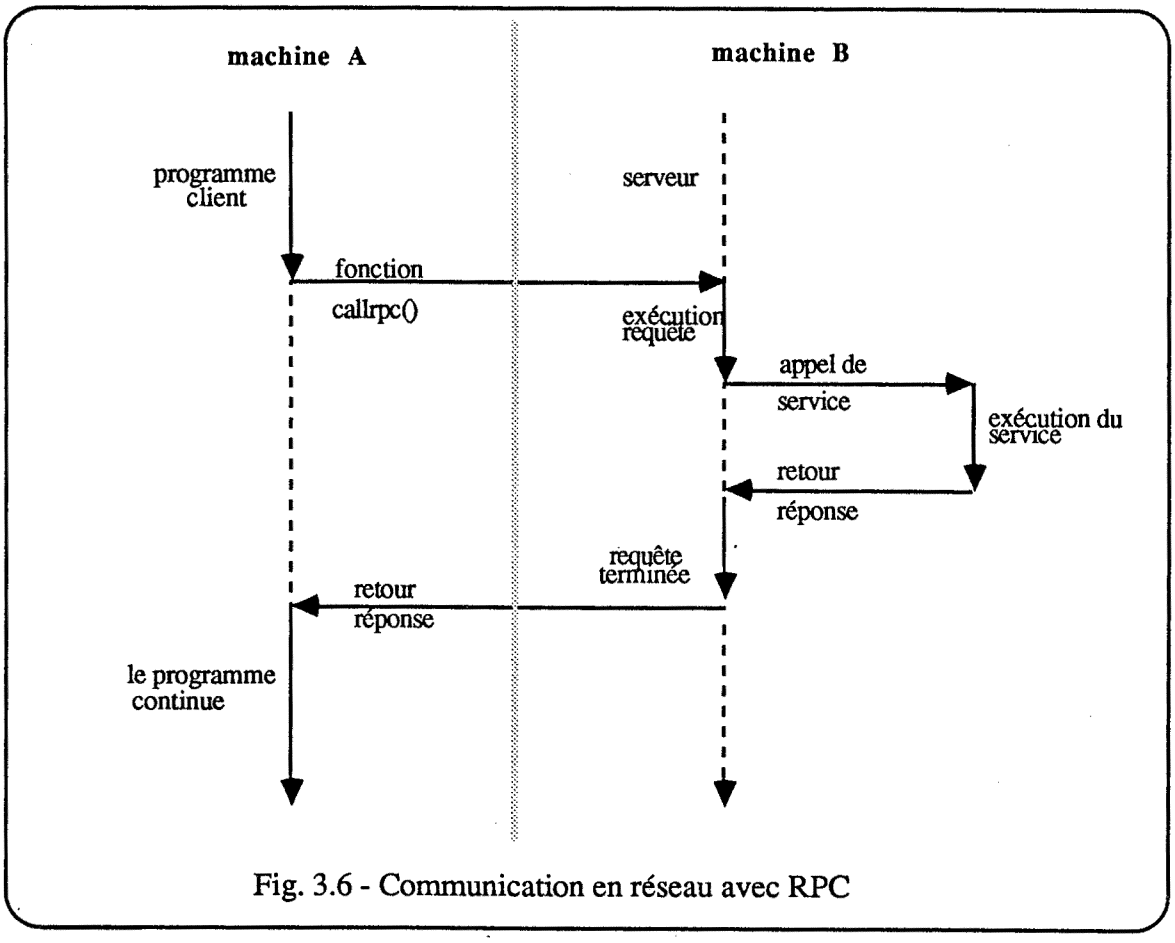



Fig. 3.6 - Communication en réseau avec RPC

Le client appelle en premier une procédure pour envoyer un paquet de données au serveur. Lorsque le paquet arrive, le serveur appelle une routine, exécute le service demandé, renvoie la réponse et l'appel de procédure retourne chez le client.

L'interface RPC est divisée en trois couches:

- la couche haute est totalement transparente au programmeur. Le programme contient alors des appels de procédures distantes tout comme s'il appelait une procédure locale.

- les routines de cette intermédiaire sont utilisées dans la plupart des applications et évitent à l'utilisateur de connaître les "sockets".

- la couche basse est destinée à des applications plus sophistiquées.

Des interfaces de communication sont indispensables entre les processus appelants et appelés. Une procédure nommée ***"stub client"*** gère une interface "socket" entre le

programme client local et la procédure distante. Inversement, le retour de la procédure distante s'effectue par l'intermédiaire d'une procédure appelée ***"stub serveur"*** qui gère, de même, les communications au niveau "socket" [DIAZ85].

Les procédures appelées par le client sont déclarées dans le "stub client". Ce dernier appelle les procédures déclarées dans le "stub serveur". Le "stub serveur" appelle les procédures définies dans le programme serveur.

L'outil doit permettre de générer ces deux modules "stub" automatiquement, pour cela, il faut spécifier l'interface RPC.

Le "stub client" doit être lié au code du client sur le site client et le "stub serveur" avec le code serveur sur le site serveur.

Voyons plus clairement le fonctionnement d'un système de communication de type RPC, à travers le schéma de la figure 3.7.

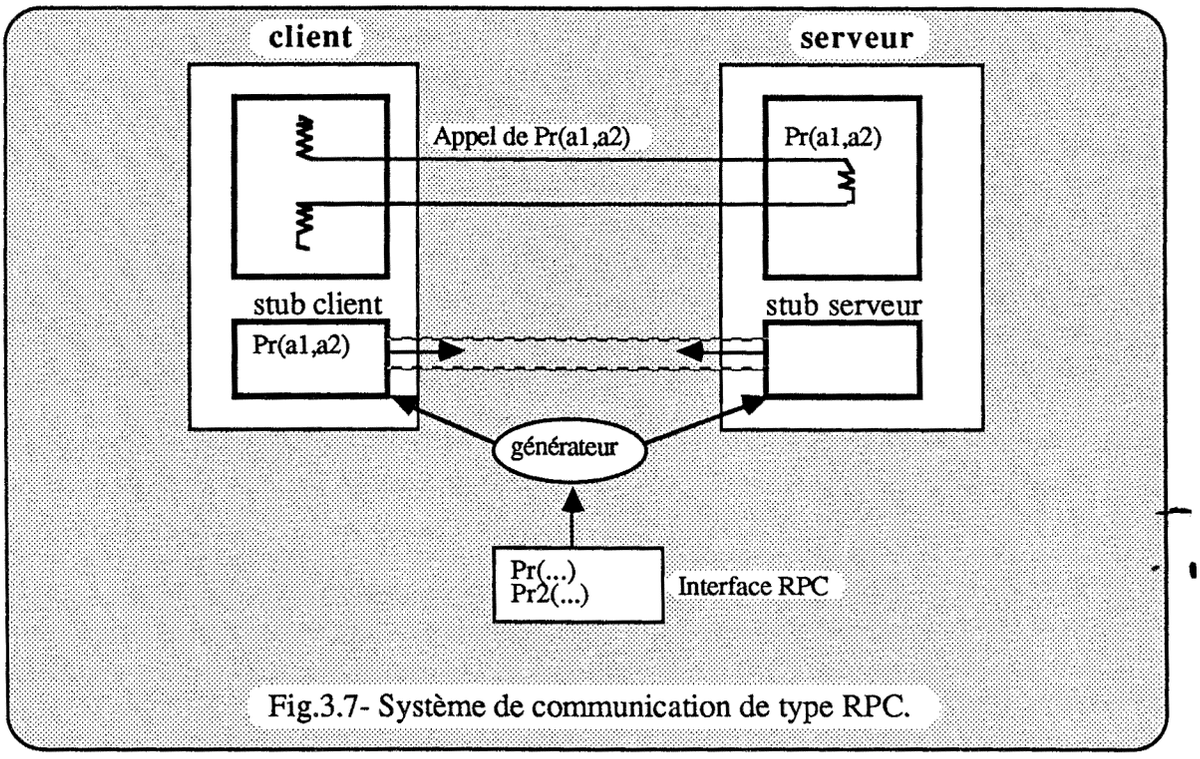



Fig.3.7- Système de communication de type RPC.

Pour chacune des procédures appelables à distance, il existe dans le "stub client", la définition d'une procédure possédant la même syntaxe: même identifiant et mêmes

paramètres. Le "stub client" contient toutes les procédures permettant la transformation des paramètres envoyés et celle des résultats reçus.

Le "stub serveur" contient les procédures inverses de celles du "stub client" permettant le décodage des paramètres reçus et le codage des résultats à envoyer au client.

Dans le paragraphe suivant, nous détaillons le fonctionnement de l'outil RPC.

### 2.4.2. RPC (Remote Procedure Call)

Comme il s'agit d'un outil d'intérêt plus général, les spécifications en ont été mises dans le domaine public; RPC a ainsi pu être implanté sur la plupart des machines UNIX, au prix d'un partage préalable des "sockets" dans le cas des System V, ainsi que sur d'autres systèmes (MS-DOS, VMS). RPC a été, tout d'abord, défini par la Société SUN comme couche intermédiaire entre les sockets et le système de fichiers répartis NFS®, pour faciliter la réalisation de ce dernier [DIAZ85].

L'outil RPC présente l'avantage de s'adapter à toute machine et à tout système d'exploitation. Son système de communication est basé sur celui des sockets.

TCP/IP est destiné à être utilisable sur n'importe quel ordinateur. Malheureusement, la représentation des données n'est pas normalisée.

XDR (EXternal Data Representation) fournit un moyen simple de résoudre ces problèmes. La philosophie XDR fait partie de l'ensemble RPC/XDR imaginé par SUN et largement distribué puisqu'il est mis dans le domaine public, c'est à dire que l'on peut l'implanter sur toutes les machines. Le protocole XDR transforme les données en un code de représentation unique tels que chaînes de caractères, entiers, booléens, unions et tableaux [LAFO88].

Chaque type élémentaire possède, dans une librairie propre à chaque machine, une fonction permettant de transformer la représentation de sa valeur en une représentation indépendante de tout support. les structures complexes sont représentées à partir des types de données de base de XDR. Le protocole XDR se situe au niveau de la couche présentation.

---

® NFS: Network File System

L'outil "RPCGEN" permet de générer automatiquement les procédures de codage et de décodage des structures propres à l'application. Il suffit de les décrire dans une forme très proche de la déclaration de structures C pour que "rpcgen" génère le squelette de l'application côté client et côté serveur; ces intervenants sont les "stubs" dont nous avons parlé précédemment.

RPCGEN accepte la définition de l'interface d'un programme distant écrit dans un langage appelé le langage RPC, similaire au C. Il produit une sortie en langage C qui comprend les ""stubs" client et serveur, les routines XDR pour les paramètres et les résultats, et un fichier "header" qui contient les définitions communes.

Cet outil génère la plupart du travail fastidieux, ce qui nous permet de passer du temps sur l'application plutôt que de le passer à débogger le code de l'interface réseau.

Les fichiers créés par ***rpcgen*** peuvent être compilés et liés normalement. Le programmeur écrit les procédures du serveur et les lie avec le "stub serveur" produit par *rpcgen* pour obtenir un programme serveur exécutable.

Pour utiliser un programme distant, le programmeur écrit un programme principal ordinaire qui fait des appels de procédures locales au "stub client" produit par *rpcgen*. En liant ce programme avec le "stub client", on crée un programme exécutable.

Voyons une application toute simple comme ***exemple***. Soit un *serveur de multiplication* que nous appellerons "mult" dont le rôle est de multiplier deux nombres donnés par l'utilisateur et de lui renvoyer le résultat de l'opération. Les programmes client et serveur sont écrits et s'appellent respectivement "multcl.c" et "multsv.c". L'interface est également à définir, elle est la clé de la génération. Elle contient la déclaration de toutes les procédures susceptibles d'être appelées à distance par le client ainsi que le détail de leurs paramètres.

La figure 3.8 montre les différents modules générés.

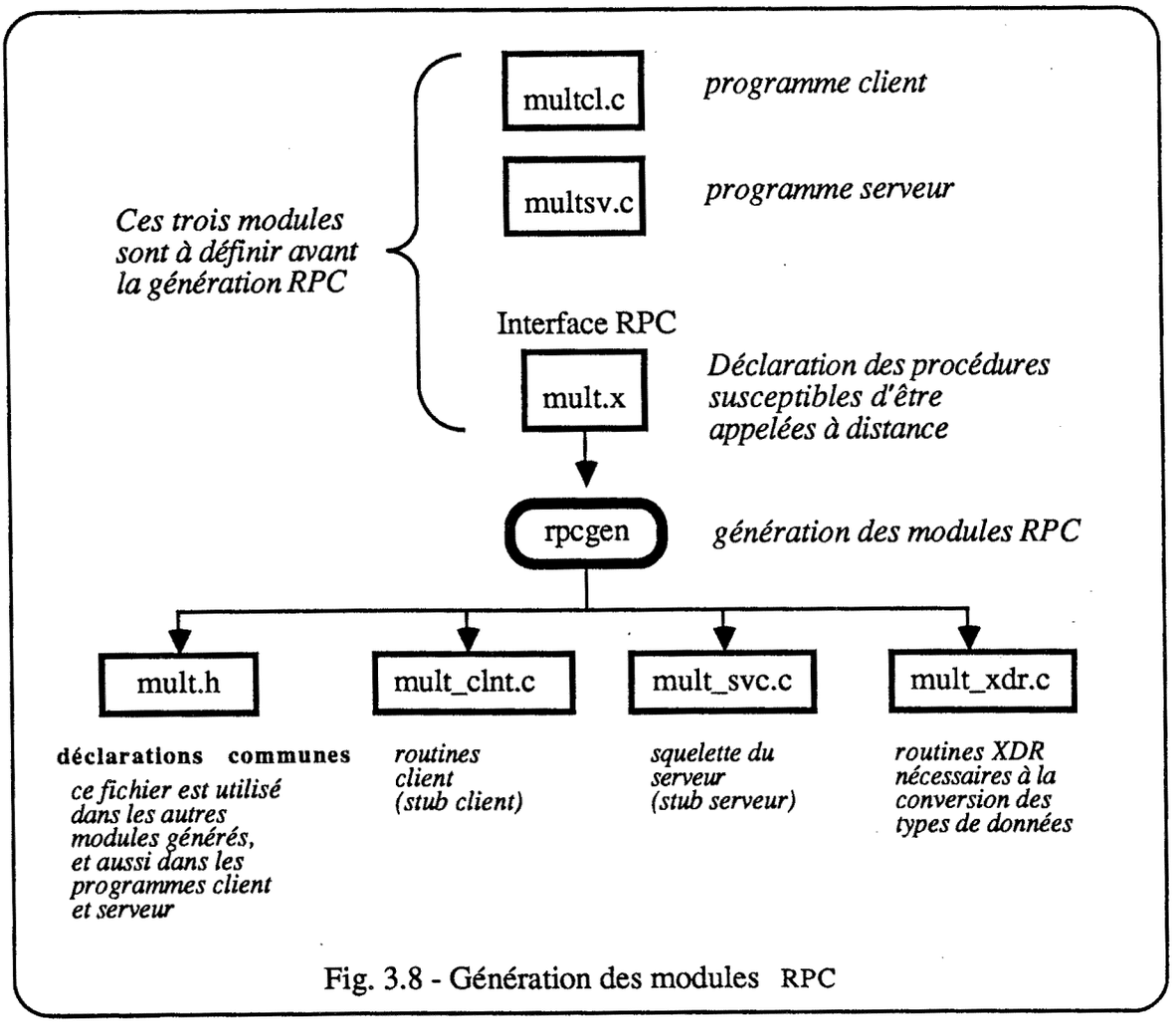



Fig. 3.8 - Génération des modules RPC

Les pièces de ce puzzle (détaillées dans l'Annexe 6) s'assemblent suivant les lignes ci-dessous dans le cas d'une programmation en langage C (cc étant le compilateur C):

```
rpcgen mult.x
cc multcl.c mult_clnt.c mult_xdr.c -o    mult_client
cc multsv.c mult_svc.c mult_xdr.c -o    mult_serveur
```

Le serveur doit tourner en permanence en "arrière plan". Un numéro de programme lui est attribué dans le fichier interface (le fichier mult.x dans l'exemple précédent). Un seul serveur de même numéro peut être actif à un moment donné.

Supposons que nous copions le serveur sur une autre machine nommée CASTOR (il faut alors exécuter à nouveau les trois lignes d'assemblage afin de créer un nouveau code exécutable dépendant de la machine) et nous le lançons en arrière plan à l'aide de la commande : **mult_serveur &**

En repassant sur notre machine locale appelée "IMAG", nous pouvons demander au serveur d'effectuer la multiplication des deux chiffres donnés en paramètres par la commande : **mult_client 300 60**

Le serveur envoie alors le résultat : 18000.

Les différents programmes sources intervenants dans ce serveur de multiplication sont disponibles en Annexe 6. Ils peuvent être comparés au serveur de multiplication écrit à l'aide du mécanisme des "sockets" placé en Annexe 5. On remarque rapidement que le nombre de lignes généré par rpcgen est important. En contrepartie, le nombre de lignes écrites avec RPC par le programmeur est moindre par rapport aux lignes écrites avec les "sockets".

En résumé, plusieurs étapes sont nécessaires à la construction d'une communication entre processus :

- définir les procédures appelables à distance,
- définir leurs paramètres,
- définir l'interface à l'aide de ces procédures,
- générer les "stubs" à l'aide de l'outil rpcgen.

Chaque machine, voulant accéder au service proposé par un serveur, doit être dotée des outils RPC et RPCGEN.

# 3. Mise en œuvre et problèmes

Nous étudions dans ce chapitre, les structures possibles d'organisation entre clients et serveur. Nous exposons les problèmes de communication entre machines différentes et nous expliquons notre solution.

Les possibilités d'interfacer Oracle et le serveur sont diverses. Nous détaillons trois cas d'interface de communication et expliquons notre choix. Nous terminons ce chapitre sur les difficultés rencontrées lors de la réalisation du projet.

## 3.1. Etude du projet

La base de données des projets de recherche de l'IMAG étant centralisée sur une machine, un serveur assurera les services demandés par les clients.

Les programmes clients seront installés sur toutes les machines du réseau susceptibles de devenir clientes du serveur, y compris la machine où se trouve la base de données et le serveur lui même.

Un premier type de structure d'organisation **n clients et un serveur** est représenté à la figure 3.9.

Cette structure présente un inconvénient majeur, celui de la cohabitation du serveur avec le SGBD Oracle dans le cas de plusieurs clients. En effet, le serveur ouvre une connexion dans Oracle pour un client en affectant au programme que représente le serveur, une zone de données qui doit être unique. Cette contrainte est imposée par Oracle qui ne saurait gérer deux utilisateurs du SGBD dans le même programme. Le serveur ne peut donc satisfaire qu'un seul client à la fois.

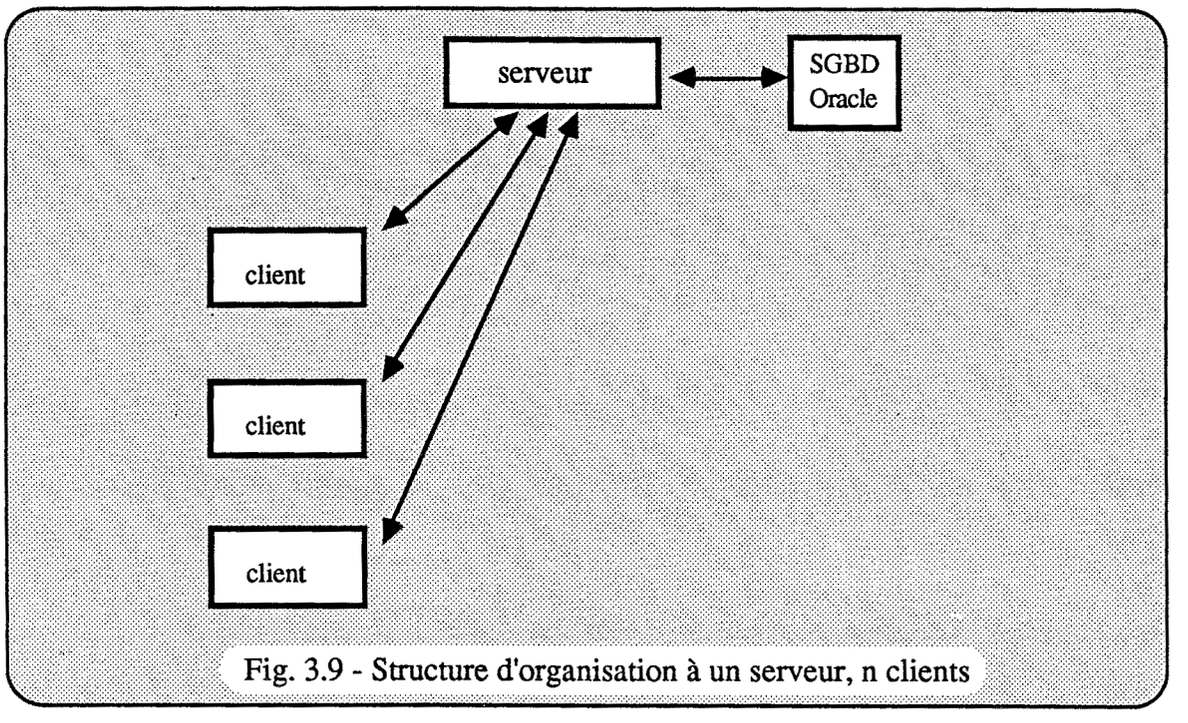



Fig. 3.9 - Structure d'organisation à un serveur, n clients

De plus, si le serveur est défectueux à un moment donné, tous les clients actifs sont pénalisés.

Cette structure d'organisation n'est donc pas envisageable pour un serveur de cette nature. La solution idéale serait un serveur par client.

Un deuxième type de structure d'organisation est donc étudié : **n+1 serveurs et n clients.**

Le client ne connaît qu'un seul serveur, appelé "démon serveur". Ce dernier se charge du lancement des serveurs unitaires à chaque demande d'un client et effectue le lien de communication entre le serveur et le client.

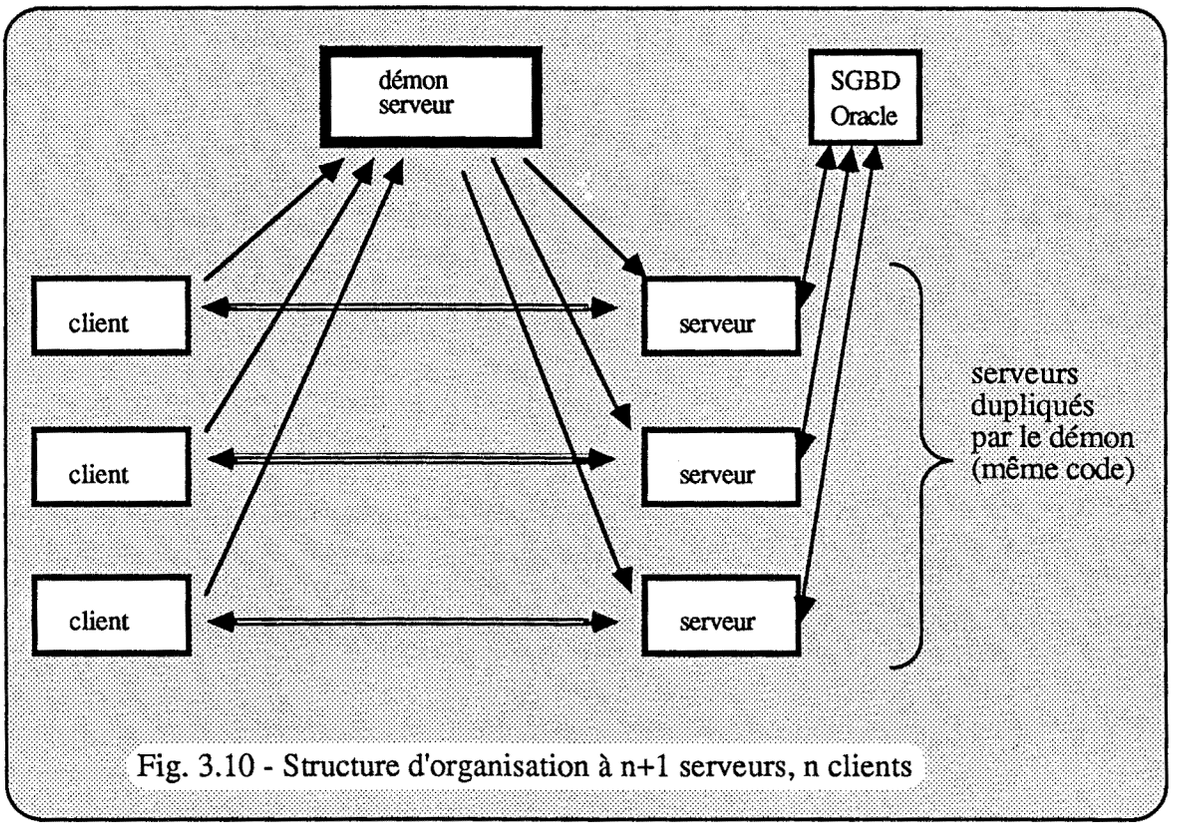



Fig. 3.10 - Structure d'organisation à n+1 serveurs, n clients

Ce type de structure liant un serveur à un client ne pénalise que le client attaché au serveur en cas de problème au niveau de ce dernier.

Il est évident que l'on ne connaît pas à l'avance le nombre de clients plausibles, ni par conséquent, le nombre de serveurs qu'il faut rendre actifs. Cette organisation représente toutefois, le type de structure approprié à notre travail.

Nous exposons les problèmes et les solutions concernant la réalisation de cette deuxième structure dans le chapitre "Difficultés rencontrées".

Revenons à un schéma de communication plus général (figure 3.11) pour la mise en œuvre de notre prototype. Requêtes et réponses naviguent à travers le réseau entre un client et le serveur grâce au mécanisme de l'appel de procédure à distance.

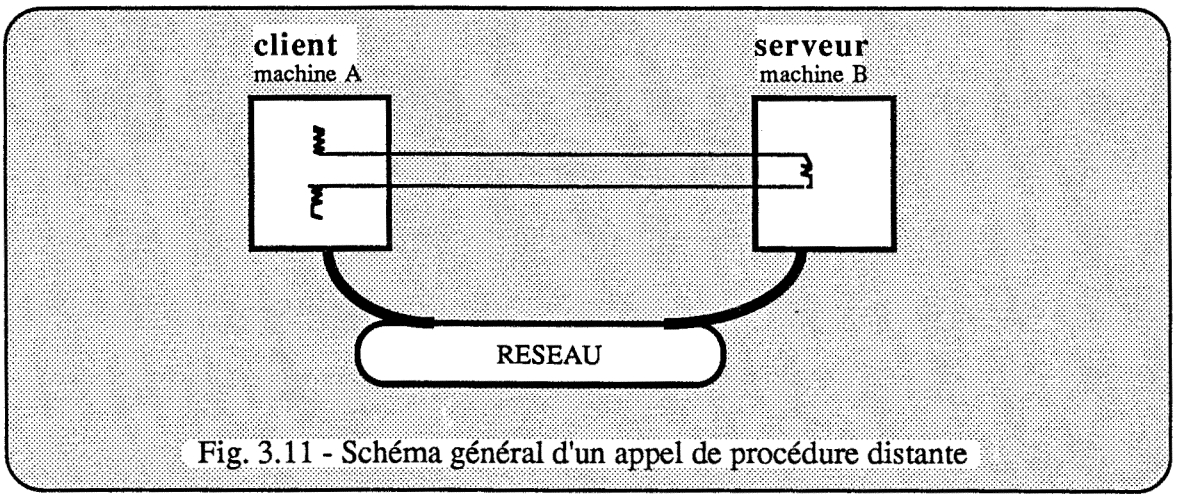



Fig. 3.11 - Schéma général d'un appel de procédure distante

Le programme gérant l'interface utilisateur représenté à la figure 3.12 est un ensemble de menus parmi lesquels l'utilisateur fait un choix. A chacun de ces choix, correspond une requête SQL® à exécuter sur le site serveur. La requête transite via le réseau. Le programme du serveur la transmet à Oracle. La réponse obtenue suit le chemin inverse.

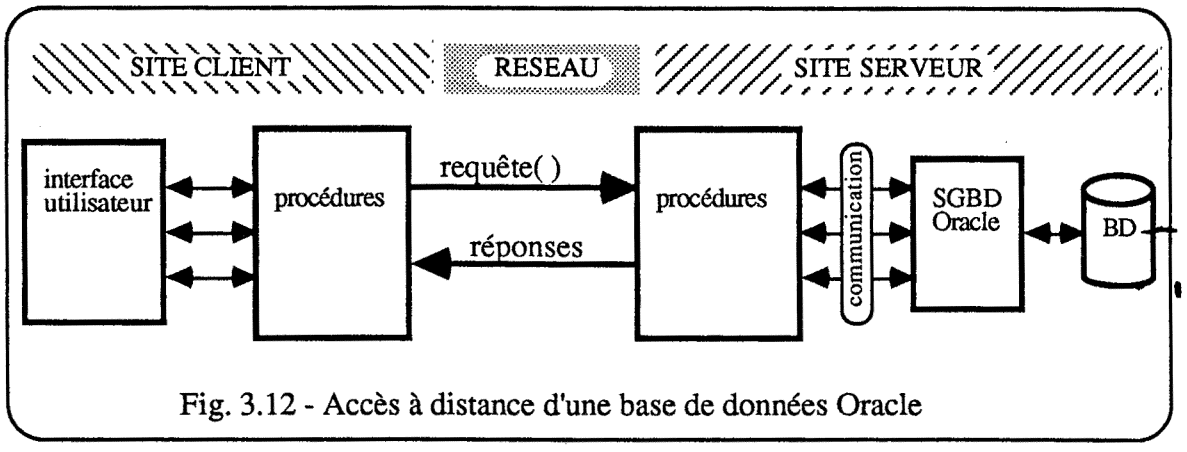
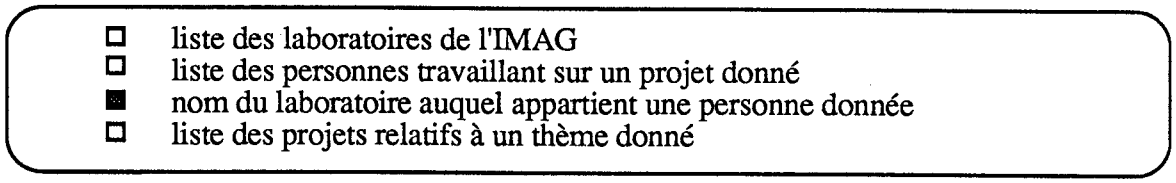



Fig. 3.12 - Accès à distance d'une base de données Oracle

® SQL: Structured Query Language

***Exemple de démarche***

L'utilisateur lance sa transaction d'interrogation de la base IMAG et un menu lui est proposé:




Si l'utilisateur choisi la troisième ligne, le nom de la personne lui est demandé; supposons qu'il saisisse le nom "Tartampion".

La requête SQL constituée à l'aide du paramètre "NOM" est alors:

```
SELECT SIGLABO, LBLABO FROM LABO L, PERS P
       WHERE NOM = "TARTAMPION"
       AND   L.IDLABO = P.IDLABO;
```

La requête est prise en charge par l'environnement RPC. Elle est codée pour le transfert sur le réseau et décodée à son arrivée. Un interface de communication entre le serveur et le SGBD Oracle permet de soumettre la requête et d'obtenir une réponse.

La réponse retourne vers le programme d'origine, en passant également par les phases de codage et décodage indispensables à la transmission, pour être affichée au bon gré du programmeur.

On a vu dans la première partie de ce mémoire qu'un programme écrit dans un langage hôte (ici, en langage C) peut contenir des ordres SQL, moyennant une certaine programmation à respecter. Il y a deux façons principales de programmer du SQL :

- soit les requêtes sont toujours les mêmes ; elles sont alors figées dans le programme,

- soit les requêtes ne sont pas connues à la compilation du programme; elles ne sont pas figées dans le programme, c'est du SQL **dynamique.**

Nous nous situons évidemment au niveau du deuxième cas puisque le site serveur doit accepter tout genre de requête SQL en provenance du ou des clients. Il ne connaît pas à l'avance la requête qu'il va exécuter. Il est donc impossible de figer les requêtes une fois pour toutes dans le programme.

Nous présentons dans les pages qui suivent, les transformations nécessaires à la communication entre deux machines lorsque l'outil RPC est employé.

## 3.2. Transformations nécessaires à la communication

Nous avons introduit, dans un chapitre antérieur, que RPC imposait certaines contraintes. Voyons pourquoi.

- Une première obligation, qui peut paraître évidente puisqu'on passe d'une machine à une autre, est le passage des **paramètres par valeur.**

- RPC génère dans le "stub serveur", un suffixe aux fonctions déclarées dans l'interface de communication entre processus. Il rajoute **"_1"** au nom de la procédure.

- Du côté serveur, il faut travailler avec des **pointeurs** sur les paramètres reçus.

- RPC n'accepte qu'**un seul paramètre** en entrée par fonction, il faut donc toujours passer par l'intermédiaire d'une structure C, cela implique une gymnastique importante au niveau des types de données et des pointeurs

- Les **paramètres de retour** (réponses) doivent être déclarés en **static** et affectés dans le **type de la fonction.** Cela signifie que si l'on a au moins deux paramètres de retour, le type de la fonction est une structure, puisqu'un seul paramètre peut transiter sur le réseau via RPC.

Toutes ces transformations imposées par RPC se caractérisent par l'introduction d'un **agent client** et d'un **agent serveur** que nous avons mis en place pour notre prototype.

L'agent client est composé de la déclaration de toutes les procédures susceptibles d'être appelées à distance avec la même syntaxe que l'appel lui même, c'est à dire, même identifiant et même nombre de paramètres [HAJH88].

Chaque procédure de l'agent client transforme les paramètres en les intégrant dans une structure. Cette dernière contient les paramètres d'entrée de la procédure appelée. Les paramètres d'entrée et les paramètres de sortie sont aiguillés vers deux structures. L'une pour les paramètres d'entrée et l'autre pour les paramètres de sortie dans le cas où plus d'un champ résultat est attendu.

Chaque procédure de l'agent client appelle une procédure définie dans l'agent serveur.

L'agent serveur définit autant de procédures que l'agent client en déclare. Chaque procédure de l'agent serveur reconstitue l'appel de la fonction telle qu'elle a été formulée dans le programme d'application.

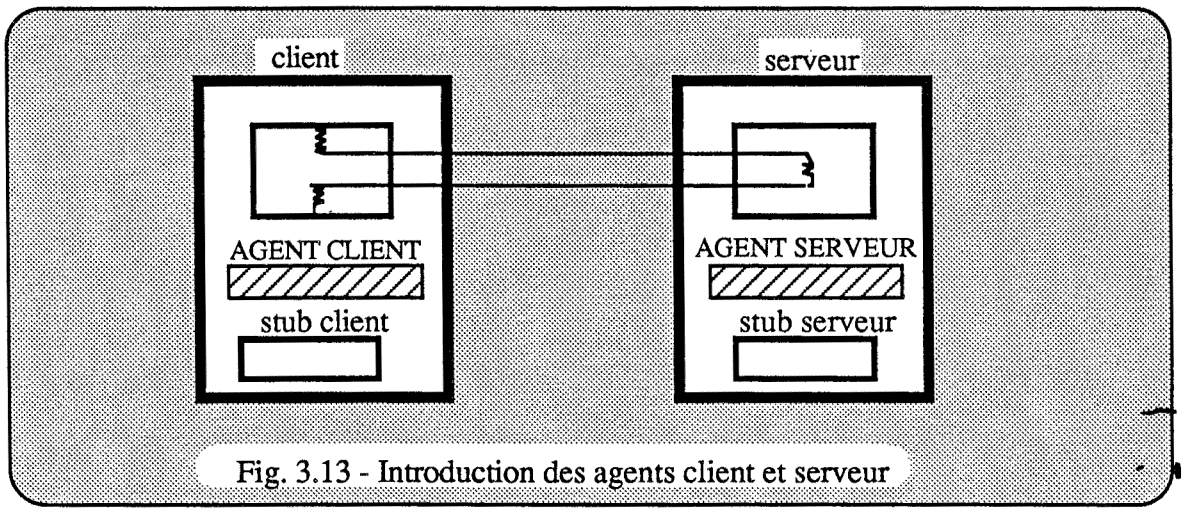



Fig. 3.13 - Introduction des agents client et serveur

exemple:

interface utilisateur

```
---
multiplication (opérande1, opérande2, résultat)
---
si résultat > valeur alorscommande      finsi
```

L'agent client doit contenir la déclaration de la procédure multiplication:

```
multiplication(wopd1, wopd2, résult)

calcul.opd1 = wopd1                    /* la structure calcul va contenir  les deux
calcul.opd2 = wopd2                    /*  opérandes : wopd1 et wopd2
résult = sv_mult_1 (&calcul,cl)        /* la procédure sv_mult est définie dans l'agent serveur */
```

Le résultat de l'opération s'obtient en appelant la procédure "sv_mult_1" définie chez l'agent serveur.

On constate deux paramètres dans l'appel de la procédure "sv_mult". Le premier est l'adresse de la structure "calcul" et le second correspond au lien créé entre le serveur et le client.

Du côté agent serveur

La procédure "sv_mult" est déclarée avec un seul paramètre en entrée (la structure des deux opérandes). Elle est de type entier car elle rend le résultat de la multiplication (on remarquera les pointeurs (*)):

```
int * sv_mult_1(operat)                          /* operat est une structure de deux nombres
int static résult                                  /* résultat obligatoirement  déclaré en sta
multiplication(operat->opd1, operat->opd2, résult)
- - -
return(&result)                  /* le type de la fonction étant un entier, elle retourne le résultat *
```

L'exemple du serveur de multiplication, disponible en Annexe 6, met en évidence les transformations exigées par RPC, mais pour des raisons de simplifications de tests,

l'interface utilisateur est intégrée dans l'agent client et le programme serveur dans l'agent serveur.

## 3.3. Choix de l'interface procédurale de communication avec Oracle

Le serveur nécessite une ***interface procédurale*** pour communiquer avec Oracle. Nous avons étudié trois schémas de réalisation, basés respectivement sur:

- le code HLI® d'Oracle pour programmer les fonctions du serveur,

- le précompilateur d'Oracle et le SQL dynamique,

- une interface de haut niveau pour Oracle appelée SOUPE© .

Nous avons conclu sur le choix du troisième schéma, pour des raisons en partie générales, en partie conjoncturelles comme nous le montrons ci-après.

### 3.3.1. Le code HLI (High Level Interface)

Une liste de fonctions est mise à la disposition du client lui permettant d'accéder à sa base de données Oracle à travers un programme C.

Les fonctions HLI d'Oracle sont proches du code généré par Oracle. Par conséquent, la programmation en est complexe. Il faut savoir que ce code HLI est susceptible d'être modifié avec les versions d'Oracle.

Voyons quand même, ce que donnerait l'application de ce choix avec l'exemple de la fonction qui permet de créer une connexion à Oracle représenté à la figure 3.14.

Le programme d'application appelle la procédure "cl_hello" définie dans la liste des fonctions disponibles. Celle-ci est déclarée chez le client. Elle transforme les paramètres si

---

® HLI: High Level Interface
© SOUPE: Simple Oracle User Programming Extension

cela est nécessaire (nous abordons ce sujet plus loin) avant d'effectuer un appel à distance de la procédure "r_hello" définie chez dans l'agent serveur.

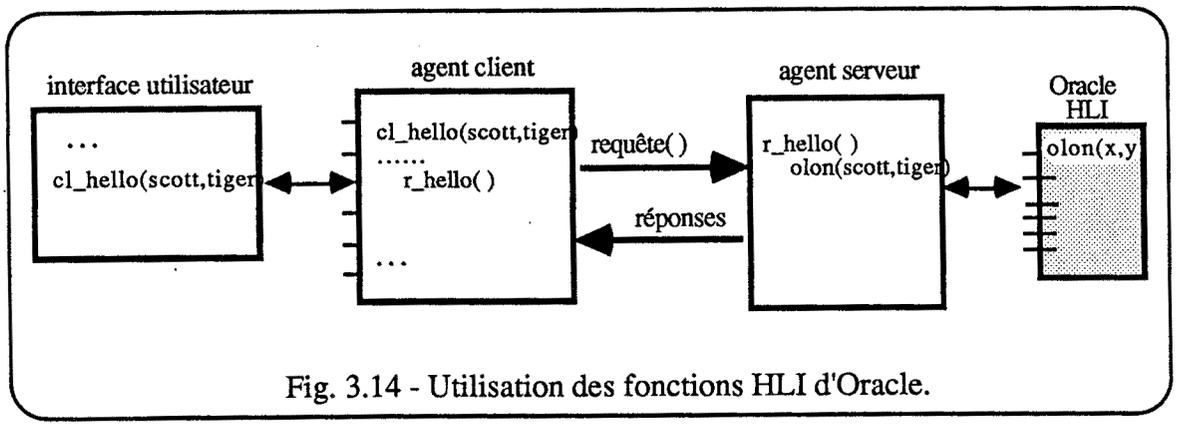



Fig. 3.14 - Utilisation des fonctions HLI d'Oracle.

La fonction "olon(user,password)" est une fonction HLI d'Oracle qui permet d'ouvrir une connexion pour l'utilisateur dont le nom et le mot de passe sont donnés en paramètres soit respectivement dans l'exemple de la figure 3.14: 'SCOTT' et 'TIGER'..

C'est seulement au niveau du serveur que nous utilisons une interface procédurale de communication avec Oracle. Cette solution présente des inconvénients :

- l'agent serveur risque de devoir être maintenu à chaque changement de version d'Oracle [ORA2],
- le programmeur doit se familiariser avec une nouvelle norme de programmation pour accéder à sa base de données.
- la mise au point des agents client et serveur demande une lourde programmation difficile à maintenir. Le nombre des paramètres des fonctions HLI est important et il faut tenir compte des contraintes de RPC à ce sujet (on ne peut faire transiter qu'un seul paramètre sur le réseau ).

D'autres solutions, toujours avec des fonctions HLI, ont été discutées, mais sans succès car elles présentaient encore plus d'inconvénients.

Une des solutions demandait au programmeur d'application de connaître les fonctions HLI d'Oracle et de programmer comme s'il était en site local. Le rôle des agents aurait été alors bien plus complexe. En effet, l'agent client aurait du transformer les fonctions appelées par les programmes d'applications en fonctions "transportables" puis l'agent serveur les aurait retransformées en format HLI. Tous les paramètres devant suivre, cette solution n'était pas envisageable.

Ce n'est qu'après cette étude, et la constatation de tous les inconvénients que pose une programmation utilisant les fonctions HLI, que nous nous sommes penchés vers une autre stratégie proposée par Oracle : PRO*C.

### 3.3.2. PRO*C

PRO*C est le précompilateur du langage C pour Oracle.

En ce qui concerne le programmeur d'application, la situation est similaire au cas précédent. En effet, celui-ci dispose également d'une liste de fonctions pour interroger la base de données. La différence avec le cas HLI réside dans le fait que les fonctions du serveur exécutent des ordres SQL.

**Exemple**:

Pour la connexion à Oracle, la fonction de communication définie côté serveur et appelée "r–hello" utiliserait l'ordre SQL suivant pour se connecter à Oracle:

```
EXEC SQL CONNECT :uid IDENTIFIED BY :pwd
```

où uid et pwd sont des variables contenant respectivement le nom de l'utilisateur Oracle et son mot de passe.

Nous avons montré précédemment la nécessité de programmer en SQL dynamique. La documentation Oracle [ORA2] donne un exemple complet de programme écrit en C avec du SQL dynamique. Son rôle est celui d'un mini UFI. Il accepte n'importe quel ordre SQL effectué sur les tables de l'utilisateur connecté sous Oracle. Nous l'avons fait fonctionner localement sur le SPS9 en Version 5 d'Oracle.

Les ordres principaux utilisés pour la programmation en SQL dynamique sont :

| | |
|---|---|
| - PREPARE requête | - DECLARE curseur |
| - DESCRIBE requête | - FETCH |

C'est à ce niveau qu'intervient le précompilateur. Il transforme les ordres SQL en C avant l'étape du compilateur lui même.

Nous nous sommes inspirés de l'exemple d'Oracle en SQL dynamique et de la documentation DB2 [DB287] pour mettre en place nos tests localement. Ceux-ci terminés, il ne restait plus qu'à les tester à distance, c'est à dire en liant les modules RPC avec ceux d'Oracle.

Malheureusement pour nous, la version 5 d'Oracle sur matériel BULL est une version en "Beta Test", le précompilateur du SGBD nécessite une option particulière (-FS) lors de la compilation. Celle-ci perturbe l'ordre des priorités de recherche dans les bibliothèques en ligne. Il s'agit d'ailleurs d'un choix discutable des réalisateurs d'Oracle, même pour une "Beta-Test", car l'option "-FS" implique le recours à un compilateur C non standard.

Tout cela signifie que si l'on veut assembler les modules Oracle et ceux de RPC, l'étape du précompilateur doit être supprimée.

Une autre solution consisterait à écrire les fonctions sans faire appel à RPC, c'est à dire, en écrivant toutes les lignes nécessaires à la communication entre processus via les sockets. Cela représente un travail très important. Malgré tout, nous l'avons testé avec une fonction simple et ça fonctionne très bien.

Mais il existe une autre interface qui fait appel à un outil développé dans l'antenne Grenobloise du Centre de Recherche de BULL (lié à un laboratoire de l'IMAG par une convention et localisé dans les locaux-mêmes de l'IMAG). Cet outil est SOUPE (Simple Oracle User Programming Extension).

### 3.3.3. L'interface ORACLE "SOUPE"

SOUPE est une interface Oracle de haut niveau pour les langages C et Pascal [SOUP88]. Ce logiciel constitue une interface de programmation pour Oracle réunissant les avantages du précompilateur (expression des requêtes SQL en clair) et de HLI (élimination de la phase de précompilation).

L'avantage du choix SOUPE réside dans le fait que les fonctions sont déjà existantes.

Voyons un exemple de programmation d'une requête intégrée dans un programme C avec la fonction SOUPE "sql":

```
sql("select NOM,SALAIRE from EMPLOYES where SALAIRE < %i", 7000)
```

Le premier paramètre d'appel est la commande SQL à exécuter. Cette chaîne de caractères contient une chaîne "%i" qui précise à SOUPE qu'un paramètre doit être substitué avant l'exécution de la requête par Oracle (le "%" indique un paramètre d'entrée). Dans ce cas, la valeur "7000", de type entier - le "i" qui suit le "%" - se substitue en entrée au "%i". La requête envoyée à Oracle est donc :

```
"select NOM,SALAIRE from EMPLOYES where SALAIRE < 7000"
```

Le schéma de communication intègre alors l'interface SOUPE comme le montre la figure 3.15.

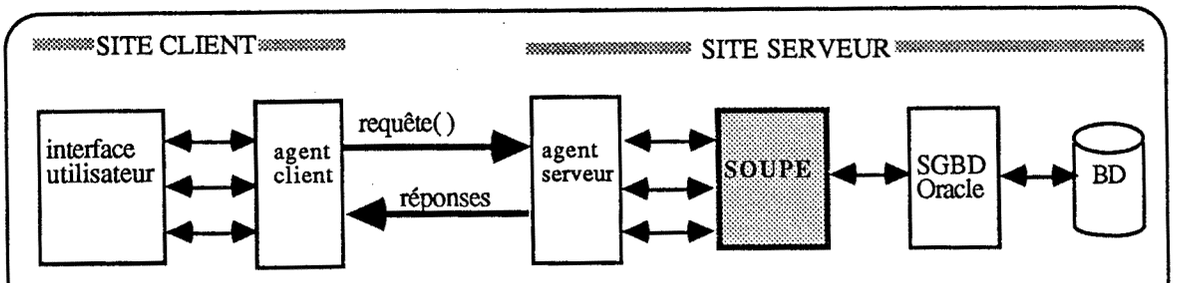



Fig. 3.15 - Schéma de communication avec l'interface SOUPE.

En ce qui concerne l'interface utilisateur, deux solutions se présentent :

- soit nous donnons une nouvelle liste de fonctions au programmeur qu'il devra utiliser pour accéder à sa base de données comme dans les deux cas précédents (HLI et PRO*C),

- soit le programmeur utilise les fonctions SOUPE telles qu'elles existent actuellement.

La cible étant de ne pas perturber le programmeur et les éventuelles applications déjà existantes, nous avons choisi la deuxième solution qui ne demande pas d'adaptation nouvelle à une autre syntaxe.

Chacune des fonctions SOUPE disponible pour un programmeur local est redéfinie dans les agents clients et serveur pour effectuer les transformations nécessaires à la communication par RPC.

Le paragraphe suivant présente le prototype mis en œuvre.

## 3.4. Réalisation du prototype

Le programme d'application qui représente l'interface utilisateur peut appeler toutes les fonctions de SOUPE comme si le programme était local.

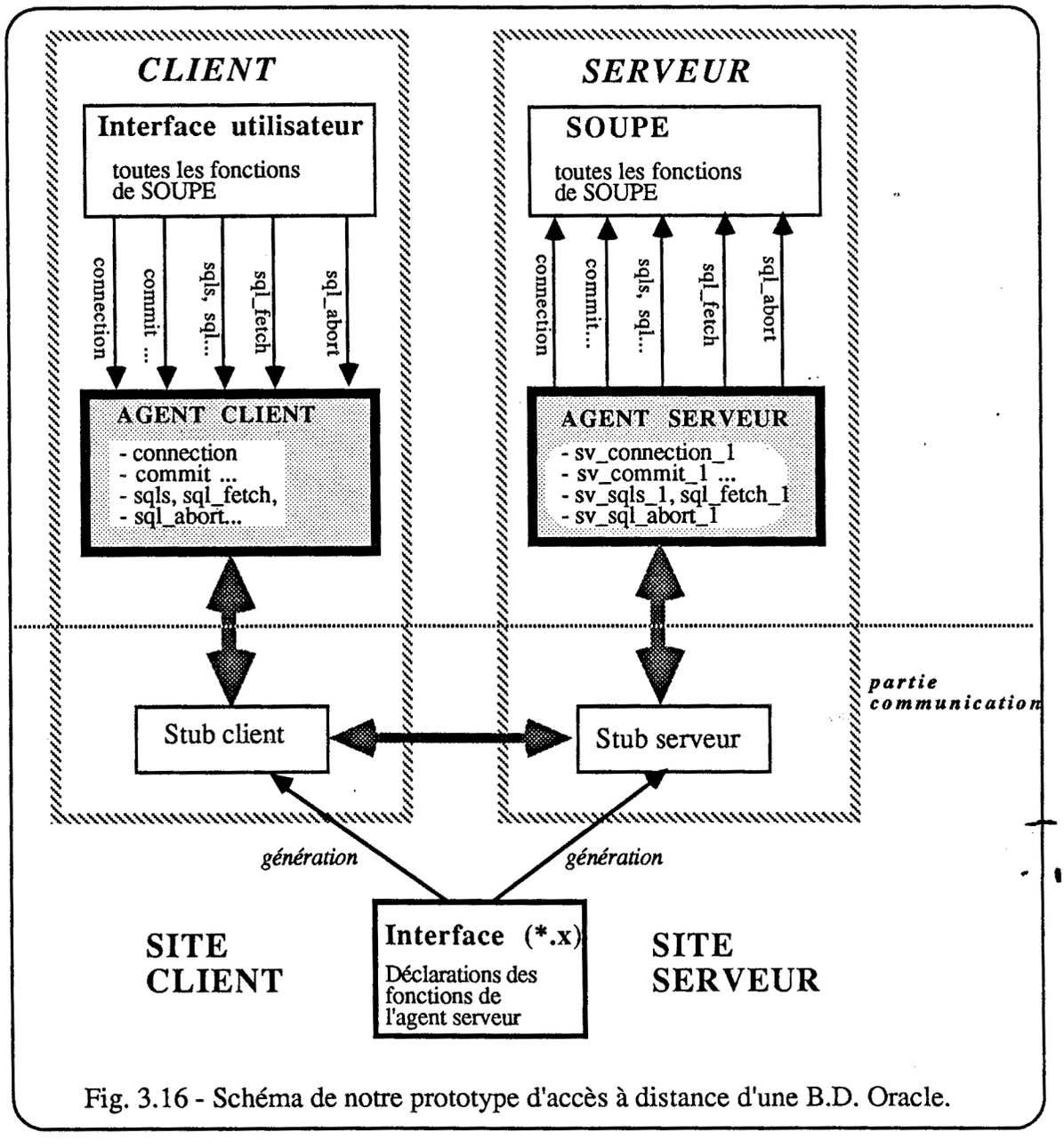



Fig. 3.16 - Schéma de notre prototype d'accès à distance d'une B.D. Oracle.

Les agents client et serveur se chargent des transformations préalables à la communication via le réseau. Les stubs client et serveur, générés par l'outil RPCGEN, sont chargés de la communication entre les deux processus.

Nous remarquons sur la figure 3.16, la composition du client et du serveur. Le programme d'application, l'agent client et le stub client forment l'ensemble du client. Symétriquement, SOUPE, l'agent serveur et le stub serveur représentent le serveur.

La liste des fonctions SOUPE dont disposait les programmeurs jusqu'à maintenant est sensiblement modifiée dans la mesure où deux fonctions disparaissent, *sqlt* et *sqlti* qui étaient prévues pour le cas où le langage de la base de données n'était pas SQL. Elles appelaient un module externe de transcodage vers SQL. Mais aucun module de ce type n'a été réalisé.

Le programmeur dispose donc des fonctions suivantes pour interroger sa base de données :

- sql : substitution des paramètres en entrée et affichage des réponses par SOUPE.

- sqlo : requête immédiate (sans fonction, ni substitution de paramètre).

- sqls : substitution des paramètres en entrée.

- sqli : pour les programmes écrits en langage PASCAL car la syntaxe est différente.

- sql_fetch : lecture d'un tuple, doit être précédé d'un appel de fonction sqls ou sqlb dans le programme d'application.

Un module supplémentaire sera ajouté au niveau de l'agent client afin de donner au programme distant toutes les fonctionalités de SOUPE existantes sur le site local actuellement. Ce point est traité dans un autre mémoire [PAJO89].

Supposons qu'un programme d'application veuille se connecter à Oracle sous le nom d'utilisateur "SCOTT" avec le mot de passe "TIGER". D'après la documentation SOUPE [SOUP88], la syntaxe est la suivante:

**Pour le programme d'application :**

```
if (connection(1,"SCOTT","TIGER") == 0)
{
 printf("nom ou mot de passe incorrect \n");
 exit(1);
}
```

L'agent client dans lequel sont définies toutes les fonctions telles qu'elles sont appelées dans le programme d'application, effectuent des transformations avant d'appeler les fonctions déclarées chez l'agent serveur. Mais nous avons vu auparavant que l'interface RPC est à définir en premier pour y déclarer les structures de communication lorsqu'une fonction a plus d'un paramètre. C'est le cas de l' exemple où nous présentons la fonction "connexion" qui permet d'ouvrir une connexion sous Oracle pour un utilisateur donné;

**Exemple de l'interface RPC appelé "exemple.x":**

```
struct s_hello {
  int no;
  string uid<20>;
  string pwd<20>;
};
program EXPROG1 {
  version EXVERS
   int CONNECTION(s_hello) = 1;
  } = 1;
} = 94;
```

Ici nous ne déclarons qu'une seule fonction disponible chez le serveur; la fonction CONNECTION. Il est évident que ce n'est qu'un exemple car cette fonction seule ne servirait à rien.

Les agents vont se servir du module de déclarations communes créé par RPCGEN, c'est le fichier "exemple.h".

**L'agent client** appelé **exemplecl.c:**

```
≠include <rpc/rpc.h>
≠include <stdio.h>
≠include "exemple.h"                                    /* déclarations communes créées par rpcgen

int connection(wno,wuid,wpwd)
int wno;
char *uid;
char *wpwd;
{
 struct s_hello whello;                                  /* strcuture des paramètres d'entrée
 CLIENT *cl;                                              /* déclaration du pointeur de communica
 char *server;
 server = "castor"                                       /* si le serveur tourne sur CASTOR
 whello.no = wno;
 whello.uid = wuid;
 whello.pwd = wpwd;
 cl = clnt_create(server, EXPROG1, EXVERS1, "tcp");      /* création du lien de communication  */
 if (cl == NULL)
 {
  clnt_pcreateerror(server);
  exit(1);
 }
 result = connection_1(&whello,cl);                      /* adresse de la structure
 if (result == NULL)
 {
  clnt_perror(cl,server);
  exit(1);
 }
 if (*result == 0)
 {
  fprintf(stderr,"%s ne peut pas établir la connexion \n", server);
  exit(1);
 }
}
```

Dans cet exemple, les principaux rôles de l'agent client sont :

- l'affectation des paramètres d'entrée dans les champs de la structure de communication appelée ici "whello",

- la création d'un lien avec le serveur (clnt_create),

- l'appel et l'exécution de la fonction "connection_1" avec l'adresse de la structure de communication en paramètre ainsi que le pointeur client nécessaire au serveur pour renvoyer le résultat de la fonction au bon client.

Voyons maintenant le **côté serveur** :

Dans l'agent serveur, les noms de fonctions sont forcément suffixés par "**_1**" et la fonction rend un pointeur sur un type de données (ici, un entier qui précise le bon ou le mauvais déroulement de la connexion). Si nous avions eu plusieurs paramètres en retour, le type de la fonction aurait été une structure de champs.

```
≠include <rpc/rpc.h>
≠include <stdio.h>
≠include <sys/dir.h>
≠include "exemple.h"                    /* déclarations communes créées par rpcgen      */
≠include "soupe.h"

int *connection_1(logon)
struct s_hello  *logon;                 /* déclaration pointeur sur structure de type s_hello */
{
 static int result;                     /* doit être déclaré en static   */
 if (connection(1, logon->uid, logon->pwd) == 0) /*appel identi à celui de l'interface utilisateur */
 {
  result = 1;
  return(&result);
 }
 result = 0;
 return(&result);
}
```

Le paramètre de la procédure connection_1 est appelé "logon". Il est déclaré comme un pointeur sur une structure de type "s_hello". Ce type de structure a été défini dans l'interface RPC identifié ici par "exemple.x" et sa déclaration se trouve dans le fichier "exemple.h" généré par *rpcgen*.

La fonction "connection" appelée dans la fonction "connection_1" de l'agent serveur suit la même syntaxe que dans le programme d'application du client.

Voilà toute la démarche nécessaire pour rendre invisibles au programme d'application, les étapes de la communication entre processus, l'appel de procédure à distance.

La mise en œuvre de notre prototype permet la communication entre un client et un serveur. Nous étudions dans le paragraphe suivant les problèmes et solutions envisagées pour une communication à plusieurs clients.

## 3.5. Difficultés rencontrées

La seule structure d'organisation étant celle qui est représentée à la figure 3.10, à savoir n clients et n+1 serveurs, les lignes suivantes se rapportent à ce type de schéma.

Chaque serveur actif est représenté, dans la machine, par un numéro, c'est le numéro de programme figé dans l'interface RPC avant la génération RPCGEN [exemple en Annexe 6]. Deux serveurs de même numéro, ne peuvent être actifs en même temps sur la même machine.

Une solution consisterait à pouvoir modifier dynamiquement ce numéro, dans le démon, à chaque duplication d'un serveur unitaire. Mais RPC ne nous permet pas de mettre en place une telle structure.

Une autre solution semble envisageable sans trop d'inconnues. En limitant le nombre de serveurs actifs et le nombre des clients, le démon serveur devra gérer les correspondances entre client et serveur. Le démon devra être capable de détecter la fin d'activité d'un client, de repérer le numéro du serveur pour le tuer et le réaffecter à un autre client en attente.

A la fin de la rédaction de ce mémoire, les tests en cours montrent qu'il est tout à fait possible de réaliser ce type de structure en employant l'instruction "fork" pour générer des serveurs "fils". Le nombre d'utilisateurs reste toutefois un paramètre à initialiser.

# CONCLUSION

Deux domaines ont été étudiés au cours de ce mémoire, à savoir le domaine des bases de données relationnelles et la communication entre processus sur un réseau hétérogène.

La première phase du travail concerne l'étude et la création de la base de données IMAG. L'importance du temps de réalisation de cette base n'avait pas été estimé. Mes connaissances sur les bases de données relationnelles acquises lors de mon précédent emploi ont permis, non pas de progresser plus vite, mais de progresser avec méthodologie. Cela nous a amené à établir une codification de l'environnement Oracle sous Unix. Cette première étape n'était pas prévue dans le sujet initial de ce mémoire, elle a néanmoins permis d'acquérir une connaissance approfondie du SGBD Oracle et de ses outils tels que PRO*SQL, ODL, SQL*PLUS et PRO*C.

La seconde phase du travail étudie le domaine des interrogations de base de données relationnelles à travers un réseau hétérogène avec tous les problèmes de communication que cela engendre, notamment au niveau de la représentation des données.

L'interface de communication avec Oracle retenue (SOUPE) est une solution du à la version Béta-Test du SGBD installée sur l'ordinateur utilisé. En effet, les tests effectués ont prouvé que l'usage du précompilateur Oracle était tout à fait envisageable puisque les options bloquantes s'avèrent ne pas exister dans les versions officielles d'Oracle.

Un des points très positif de ce travail concerne la transparence donnée à l'utilisateur au niveau de la localisation de la base de données. En effet, ce dernier peut consulter à partir de n'importe quel système Unix du réseau sans savoir où se trouve la base de données. Le logiciel écrit permet l'établissement d'un dialogue entre un ordinateur client et l'ordinateur serveur. Les programmes clients connaissent le nom du serveur avec qui ils vont communiquer.

L'emploi d'un mécanisme d'appel de procédure distante nous a permis de constater la simplification apportée dans l'écriture d'applications distribuées. Nous avons également découvert la difficulté que représente les tests de mise au point de programmes incluant des communications inter-processus notamment lorsque les versions du mécanisme d'appel de procédures à distance sont différentes d'une machine à l'autre.

L'interface homme/machine reste à développer à l'aide du logiciel SOUPE. Dans un souci de convivialité, un enchaînement de menus-écrans permettront à l'utilisateur de consulter ou de modifier les informations de la base de données. Une gestion des autorisations peut être envisagée et facilement réalisée avec le SGBD Oracle

Cet environnement dans lequel j'ai navigué durant une année m'était totalement inconnu auparavant. Malgré un isolement géographique l'étude de ce projet constitue une expérience positive au niveau de la conception et de la réalisation d'une application distribuée, et aussi au niveau de l'utilisation des services de communication.

Ce travail nous a apporté une connaissance du langage C, du SGBD Oracle et du système d'exploitation Unix, notamment de la gestion des processus et du fonctionnement de la communication entre processus sous Unix. L'apprentissage des réseaux a été une découverte sur le plan technique. L'étude du modèle ISO et des protocoles TCP/IP a enrichi nos connaissances.

Une étude complémentaire portant sur la réalisation d'un serveur d'informations bibliographiques fait actuellement l'objet du mémoire CNAM de MC. PAJON [PAJO89].

# ANNEXES

# ANNEXE 1 - Organisation d'une base de données Oracle

Une base de données Oracle est constituée de partitions. Il y en a au moins une, la partition système, qui contient :

- les tables du dictionnaire de données,
- les tables temporaires,
- les tables d'aide en ligne.

Le premier fichier de la partition système est le fichier appelé INIT.ORA qui contient les paramètres de la base de données.
Une partition est à son tour, découpée en "space definition".

Un "space definition" permet le contrôle de l'espace utilisé pour les parties données et index de la ou des tables qu'il va contenir. Ces deux espaces sont appelés segments. La création d'un "space definition" par soi-même est indépendant de la création de la table. Ce n'est que dans l'ordre de création de la table que celui-ci sera nommé.

Le "space definition" n'alloue aucun espace. L'espace est alloué lorsqu'une table est crée, en accordant au "space definition" de contrôler la table. Des formules sont données dans la documentation Oracle [ORA1] pour calculer le nombre de segments données et le nombre de segments index d'une table.

Une table peut contenir jusqu'à 254 colonnes et chaque colonne est spécifiée comme un des types de données suivants:

- CHAR (maximum 240 caractères)
- NUMBER
- DATE (siècle, année, mois, jour, heure, minutes, secondes)
- LONG (maximum 65535 caractères)
- RAW ou LONG RAW

Malgré sa grande utilisation, quelques contraintes du type LONG sont à noter, par exemple :

- une seule colonne de type LONG par table,

- les colonnes de type LONG ne peuvent pas être indexées,
- les colonnes de type LONG ne peuvent pas être référencées dans WHERE, GROUP BY, CONNECT BY ou dans la clause DISTINCT.

Chaque table, chaque colonne créée, chaque opération entraîne une mise à jour directe du dictionnaire de données.

Le dictionnaire de données est un ensemble de tables et de vues automatiquement générées et mises à jour par Oracle. Il est capable de donner l'identification de tous les utilisateurs d'Oracle, le nom des objets de la base avec leur propriétaire, l'espace qui a été alloué, les droits et privilèges qui ont été donnés.

La manipulation et le traitement des données d'une base Oracle peuvent être appuyés par l'un ou plusieurs des produits proposés par Oracle [ORA1]:

- **SQL*PLUS:** permet d'interroger les tables Oracle interactivement (remplace le UFI de la version 4 d'Oracle).
- **SQL*FORMS:** formateur d'états plein écran, permet de mettre à jour les tables Oracle en utilisant des formes prédéfinies par l'utilisateur.
- **SQL*REPORT:** permet de fusionner et formater du texte avec les données d'Oracle.
- **SQL*NET:** permet d'accéder aux bases de données à partir d'ordinateurs autres que là où se trouvent les bases de données.
- **ODL:** Oracle Data Loader, utilitaire de chargement des tables Oracle.
- **CCF:** Create Contiguous File, permet la création des fichiers Oracle, utilisé pour allouer ou nettoyer les blocs de la base de données.
- **PRO*SQL:** fonctions HLI de la Version 5 d'Oracle.
- **PRO*C:** précompilateur C.

Grâce à ce précompilateur, on peut faire également du SQL dynamique; c'est à dire ne pas préprogrammer l'ordre SQL à exécuter.

***Exemple:***

L'ordre SQL est donné par l'utilisateur lors de l'exécution du programme, il n'a donc pas été transformé en C lors de la précompilation. PRO*C prévoit des ordres propres à ce type de programmation.

D'autres produits existent mais ne sont pas évoqués ici, tels que **SQL*Calc**, **SQL*Menu**, **SQL*Graph**.

# ANNEXE 2 - Exemple d'un fichier de création de table

Le fichier crtequipe.sql

```
drop index equipei1;
drop table equipe;
drop space definition sp_equipe;
commit;

create space definition sp_equipe
datapages     (initial 5,
               increment 25,
               maxextents 300,
               pctfree 20)
indexpages    (initial 5,
               increment 25,
               maxextents 100)
partition system;

create table equipe     (idlabo char(2) not null, idequip char(2) not null,
                         lbeqip char (80),
                         idresp char(3)
                         )
space sp_equipe;

create unique index equipei1 on equipe (idlabo, idequip);

grant select on equipe to public;
```

# ANNEXE 3 - Formulaire de saisie

Un formulaire de saisie a été envoyé à chaque équipe de l'IMAG pour rassembler les informations nécessaires à la création de la base de données.

Ainsi, pour une équipe donnée, nous regroupons les renseignements suivants:

- Sigle Labo
- Libellé équipe
- Liste des associations thème principal / sous-thème auxquelles s'intéresse l'équipe sachant qu'un sous-thème est rattaché au moins à un thème (une liste des thèmes et sous-thèmes a été proposée).
- Pour chaque personne travaillant dans l'équipe :
  Nom,
  Prénom,
  Statut (Chercheur,Enseignant-chercheur,Thésard,Ingénieur) ,
  Sigle des projets auxquels participe la personne.
- Pour chaque projet présenté :
  Sigle du projet Ex : SPECTRE
  Libellé du projet Ex : Spécification Temps Réel
  Thèmes auquels se rattache le projet
  Sous-thèmes auxquels se rattache le projet
  Soutien financier (C3,Esprit,Cnet etc...)
  OST correspondants
  Nom du responsable (un ou plusieurs)
  Contenu - présentation ( maximum 2 pages)

Nous avons indiqué à tous les chefs d'équipe concernés qu'un fichier type était disponible sur le Vax dans le but d'être rempli et renvoyé sur le Vax.

Le squelette de ce fichier formulaire est décrit ci-dessous.

SIGLE EQUIPE : _____
LIBELLE EQUIPE : _____

LISTE DES RUBRIQUES DE L'EQUIPE :

Theme : _____
Sous-theme : _____
Sous-theme : _____

Theme : _____
Sous-theme : _____
Sous-theme : _____

etc...

LISTE DES PERSONNES TRAVAILLANT DANS L'EQUIPE :

Nom : _____
Prenom : _____
Statut : _
c: chercheur
e: enseign-cherch
t: thesard
i: ingenieur
Projet 1 (sigle) : _____
Projet 1 (sigle) : _____
Projet 3 (sigle) : _____

Nom : _____
Prenom : _____
Statut : _
c: chercheur
e: enseign-cherch
t: thesard
i: ingenieur
Projet 1 (sigle) : _____
Projet 1 (sigle) : _____
Projet 3 (sigle) : _____

etc...

PRESENTATION DES PROJETS DE L'EQUIPE :

Sigle du projet 1 : _____
Libelle du projet 1 : _____

Theme 1 auquel se
rattache le projet : _____

Sous-theme 1 auquel
se rattache le projet : _____

Sous-theme 2 auquel
se rattache le projet : _____

Theme 2 auquel se
rattache le projet : _____

Sous-theme 1 auquel
se rattache le projet : _____

Sous-theme 2 auquel
se rattache le projet : _____

etc. ..

Soutien financier : _____
(C3,Esprit,Cnet etc.) _____
_____

OST correspondants : ____
(218,214 etc.) ____
____

Nom responsable proj.: _____
Contenu (maxi 1 page 1/2 à 2 pages ) :

_____
_____
_____ RAPPORT D'ACTIVITE
_____
_____ DU PROJET
_____
_____
_

≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈≈

Sigle du projet 2 : _____
Libelle du projet 2 : _____

Theme 1 auquel se
rattache le projet : _____

Sous-theme 1 auquel
se rattache le projet : _____

etc...

Sigle du projet 3 : _____
Libelle du projet 3 : _____

etc...

# ANNEXE 4 - Les protocoles TCP/IP: routage et fragmentation

IP assume qu'un système est connecté à un réseau local. Le système peut envoyer des datagrammes à un autre système sur son propre réseau. Un problème se pose lorsqu'on demande à un système d'envoyer un datagramme à un autre système sur un réseau différent. Ce problème est résolu par les passerelles.

Une ***passerelle*** est un système qui connecte un réseau avec un ou plusieurs autres réseaux. Les passerelles sont soit des ordinateurs non dédiés ayant plusieurs contrôleurs de communication, soit des dispositifs spécialisés.

Le ***routage*** dans IP, est basé entièrement sur le numéro de réseau des adresses destination. Chaque ordinateur a une table des numéros de réseau. Pour chaque numéro de réseau, une passerelle est indiquée; c'est la passerelle à emprunter pour aller sur ce réseau. Afin que deux machines de réseaux différents puissent communiquer, on passe par une ou plusieurs passerelles comme le montre la figure A4.1.

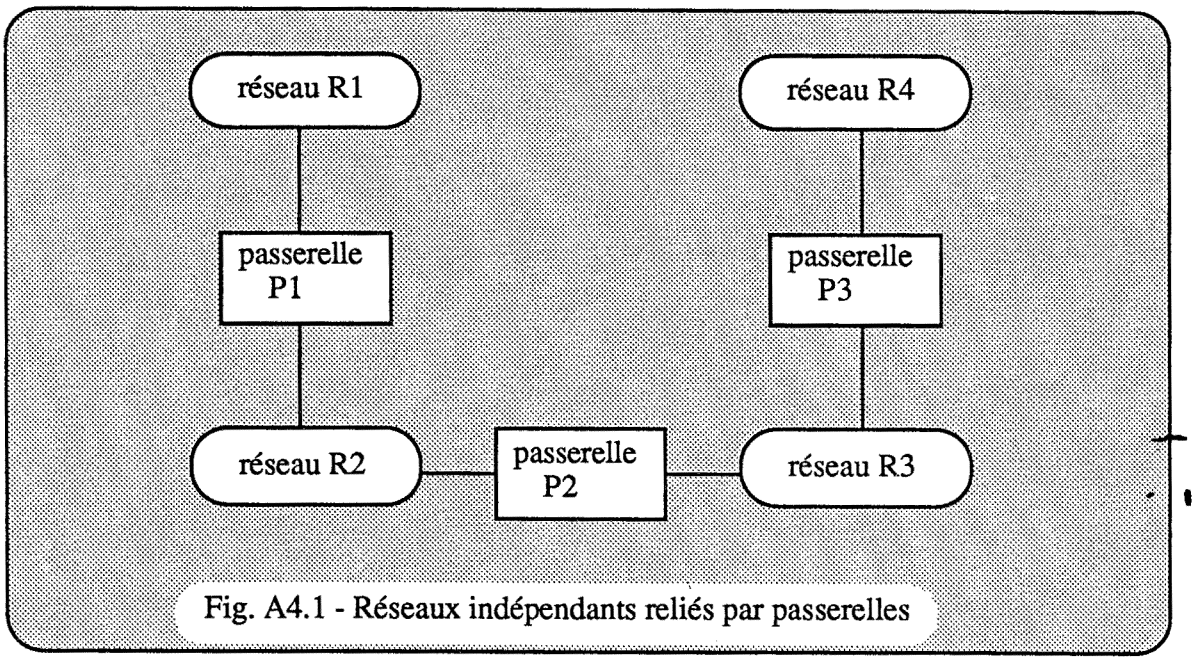



Fig. A4.1 - Réseaux indépendants reliés par passerelles

Grâce à une table de routage, on saura que :

- pour aller du réseau R1 au réseau R4, il faut envoyer à la passerelle P1,

- P1 va recevoir le paquet d'informations de R1 et l'envoyer à R2,
- etc ...

Le ***routage*** détermine la machine passerelle qui connait un chemin vers la machine destination lorsque le chemin n'est pas direct. La longueur du chemin correspond au nombre de passerelles nécessaires pour atteindre le réseau final.

Le ***routage*** s'effectue dans les couches basses de TCP/IP et reste invisible pour les utilisateurs. En effet, le routage se fait de proche en proche et au niveau de IP. Si IP détecte que le paquet est pour lui, il le passe à TCP/UDP, sinon, il fait la même chose que R1 à partir de sa table de routage, à savoir, l'envoi du paquet sur la passerelle ou le réseau suivant qu'il faut emprunter pour aller jusqu'au réseau R4.

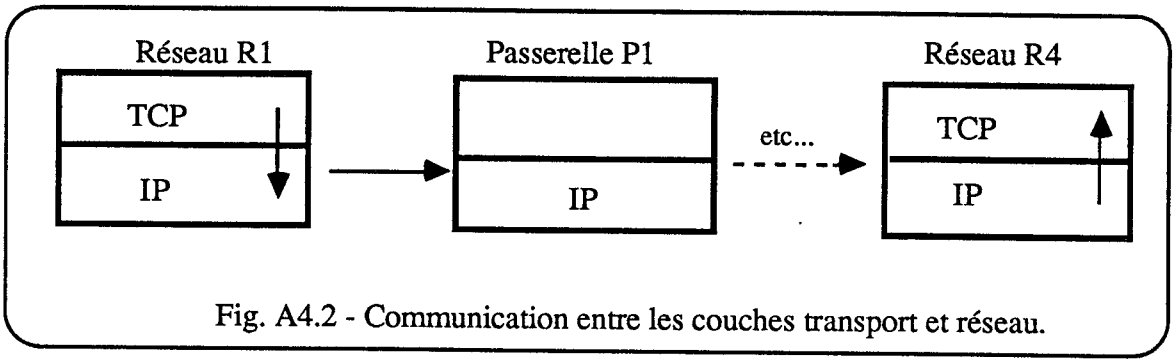



Fig. A4.2 - Communication entre les couches transport et réseau.

Dans les machines intermédiaires, on ne passe que par IP. Ce n'est que dans la machine destinataire que TCP recevra le paquet.

La ***fragmentation*** est la phase qui suit le routage. Au départ de la machine émettrice et de chaque passerelle, les segments TCP et UDP sont découpés en fragments IP de taille maximale égale à l'unité maximum de transmission (MTU) de l'interface choisie par le routage.

IP ne peut pas manier des paquets de taille importante. Pour cette raison, on éclate les datagrammes en morceaux; on parle alors de fragmentation. L'entête IP contient des champs indiquant qu'un datagramme a été fragmenté, et des informations pour que les morceaux soient remis ensemble.

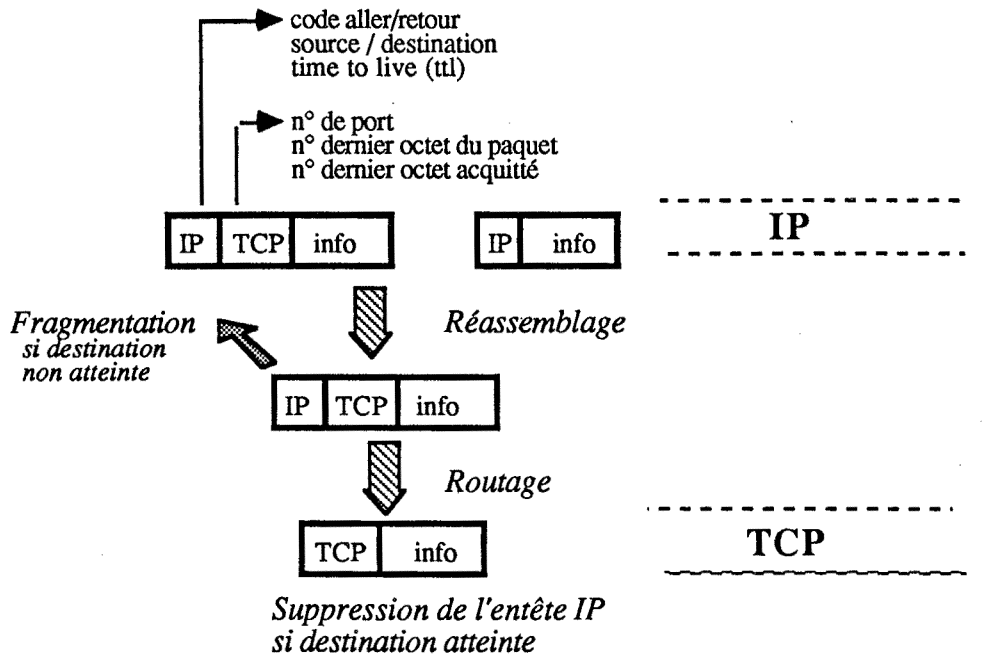



Fig. A4.3 - Arrivée d'un fragment sur une machine

A l'arrivée sur chaque passerelle et sur la machine destinataire, IP réassemble les fragments pour recomposer le segment initial. Les fragments d'un segment incomplet sont détruits après expiration d'un "timeout" spécifié dans l'entête de chaque fragment (le ttl : Time To Live), c'est le temps maximum que doit attendre un fragment en phase de réassemblage. Ce délai passé, IP détruit le fragment. Ainsi, IP ne conserve jamais de fragment parasite.

Si un fragment est perdu, TCP reémet le segment entier (alors que UDP ne reémet pas). Chaque segment contient l'adresse source et destination.

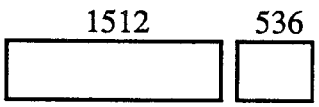
Voyons à l'aide d'un exemple, la fragmentation et le réassemblage d'un segment.

**Exemple:**

Supposons que la taille du segment à transférer est de 2048 octets. Le segment reçu sera fragmenté pour être conforme à ce qu'attend le média c'est ) dire, son unité de transfert maximum: MTU (Max Transit Unit).

Cette taille maximum diffère suivant le média :

| | | |
|---|---|---|
| trame Ethernet | = | 1512 octets |
| paquet X25 | = | 128 octets |
| trame LAS | = | 1006 octets |




*2048 octets en deux fragments*

Une unité de transport

IP va recevoir le premier fragment de 1512 octets puis l'autre de 536 octets.

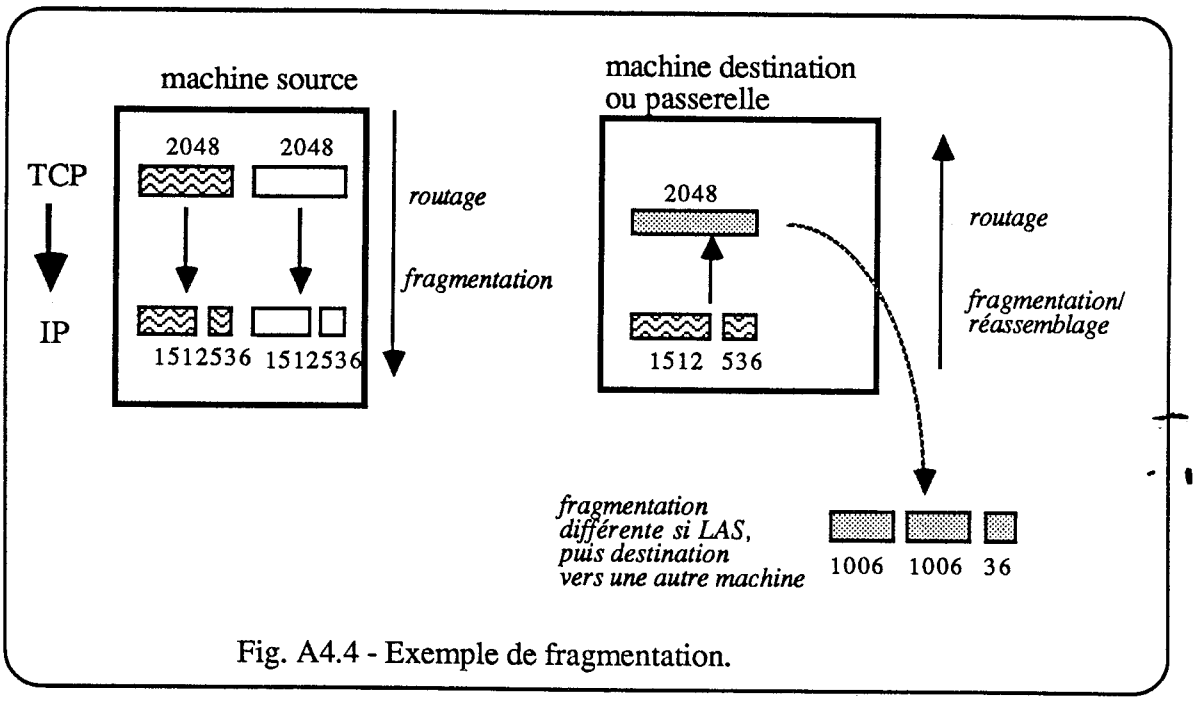



Fig. A4.4 - Exemple de fragmentation.

L'un des intérêts de IP est la faculté de s'adapter aux possibilités de tout média car, non seulement il fragmente un segment à la dimension des unités de transfert du média, mais aussi il réassemble les fragments d'un même segment arrivés dans n'importe quel ordre.

Si la machine est une passerelle, le routage est appliqué à nouveau sur chaque segment. Ce dernier est redécoupé en fragments en fonction du média suivant à utiliser.

# ANNEXE 5 - Serveur de multiplication via les "sockets"

```
/*****************************************************************/
/*                                                               */
/*     serveur de multiplication dans le monde INET              */
/*                                                               */
/*****************************************************************/
≠include <signal.h>
≠include <stdio.h>
≠include <sys/types.h>
≠include <netinet/in.h>
≠include <sys/socket.h>
≠include <sys/un.h>
≠include <netdb.h>

≠define NAME "/users/imag/sel/voboril/socket"
define ERROR -1
define TRUE 1
define FALSE 0

/*      fonctions externes      */
/*      ----------------        */
extern char *rindex();

/*      varaiables globales     */
/*      -------------------     */
int sock;
struct sockaddr_in name;
char buff[1024];

/*****************************************************************/
/*          cleanup    :    nettoyage sur fin de programme        */
/*****************************************************************/
cleanup()
{
  char mes[128];
  if (close(sock) == ERROR)
  {
  sprintf(mes,"Erreur sur close de sock");
  perror(mes);
  exit(8);
  }
}
/*****************************************************************/
/*          handler    :    traitement de signal                  */
/*****************************************************************/
handler()
{
  cleanup();
  exit(0);
}
/*****************************************************************/
/*          service    :    travail d'une transaction             */
/*****************************************************************/
```

```
service()
{
int new_sock, addrlen, nbytes, opd1, opd2;
struct sockaddr_in from;
char mes[128];

/*      phase accept      */
/*      ------------      */
addrlen = sizeof(strcut sockaddr_in);
new_sock = accept(sock, &from, &addrlen);
if (new_sock == ERROR)                          /*   traitement d'erreur    */
{
  sprintf("Erreur sur accept");
  perror(mes);
  exit(4);
}
printf("le serveur a accepté une connexion \n");    /*   trace              */

/*      phase de lecture        */
/*      ----------------  */
nbytes = read(new_sock, buf, 1024);

printf("le serveur a lu \n", buf);                  /*   trace              */
sscanf(buf,"%d %d", &opd1, &opd2);
sprintf(buf,"%d", opd1 * opd2);
printf("le serveur envoie %s \n",buf);              /*   trace              */
write(new_sock, buf,strlen(buf));

if (close(new_sock) == ERROR)                   /*   traitement d'erreur    */
{
  sprintf(mes,"Erreur sur close de new_sock");
  perror(mes);
  cleanup();
  exit(7);
}
}
/*************************************************************************/
/*              main :     programme principal                           */
/*************************************************************************/
main(argc,argv)
int argc;
char *argv[];
{
int namelen, nbytes, fini, i, type_lec;
struct servent *sp;
char mes[128];

sock = socket(AF_INET, SOCK_STREAM, 0);         /*   primitive socket     */

if (sock < 0)                                   /*   traitement d'erreur    */
{
  sprintf(mes,"Impossible d'ouvrir un socket");
  perror(mes);
  exit(1);
}
```

```
/*      phase de bind    */
/*      ------------     */
name.sin_family = AF_INET;
name.sin_addr.s_addr = INADDR_ANY;
sp = getservbyname("essai","tcp");
if (sp == NULL)
{
  printf("servcie inconnu: essai /n");
  exit(11);
}
name.sin_port = sp->s_port;
if (bind(sock, &name, sizeof(name)) == ERROR)        /*   primitive bind        */
{                                                    /*   traitement d'erreur   */
  sprintf(mes,"Impossible de 'binder' la socket");
  perror(mes);
  exit(2);
}

/*      cleanup en cas de 'kill'      */
/*      ---------------------- */
signal(SIGINT,handler);
signal(SIGTERM,handler);

/*      phase de listen                                                          */
/*      si on n'exécutait pas cette phase, 'accept' répondrait: 'invalid argument' */
/*      ------------------------------------------------------------------------ */
if (listen(sock,5) == ERROR)                         /*   primitive listen      */
{                                                    /*   traitement d'erreur   */
  sprintf(mes,"Erreur sur listen");
  perror(mes);
  exit(3);
}

do                                                   /* boucle d'exécution      */
  service();
while (TRUE);
}                                                    /*   fin du programme serveur */
```

```
/*************************************************************/
/*                                                           */
/*     Utilisation du serveur de multiplication dans le monde INET  */
/*                                                           */
/*************************************************************/
≠include <stdio.h>
≠include <sys/types.h>
≠include <netinet/in.h>
≠include <sys/socket.h>
≠include <netdb.h>

define ERROR -1

/*      fonctions externes      */
/*      ----------------        */
extern struct hostent *gethostbyname();
extern char *rindex();
/*************************************************************************/
/*              main :      programme principal                          */
/*************************************************************************/
main(argc,argv)
int argc;
char *argv[];
{
int sock, new_sock, addreln, nbytes, total, i, opd1, opd2;
struct sockaddr_in server;
struct hostent *hp;
char buf[1024];
strcut servent *sp;
char mes[128];

sock = socket(AF_INET, SOCK_STREAM, 0);              /*    primitive socket
  */
if (sock < 0)                                        /*    traitement d'erreur          */
{
  sprintf(mes,"le client: Impossible d'ouvrir un socket");
  perror(mes);
  exit(1);
}
/*      lecture des caractéristiques du host distant    */
/*      ---------------------------------------        */
hp = gethostbyname("castor");                        /* demande l'adr. de la machine "castor" */
if (hp == NULL)
{
  printf("Erreur sur gethostbyname");
  exit(2);
}

/*      initialisation de la sockaddr_in    */
/*      --------------------------          */
bzero ((char *) &server, sizeof(server));
server.in_family = hp->h_addrtype;
bcopy(hp->h_addr, &server.sin_addr, hp->h_length);

sp = getservbyname("essai");
```

```
if (sp == NULL)
{
  printf("service inconnu: essai \n");
  exit(11);
}
server.sin_port = sp->s_port;

/*      phase de connect       */
/*      ---------------- */
if (connect(sock, &server, sizeof(server)) == ERROR)        /*   primitive connect          */
{                                                           /*    traitement d'erreur       */
  sprintf("Impossible de se coinnecter sur la socket");
  perror(mes);
  exit(2);
}

/*      phase d'E/S      */
/*      ----------       */
printf("Entrez deux nombres:\n");
scanf("%d %d", &opd1, &opd2);                               /*   récupération de s nombres saisis
  */
sprintf(buf,"%d %d", opd1, opd2);
printf("le client envoie %s \n",buf);                       /*   trace                      */
if send(sock, buf, strlen(buf), 0) == ERROR)                /*   primitive send
  */
{                                                           /*   traitement d'erreur        */
  sprintf(mes,"Erreur sur le send");
  perror(mes);
  exit();
}

/*      réception et utilisation de la réponse   */
/*      ------------------------------------ */
nbytes = read(sock, buf, 1024);
buf[nbytes] = '\0';                                         /*   caractère de fin de chaîne */
printf("réponse: %s \n",buf);                               /*   affichage de la réponse
  */

/*      close:      fermeture de la socket       */
/*      ------------------------------           */
close(sock);
}                                                           /* fin du prog. client  multiplication */
```

# ANNEXE 6 - Serveur de multiplication via RPC

Nous trouvons dans cette annexe, le détail des modules intervenants dans RPC pour l'exemple du serveur de multiplication.

- mult.x : interface RPC du protocole de multiplication à définir,
- multsv.c : programme du serveur à définir,
- multcl.c : programme du client à définir,
- mult.h : déclarations communes créées par "rpcgen",
- mult_clnt_c : routines client créées par "rpcgen",
- mult_svc.c : squelette du serveur généré par "rpcgen",
- mult_xdr.c : routines de conversions nécessaires à la communication.

```
/*-------------------------------------------------------------------------------*/
/*      mult.x :    interface RPC du protocole de multiplication de 2 nombres    */
/*                  à définir par le programmeur avant de faire "rpcgen"         */
/*-------------------------------------------------------------------------------*/
struct nbres {
        int opd1;
        int opd2;
        };

program MULPROG1 {
        version MULVERS1 {
                int SVMULT ( nbres ) = 1;
        } = 1;
} = 91;

/*-------------------------------------------------------------------------------*/
/*      multsv.c:   programme serveur                                            */
/*                  implantation de la procédure distante "svmult"               */
/*-------------------------------------------------------------------------------*/
≠include <rpc/rpc.h>
≠include <sys/dir.h>
≠include "mult.h"

int  *
svmult_1(operat)
struct nbres *operat;
{
        static int result;                          /*   doit être déclaré en static
  */
        result = operat->opd1 * operat->opd2;
        return(&result);
}
```

```
/*-------------------------------------------------------------------------------*/
/*      multcl.c:       programme serveur                                        */
/*                      implantation de la procédure distante "svmult"           */
/*-------------------------------------------------------------------------------*/
≠include <rpc/rpc.h>
≠include <stdio.h>
≠inlcude "mult.h"                          /* déclarations communes générées par rpcgen
  */

main(argc,argv)
int argc;
char *argv[];
{
  CLIENT *cl;                              /* déclaration du pointeur de type CLIENT      */
  int *result;
  char *server;
  int wopd1, wopd2;
  struct nbres calcul;

  if (argc > 2) {                          /*    si nombre d'arguments ttrop grand erreur */
       fprintf(stderr,"\nUsage: %s [host] \n", argv[0]);
       exit(1);
  }
  server = "localhost";                    /*  par défaut, server = host du client        */
  if (argc > 1) server = argv[1];          /* si host précisé, server = 2ème argument     */

  printf("\nEntrez deux nombres: \n");
  scanf("%d %d", &wopd1, &wopd2);          /* lecture des deux nombres à l'écran          */
  calcul.opd1 = wopd1;                     /* affectation des deux nombres dans la structure */
  calcul.opd2 = wopd2;

  cl = clnt_create (server, MULPROG1, MULVERS1, "tcp");          /* création du lien avec le
serveur       */
  if (cl == NULL) {
       clnt_pcreateerror(server);
       exit(1);
  }
  result = svmult_1(&calcul,cl);           /* on passe l'adresse de la structure
  */
                                           /* à l'appel de la procédure distante          */
  if (result == NULL) {
       clnt_perror(cl, server);
       exit(1);
  }
  if (*result == 0) {
       fprint(stderr,"%s: %s ne peut pas établir la connexion\n",argv[0], server);
       exit(1);
  }
  exit(0);
}
```

```
/*-------------------------------------------------------------------------------*/
/*    mult.h :  déclarations communes générées par rpcgen                        */
/*-------------------------------------------------------------------------------*/
struct nbres {
        int opd1;
        int opd2;
};
typedef struct nbres nbres;
bool_t xdr_nbres();

≠define MULPROG1 ((u_long)91)
≠define MULVERS1 ((u_long)1)
≠define SVMULT ((u_long)1)
extern int *svmult_1();


/*-------------------------------------------------------------------------------*/
/*    mult_svc.c:  squelette du serveur généré par rpcgen                        */
/*-------------------------------------------------------------------------------*/
≠include <rpc/rpc.h>
≠include <stdio.h>
≠inlcude "mult.h"                          /* déclarations communes générées par rpcgen
  */

static void pirprog1_1();

main()
{
  SVCXPRT *transp;
  pmap_unset (MULPROG1, MULVERS1);
  transp = svcudp_create(RPC_ANYSOCK);
  if (transp == NULL) {
        fprintf(stderr, "cannot create udp service.\n");
        exit (1);
  }
  if (!svc_register(transp, MULPROG1, MULVERS1, mulprog1_1, IPROTO_UDP)) {
        fprintf(stderr, "unable to register (MULPROG1, MULVERS1, udp). \n");
        exit(1);
  }
  transp = svctcp_create(RPC_ANYSOCK, 0, 0);
  if (transp == NULL) {
        fprintf (stderr, "cannot create tcp service.\n");
        exit(1);
  }
  if (!svc_register (transp, MULPROG1, MULVERS1, mulprog1_1, IPROTO_TCP)) {
        fprintf(stderr,"unable to register (MULPROG1, MULVERS1, tcp).\n");
        exit(1);
  }
  svc_run();
  fprintf(stderr,"svc_run returned\n");
  exit(1);
}
```

```
static void
mulprog1_1(rqstp, transp)
struct svc_req *rqstp;
SVCXPRT *transp;
{
  union {
        nbres svmult_1_arg;
  } argument;
  char *result;
  bool_t (*xdr_argument) (), (*xdr_result) ();
  char *(*local) ();

  switch (rqstp->rq_proc) {
        case NULLPROC:
              svc_sendreply (transp, xdr_void, NULL);
              return;
        case SVMULT:
              xdr_argument = xdr_nbres;
              xdr_result = xdr_int;
              local = (char *(*) () ) svmult_1;
              break;
        default:
              svcerr_noproc(transp);
              return;
  }
  bzero(&argument, sizeof(argument));
  if (!svc_getargs(transp, xdr_argument, &argument))  {
        svcerr_decode(transp);
        return;
  }
  result = (*local) (&argument, rqstp);
  if (result != NULL && !svc_sendreply(transp, xdr_result, result))  {
        svcerr_systemerr(transp);
  }
  if (!svc_freeargs (transp, xdr_argument, &argument))  {
        fprintf(stderr, "unable to free arguments\n");
        exit(1);
  }
}
```

```
/*-------------------------------------------------------------------------------*/
/*      mult_clnt.c :      interface RPC du protocole de multiplication de 2 nombres
  */
/*                         à définir par le programmeur avant de faire "rpcgen"      */
/*-------------------------------------------------------------------------------*/
≠include <rpc/rpc.h>
≠include <sys/time.h>
≠include "mult.h"                                   /* généré par rpcgen
  */

static struct timeval TIMEOUT = { 25, 0 };

int *
svmult_1(argp, clnt)
nbres *argp;
CLIENT *clnt;
{
        static int res;

        bzero(&res, sizeof(res));
        if (clnt_call(clnt, SVMULT, xdr_nbres, argp, xdr_int, &res, TIMEOUT) != RPC_SUCCESS) {
             return (NULL);
        }
        return (&res);
}

/*-------------------------------------------------------------------------------*/
/*      mult_xdr.c:      routine XDR nécessaire à la conversion des données
  */
/*                         générée par "rpcgen"                                  */
/*-------------------------------------------------------------------------------*/
≠include <rpc/rpc.h>
≠include "mult.h"

bool_t
xdr_nbres(xdrs, objp)
XDR *xdrs;
nbres *objp;
{
        if (!xdr_int (xdrs, &objp->opd1)) {
                return (FALSE);
        }
        if (!xdr_int (xdrs, &objp->opd2)) {
                return (FALSE);
        }
        return (TRUE);
}
```

# BIBLIOGRAPHIE

[ABRI74] ABRIAL J.R., *Data semantics in Data Base Management* (Klimbie, J.W. and Koffeman, K.L., eds), North Holland, Amsterdam.

[BOUR82] BOURNE S.R., *The Unix System,* Addison-Wesley publishing company,1982

[CHRI87] CHRISMENT Claude, *Mise en œuvre des bases de données- Principes méthodologiques,* Edition Eyrolles 1987

[COLL88] COLLET Christine et FAUVET Marie-Christine, *SQL/UFI - Reference Guide*, 1988

[CORN81] CORNAFION, *Systèmes Informatiques répartis - Concepts et techniques,* Dunod Informatique, Edition 1981

[DB287] IBM, Database 2: *Advanced Application Programming Guide - Release 3,* 1987

[DELO83] DELOBEL Claude et ADIBA Michel, *Bases de données et systèmes relationnels*, Dunod Informatique, Edition 1983

[DIAZ85] DIAZ M., MONDAIN-MONVAL P., *Appel de procédures à distance dans les réseaux hétérogènes,* Actes du premier colloque $C^3$, Septembre 1985

[HAJH88] HAJ HOUSSAIN Samer, *DOSIS: Un serveur OSI pour l'ouverture des systèmes distribués au monde extérieur,* Thèse de docteur INPG, Grenoble, Janvier 1988

[LAFO88] LAFORGUE Pierre et PAIRE Eric, *Le réseau hétérogène multi-médias de l'I.M.A.G.*, rapport interne, Grenoble 1988

[ORA1] ORACLE, *Database Administrator's Guide,* 1988

[ORA2] ORACLE, *PRO*C Compilers programmer's guide,* 1988

[ORA3] ORACLE, *SQL*PLUS User's guide*, 1988

[PAJO82] PAJON Marie-Christine, "*Conception et Réalisation d'un serveur d'Information Bibliographique*", Mémoire CNAM Grenoble, A paraître fin 1989

[PUJO82] PUJOLLE G., *La télématique réseaux et applications*, Edition Eyrolles , 1982

[REY84] REY Josiane, *Conception et réalisation du logiciel de gestion du service reprographie de l'I.M.A.G. sous Multics*, rapport interne, Octobre 1984

[RITC87] RITCHIE Dennis M, T*he C programming language reference manual*, 1987

[RPC86] SUN Microsystems, *RPC programming guide - Revision B* , Février 1986

[SOUP88] BENNET M.,BERARD P., LENNE C., LOPEZ M., *SOUPE : Une interface de haut niveau pour Oracle*. Centre de recherche du groupe BULL, rapport interne, Mai 1988

agent client 74; 83; 84
agent serveur 74; 83; 84
appel de procédure à distance 61; 71; 86
AppleTalk 9
arborescence 10
association 25
attribut 18
autorisation 31
AWK 44
base de données 68; 93
chargement 29; 41; 42
client-serveur 57
cluster 28
clé primaire 18
codification 26; 27; 30
colonne 29
communication 57; 60; 67; 68
contrôleur de communication 54
couche application 52
couche de liaison physique 50
couche liaison de données 50
couche présentation 52; 64
couche réseau 50
couche session 52
couche transport 51
DATABASE 27
datagramme 55; 100; 101
dictionnaire de données 94
documentation 10; 46
environnement d'une base de données Oracle 27
environnement utilisateur 34
Ethernet 9; 53; 103
fichier de contrôle 29; 42
fichier de création 31; 96
FICHIER LOG 30
fichiers de données 30
fonction booléenne bind 58
fonctions HLI 78
formatage des fichiers 39
formes normales 23
formulaire 37; 39; 97
fragment 102
fragmentation 56; 101; 103
I.M.A.G. 9
index 28
industrie 46
interface utilisateur 71; 81
laboratoire 18
MacIntosh 9
migration 33
mode connexion 51; 58
mode sans connexion 51; 59
modèle OSI 49
modèle relationnel 17
Normalisation 23
numérotation 18; 19
ODL 29; 42; 44
Oracle 15; 69; 93
outils 11
paquet de données 62
paramètres 73; 86
partition 27; 93
passerelle 100
pointeurs 73
programme 33; 39
protocole de réseau IP 56
protocole TCP 56
protocole UDP 55
protocole XDR 64
protocoles TCP/IP 55; 100
prototype 71; 74; 82
précompilateur 76; 78; 79
recherche 46
relations 35
représentation 10; 64
routage 56; 100
rpcgen 65
réassemblage 103
répertoire 33
segment 103
Serveur 105; 110
services communs 53
sockets 10; 57
SOUPE 76; 80
sous-répertoires 10
sous-thèmes 17
space definition 28; 93
SQL dynamique 73; 76; 78
SQL*PLUS 31
structure d'organisation 68; 70
structure de champs 86
stub client 63; 64; 65
stub serveur 63; 64
table 28; 93
thèmes 17
transmetteur 54
type LONG 44; 93
unicité 18
Unix 9; 10; 64
vue 29
échantillon 46
équipe 18

Dominique VOBORIL

Conception et réalisation d'un service réparti sur réseau pour la consultation de bases d'informations

Mémoire d'ingénieur C.N.A.M. Grenoble 1989

Une base de données relationnelle centralisée sur un ordinateur connecté à un réseau hétérogène met en évidence le problème de communication entre processus distants.

Ce mémoire analyse les données nécessaires à la mise en place d'une base relationnelle Oracle et propose une codification de l'environnement de ce SGBD sous un système Unix. Sont présentés, les outils d'Oracle utilisés pour la création de la base de données.

Un serveur a été réalisé dans un réseau local Ethernet reliant des machines fonctionnant sous Unix en utilisant un mécanisme d'appel de procédures à distance dans le but d'accéder à toute base de données Oracle.

Cette réalisation a soulevé les problèmes d'interfaçage entre les protocoles généraux de communication (TCP/IP) et les outils d'interrogation locale d'une base de données Oracle.

mots-clés : bases de données relationnelles, Oracle, interconnexion de réseaux, UNIX, appel de procédure à distance, protocoles TCP/IP

keywords : relationnal database, Oracle, network interconnection, UNIX, remote procedure call, TCP/IP protocoles