Étude et mise en place de l'architecture client-serveur dans le groupe Merlin-Gerin

Catherine Laplane

CONSERVATOIRE NATIONAL DES ARTS ET METIERS
CENTRE REGIONAL DE GRENOBLE (C.U.E.F.A)

MEMOIRE
présenté en vue d'obtenir le
DIPLOME D'INGENIEUR C.N.A.M.
en
INFORMATIQUE

par
Catherine LAPLANE

ETUDE ET MISE EN PLACE DE L'ARCHITECTURE CLIENT-SERVEUR DANS LE GROUPE MERLIN GERIN.

Les travaux relatifs au présent mémoire ont été effectués dans le groupe Merlin Gerin sous la direction de Monsieur PELLERIN.

TABLE DES MATIERES

Je tiens à remercier,

Monsieur Claude KAISER, professeur au Conservatoire National des Arts et Métiers, président du jury,

Monsieur Alain CAZES, maître de conférence au Conservatoire National des Arts et Métiers en tant que membre du jury,

Monsieur Jacques COURTIN, professeur à l'université Pierre Mendès France de Grenoble, responsable du cycle ingénieur du C.N.A.M en informatique de Grenoble, en tant que membre du jury,

Monsieur Alain LANDELLE, maître de conférence à l'Institut National Polytechnique de Grenoble, responsable des formations supérieures du C.U.E.FA, pour ses encouragements,

Madame Cécile ROISIN, maître de conférence à l'université Pierre Mendès France de Grenoble, pour l'aide qu'elle m'a apportée dans la rédaction de ce mémoire,

Monsieur Gilbert PACCOUD, responsable du Département Développement Informatique de Merlin Gerin, qui m'a permis de me consacrer entièrement à la rédaction de ce mémoire et d'assister aux séminaires sur le sujet,

Monsieur François PELLERIN , responsable du service Support Outils Logiciels de Merlin Gerin, pour m'avoir proposé le sujet de ce mémoire, pour la confiance qu'il m'a témoignée et pour sa présence au jury,

Monsieur Bernard GAUVAIN, responsable du Pôle de Compétences Réalisations de Merlin Gerin, qui a pris le relais de Monsieur PELLERIN suite à la réorganisation du Département Développement Informatique, et qui m'a laissé le temps nécessaire pour terminer ce mémoire.

Tous ont contribué à la réalisation de ce mémoire et m'ont permis de conclure ainsi de nombreuses années d'études.

RESUME

Dans le domaine de l'informatique de gestion, l'architecture client-serveur est à la mode. Les articles de presse sur le sujet sont très nombreux, des séminaires organisés sur le sujet foisonnent, les fournisseurs aussi bien de services que de matériels présentent le client-serveur comme une composante fondamentale des nouveaux systèmes informatiques.

Cependant, l'absence de consensus sur la définition des termes utilisés tels que coopératif, distribué, client-serveur génère une grande confusion. Ce mémoire propose une définition de ces concepts, présente les différents types d'applications de gestion client-serveur, les apports d'une telle architecture en informatique de gestion mais aussi les difficultés de mise en oeuvre.

Les outils de développement, principalement Atelier de Génie Logiciel et Système de Base de Données Relationnelles, permettant le développement d'applications client-serveur et répondant au cahier des charges de Merlin Gerin sont présentés.

Une expérience de développement d'un projet pilote ayant une architecture client-serveur est exposée. Elle a permis de concrétiser et de valider l'étude théorique, et de confirmer que les outils choisis permettent de développer des applications client-serveur.

INTRODUCTION

Ce mémoire a été effectué dans le service SOL, (Support Outils Logiciels) de la Direction de l'Organisation et des Technologies de l'Information du groupe Merlin Gerin.

Pour répondre aux futurs besoins de la refonte du système d'information, qui devra amener une meilleure portabilité des applications et davantage de modularité, le service SOL a lancé le projet ANAIS qui a pour objectif de définir un environnement de développement prenant en compte les nouvelles technologies telles que Atelier de Génie Logiciel et architecture client-serveur.

Le premier chapitre présente les objectifs de ce projet et les contraintes dues au contexte.

Ce mémoire a été réalisé dans le cadre du projet ANAIS, ses deux principaux objectifs sont :

- L'étude de l'architecture client-serveur, dans le domaine particulier de l'informatique de gestion, qui comprend : la définition de l'architecture client-serveur en repositionnant cette architecture par rapport aux architectures distribuées et coopératives, la définition des composants réseaux nécessaires à sa mise en oeuvre, la présentation des différents types d'architectures client-serveur en informatique de gestion, la présentation des apports d'une telle architecture en informatique de gestion mais aussi les difficultés de mise en oeuvre. Tout ceci est l'objet du chapitre 2.

- Le développement d'un projet pilote avec une architecture client-serveur en mettant en oeuvre les outils de développement choisis dans le cadre du projet ANAIS.

Les raisons du choix des outils de développement et la démarche qui a été adoptée pour permettre ce choix, sont exposées dans le chapitre 3.

L'expérience de développement du projet pilote est relatée dans le chapitre 4. Elle a permis de concrétiser l'étude théorique, de confirmer que les outils choisis permettent de développer des applications client-serveur, et de valider l'étude théorique du chapitre 2.

CHAPITRE 1.

LE PROJET ANAIS ET SON ENVIRONNEMENT

1.1. Le groupe Merlin Gerin

Le groupe Merlin Gerin, filiale du groupe SCHNEIDER, compte environ 32000 salariés et a réalisé en 1990 un chiffre d'affaires de plus de 18 Milliards de Francs. La part internationale est de 53%.

Merlin Gerin a pour vocation de concevoir, de fabriquer et d'installer des produits, ensembles et systèmes qui gèrent, protègent les installations électriques et assurent la sécurité des personnes qui les utilisent.

1.2. Le Département Développement Informatique

La principale mission du Département Développement Informatique est de mettre à disposition des entités du groupe (unités ou filiales) les applications informatiques de gestion spécifiées par les fonctions (commerce, achats, production...).
Ce département (composé d'environ 80 personnes) assure la gestion des logiciels par domaines applicatifs. Il fait partie de la DOTI (Direction de l'Organisation et des Technologies de l'Information).
Pour la société Merlin Gerin à Grenoble, ces applications tournent sur un IBM 3090 central, et plus de cinquante filiales réparties dans le monde sont équipées chacune d'un HP3000.

1.3. Le projet ANAIS dans l'équipe SOL (Support Outils Logiciels)

Tous les membres de l'équipe SOL de DDI sont impliqués dans le projet qui a pour nom de code ANAIS :

Architecture and
Norms for
Applications of
Information
Systems

La mission permanente de l'équipe SOL consiste à apporter aux utilisateurs et aux informaticiens du groupe Merlin Gerin des méthodes et des outils de développement performants.

Cette équipe est composée de spécialistes dans les domaines suivants :

- méthodes, conduites de projet,
- gestion des encyclopédies,
- bases de données relationnelles,
- outils de développement sur gros systèmes :
  langages, normes, outils spécifiques,
- outils de développement micro.

1.4. Objectifs du projet ANAIS

Doter les informaticiens du groupe Merlin Gerin d'outils de développement améliorant leur productivité.

Il s'agit d'**industrialiser** la production de code.

Le code produit devra être **indépendant** de l'architecture cible.

Cet objectif principal peut se décomposer en plusieurs sous objectifs :

- minimiser le coût de portabilité sur une architecture cible différente,
- augmenter la productivité du développement,
- augmenter la qualité du code produit.

1.5. Contexte du projet ANAIS

Le projet répond à plusieurs préoccupations actuelles de Merlin Gerin :

- l'ensemble des applications informatiques du groupe est remis en cause, elles devront être plus modulaires et plus portables,
- les utilisateurs demandent des applications offrant une interface graphique,
- la production d'applications et leur mise en oeuvre doivent être industrialisées,
- la maintenance des applications mobilise trop de ressources et doit donc être moins consommatrice à l'avenir.

Le **choix d'une architecture client-serveur** est envisagé à ce jour comme la solution la plus ouverte et la plus adaptée aux besoins de l'entreprise.
L'objectif de ce mémoire est de vérifier que les avantages attendus sont réels. Le chapitre 2 montre en effet que ce type d'architecture permet une transparence de la localisation des données et une meilleure évolutivité des applications. Les applications sont indépendantes des possibilités de re-distribution ou de re-partitionnement des données (pour équilibrer les charges par exemple), ce qui n'implique aucune modification ou recompilation des programmes (Cf. 2.6.1).

Le **soutien d'un atelier de génie logiciel** (AGL) s'est imposé pour faire face à la complexité de développement d'applications de ce nouveau type d'architecture.

Le projet ANAIS doit donc prendre en compte ces différents besoins et proposer d'une part l'utilisation de technologies nouvelles, architecture client-serveur et atelier de génie logiciel, et d'autre part définir des standards et normes communes de développement et de conception d'application, pour permettre un bon cumul d'expérience.

1.6. Stratégie informatique de Merlin Gerin

Trois projets à la DOTI (Direction de l'Organisation et des Technologies de l'Information) visent à définir la stratégie informatique du groupe Merlin Gerin :

. **HERMES** : a en charge la définition des **plates-formes techniques** c'est à dire des stations de travail, des serveurs, du réseau groupe, des outils de messagerie.

. **SIGMA :** a en charge la définition des **architectures applicatives.**

. **ANAIS** : a en charge la définition des **architectures techniques**, ainsi que la définition et la mise en place des techniques et **outils de production des applications.**

Ces trois projets partagent le même objectif : industrialiser la production des applications.

1.6.1. Le projet SIGMA

L'objectif de ce projet est de déterminer les règles d'architecture des systèmes d'information futurs.
Les applications informatiques constituent un support de transfert de l'expérience et du savoir faire Merlin Gerin dans les fonctions constituant le métier de l'entreprise (le commerce, les achats, la comptabilité, la gestion de production...). Les applications informatiques jouent un rôle important sur le niveau de performance des personnes assurant ces fonctions.
Les principes suivants ont été énoncés en début de projet et sont à prendre en compte par les deux autres projets HERMES et ANAIS.

L'objectif à atteindre est d'avoir la même application informatique partout où les fonctions de l'entreprise sont identiques. Une application informatique sera couverte par 1 à N logiciels, chaque logiciel étant unique dans le groupe.
Il faudra donc disposer d'une collection de logiciels qui, par assemblage, couvriront la majorité des entités du groupe, une entité pouvant être une unité ou une filiale du groupe.
Les règles de fonctionnement de l'entreprise seront standardisées.

Le développement "groupe" des logiciels devra être envisagé, c'est à dire que les compétences informatiques des filiales pourront être mobilisées pour le développement de projets communs. Il faut donc définir des normes de développement et une organisation rigoureuse pour le partage des développements.

Le principe de l'**autonomie** des sites d'exploitation a été retenu : on rend à l'entité la maîtrise de ses moyens de production informatique.

L'**asynchronisme** devient la règle des échanges entre les entités de l'entreprise. L'application émettrice formule des demandes mais n'attend pas une réponse éventuelle pour poursuivre son traitement.

Ce principe permet de respecter l'autonomie des applications lors des fonctionnements en mode dégradé.

Les logiciels communiquent entre eux par l'intermédiaire d'interfaces standardisées.

La redondance d'information est admise.

1.6.2. Le projet HERMES

L'objectif de ce projet est de définir les différents éléments composant le réseau international du groupe. Ce réseau international est constitué de réseaux locaux installés dans les différents sites.

L'architecture réseau retenue à la suite de ce projet est la suivante :

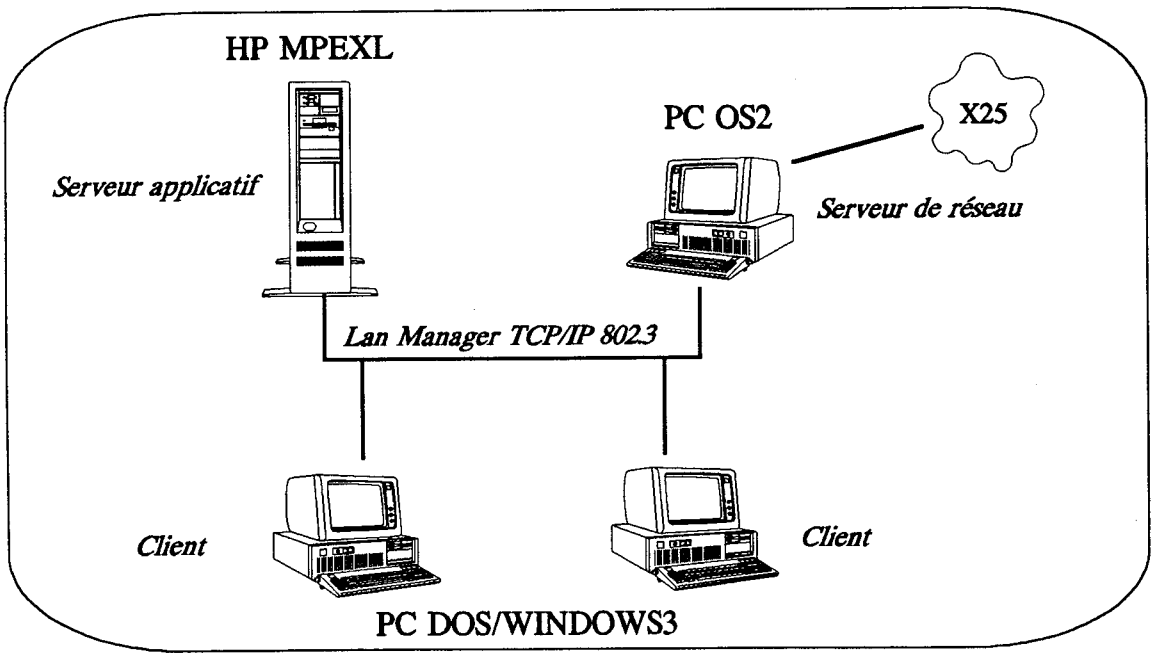


Figure 1.1 Architecture proposée par le projet Hermès.

Les préconisations pour ce qui concerne les fonctions des composants du réseau sont les

suivantes :

- Le serveur PC OS2 est utilisé comme passerelle de communication et serveur des outils bureautiques,
- Le serveur HP MPEXL sera :
  - . serveur applicatif,
  - . serveur messagerie HPBUREAU,
  - . supportera l'outil de transfert de fichiers interne GIDE.
- Les postes clients sont des micro-ordinateurs PC DOS sous Windows.
- Définition du réseau local :
  - . méthode d'accès au réseau 802.3, 10Base T Ethernet, CSMA/CD,
  - . protocole de transport TCP/IP, qui permet de connecter des matériels hétérogènes sur le même réseau,
  - . gestionnaire de réseau local LAN MANAGER de Microsoft.

Le choix du gestionnaire de réseau s'est porté sur Lan Manager de Microsoft plutôt que sur Netware de Novell pour les raisons suivantes :

- . c'est une norme retenue par l'OSF dans le cadre de son architecture DCE (Cf. 2.3.4),
- . il est deux fois moins cher,
- . les perspectives d'évolution sont intéressantes, il sera disponible sur de multiples plates-formes.

CHAPITRE 2.

L'ARCHITECTURE CLIENT-SERVEUR

2.1.Introduction

Le client-serveur est aujourd'hui un concept qui fait couler beaucoup d'encre dans le domaine de l'informatique de gestion. Les articles de presse, très nombreux sur le sujet, et les fournisseurs de matériels et d'outils de développement suggèrent une banalisation acquise de sa mise en oeuvre. Les utilisateurs et les directions informatiques émettent donc comme postulat que toute application doit dorénavant être 'client-serveur' ce qui laisse les services de développement des entreprises dans une grande perplexité sur l'art et la manière de tirer profit de cette architecture.

Faute d'une définition précise du modèle client-serveur, la notion se prête à une multiplicité d'interprétations, plus ou moins commerciales, dans le domaine de l'informatique de gestion. Chaque fournisseur a une approche différente en fonction de ses intérêts et de ses produits. Les uns ont tendances à rattacher le modèle client-serveur à une offre matérielle (micro/mainframe), les autres à une offre logicielle (accès à des données distribuées dans un environnement qui peut être hétérogène) ou encore pour les sociétés de service, c'est l'applicatif qui est au coeur de la notion (traitements clients et traitements serveurs).

L'objectif de ce mémoire est donc d'essayer de faire le point sur ce concept.
Les trois termes, *système distribué*, système *coopératif*, *architecture client-serveur* sont très souvent utilisés indifféremment alors qu'ils cachent des notions différentes.
Les définitions qui suivent tentent de différencier ces notions.

2.2. Définition

2.2.1. Système distribué

Le concept de système distribué admet de nombreuses variations dans sa définition et sa mise en oeuvre. Les ouvrages de référence sur le sujet proposent des définitions très générales.

"Un système distribué est composé de plusieurs processeurs autonomes qui ne partagent pas de mémoire, mais coopèrent en s'envoyant des messages à travers un réseau de communication" [3]. Ces processeurs ne sont pas forcément situés sur des sites éloignés géographiquement, la notion de distribution est indépendante de la localisation géographique d'après [27].

Une connectivité transparente est l'une des propriétés que doit avoir un système distribué. La localisation des données et des traitements doit être transparente à l'utilisateur [5]. Des outils de communication doivent gérer la connectivité entre les différents composants se trouvant sur le réseau.

Une application de messagerie est l'exemple type d'une application distribuée. L'ensemble des postes de travail composent un système distribué puisque l'application doit tourner sur l'ensemble des unités de traitement se trouvant sur le réseau [27].

Une architecture constituée d'un réseau local (LAN Local Area Network) ou d'un réseau étendu (WAN Wide Area Network) est dite distribuée, par contre un système multi-processeurs communiquant par l'intermédiaire d'une mémoire partagée n'entre pas dans la catégorie des systèmes distribués [3].

2.2.2. Système distribué pour l'informatique de gestion

Le terme *système distribué* a été utilisé principalement, en informatique de gestion, pour désigner une configuration réseau où les applications (comptabilité, achats, gestion de production) tournant sur des ordinateurs centraux physiquement distincts, **échangeaient entre elles des données**.
Pour l'informatique de gestion, l'utilisation du terme *système distribué* suppose une distribution des applications et des données qui leurs sont attachées sur plusieurs machines physiquement distinctes [8].

L'arrivée des micro-ordinateurs et des réseaux locaux a donné un nouveau sens à ce terme [18], en permettant **le partage par plusieurs applications, de ressources matérielles (imprimantes ou disques) et des bases de données**.

Les micro-ordinateurs étaient à l'origine destinés à une utilisation personnelle de petites applications faciles à mettre en oeuvre. Les environnements graphiques de type Windows ou MacIntosh et l'utilisation de la souris ont facilité encore davantage leur utilisation.
Les utilisateurs ont alors eu l'idée de transférer les traitements tournant habituellement sur des ordinateurs centraux ou des mini-ordinateurs sur les micro-ordinateurs.
De nouvelles applications ont alors vu le jour. Cette évolution technologique a fournit les moyens d'une décentralisation de l'informatique dans les entreprises.

L'évolution des réseaux et la capacité à faire dialoguer des machines hétérogènes est l'un des principaux facteurs qui a contribué à faire évoluer les architectures centralisées vers des architectures distribuées.

2.2.3. Système coopératif

Le système coopératif apparaît comme un sous-ensemble du système distribué. Dans un système coopératif, les traitements distribués sur plusieurs unités de traitement, composent une application unique alors qu'un système distribué peut être composé de plusieurs applications dialoguant entre elles. Le concept du système coopératif est restrictif par rapport au concept du système distribué. Cette notion de partage d'une application entre plusieurs unités de traitement est admise dans la plupart des définitions.

Exemple de l'application "distribution automatique de billet" :
Trois unités de traitement assurent le fonctionnement du système : le distributeur de billets "A", la banque "B" à laquelle appartient le distributeur, la banque "C" du client du distributeur. Les traitements composant l'application sont distribués sur ces trois unités de traitements.
Les traitements exécutés sur chaque machine sont les suivants :

. le distributeur "A" est chargé de l'interface avec le client, de la saisie du montant désiré, de la saisie du mot de passe, du décodage du nom de la banque du client, de la délivrance des billets...,

. l'unité de traitement de la banque "B" est chargé du pilotage du distributeur de billets et de l'intégrité de la transaction (une transaction est un ensemble d'opérations comportant des mises à jour de données qui doit être exécuté entièrement ou pas du tout). Il envoie l'autorisation ou l'interdiction de délivrance des billets au distributeur après avoir interrogé l'unité de traitement "C", met à jour le montant disponible dans le distributeur et envoie à la banque "C" les informations concernant la transaction (numéro du client, montant, numéro de carte),

. l'unité de traitement "C" vérifie que le compte du client est approvisionné, et met à jour son compte.

Les traitements de l'application sont répartis sur plusieurs machines géographiquement éloignées qui peuvent être de systèmes d'exploitation hétérogènes.

Un système coopératif présente les caractéristiques supplémentaires suivantes par rapport à un système distribué :

. Application unique distribuée

L'application doit être modulaire et faire appel à des 'services', c'est à dire des fonctions assurées par l'une ou l'autre des machines physiques se trouvant sur le réseau, et pouvant être réutilisables d'une application à l'autre.

. Au moins deux unités de traitement

"Deux ou plusieurs programmes interactifs et complémentaires d'une application s'exécutent simultanément sur deux ou plusieurs machines, chacun des programmes utilisant les ressources de la plate-forme sur laquelle il tourne." [1] d'après Gartner Group.

. Fonctions locales

Une partie des traitements doit être située localement sur le poste de travail de l'utilisateur. Ces traitements locaux peuvent être limités, par exemple, à la présentation d'une interface graphique [7].

En conclusion, dans une architecture coopérative, les traitements composant une même application doivent pouvoir s'exécuter simultanément sur différentes unités de traitement situées sur le réseau.
Ainsi, plusieurs applications autonomes dialoguant entre elles par l'intermédiaire d'un réseau ne sont pas des applications coopératives mais des applications distribuées.

2.2.4. L'architecture client-serveur

L'architecture client-serveur est une forme particulière de système coopératif qui implique un processus de questions/réponses entre un *client* qui demande l'exécution d'un traitement à un *serveur* [8].
Une application est découpée en plusieurs programmes qui s'exécutent sur différentes unités de traitement et communiquent entre eux par l'intermédiaire de messages transitant sur le réseau à travers un protocole d'échange standardisé.

Un **serveur** est une machine qui met à disposition des services qui ne sont accessibles que par d'autres machines et jamais directement par l'utilisateur. Un serveur est caractérisé par son interface qui spécifie les services qu'il fournit et leurs modes d'utilisation.
Le **client** fait appel à ces services via l'interface [5].
Le client a le contrôle de l'application et le serveur exécute les demandes de service que lui adresse le client. Le traitement effectué au niveau du client est indépendant de celui effectué sur le serveur.

L'architecture client-serveur n'impose pas la distribution physique des traitements sur les serveurs et sur les clients. Les notions de client et de serveur ne sont pas liés aux machines physiques qui supportent les traitements client ou serveur. C'est le rôle du traitement considéré qui détermine si celui-ci est un traitement 'client' ou un traitement 'serveur' [7].

Dans la plupart des cas, il est vrai que les programmes clients tournent sur des micro-ordinateurs et les programmes serveurs tournent sur des machines plus puissantes capables de supporter les sollicitations de plusieurs clients.
Mais, à l'extrême, le client et le serveur peuvent être supportés par la même machine.

C'est la cas par exemple de l'application STIVE de Merlin Gerin qui doit présenter la même interface à l'utilisateur sur un portable autonome ou sur un micro-ordinateur connecté à un réseau sur lequel se trouve le serveur supportant la base de données. Pour éviter un double développement, l'application de type client-serveur est supportée par un portable qui exécute à la fois les traitements du client et du serveur (Cf. 4.2.1).

2.2.4.1. Les apports de l'architecture client-serveur pour les applications de gestion

Le développement d'une application client-serveur dans un environnement qui peut être à l'extrême **non distribué,** apporte la séparation entre les traitements associés aux bases de données supportés par le serveur d'une part et les traitements associés à l'interface graphique ainsi que les traitements applicatifs (calculs, contrôles de saisie, éditions...) supportés par le client, d'autre part. L'évolution des traitements de gestion est indépendante de l'évolution des bases de données, ce qui isole les traitements applicatifs des bases de données.
La réutilisation des traitements associés à ces données en est facilitée.

L'un des attraits de l'architecture client-serveur repose donc sur la séparation entre les traitements applicatifs qui sont liés aux opérations de gestion, et les services attachés aux données de l'entreprise qui sont indépendants de l'application.

2.2.4.2. Synthèse

Les définitions du client et du serveur s'articulent autour de la notion de service : un programme serveur pour un service A peut devenir un client pour un service B.

> Le concept client-serveur est un concept **applicatif**.
> Une **architecture coopérative** est une **architecture matérielle**
> alors que **l'architecture client-serveur** est une **architecture logicielle.**

En pratique et par abus de langage, les définitions du client et du serveur sont généralement étendues de la manière suivante :

. Les traitements exécutés par un micro-ordinateur, poste de travail de l'utilisateur de l'application sont composés essentiellement d'entités *clients*. Par extension, le micro-ordinateur devient le *client*.

. Les traitements exécutés sur un ordinateur central sont composés essentiellement d'entités *serveurs*. Les ordinateurs centraux ou mini-ordinateurs sont dits *serveurs*.

Les deux notions de système coopératif et architecture client-serveur se rejoignent alors et sont utilisées indifféremment.

Avant d'appliquer ces concepts d'architecture client-serveur dans le domaine qui nous intéresse, à savoir la gestion, nous présentons les composants matériels nécessaires pour mettre en oeuvre ce type d'architecture.

2.3. Composants réseaux

Le réseau, ensemble de matériels et de logiciels chargés d'assurer la communication entre le(s) client(s) et le(s) serveur(s), permet le transfert des demandes de service et des résultats de façon transparente pour l'application. Pour que les systèmes situés sur le réseau puissent communiquer entre eux, le langage de communication doit être commun. La normalisation des protocoles réseaux a pour but d'assurer l'interconnexion de systèmes de constructeurs différents. Le modèle de référence pour l'interconnexion des systèmes ouverts est le modèle OSI. Il est présenté dans le paragraphe suivant.

2.3.1. Le modèle OSI

L'ISO (International Standards Organization) est une organisation internationale faisant partie de l'Organisation des Nations Unies, qui a été créée en 1946 pour prendre en charge les questions de normalisation internationale dans tous les domaines sauf la technologie électrique et électronique. Cette organisation regroupe 91 pays [21].
Cette organisation a défini une norme (ISO 7498) qui spécifie une architecture permettant l'interconnexion des réseaux en 7 couches. C'est le modèle OSI (Open System Interconnection). Il est présenté page suivante.

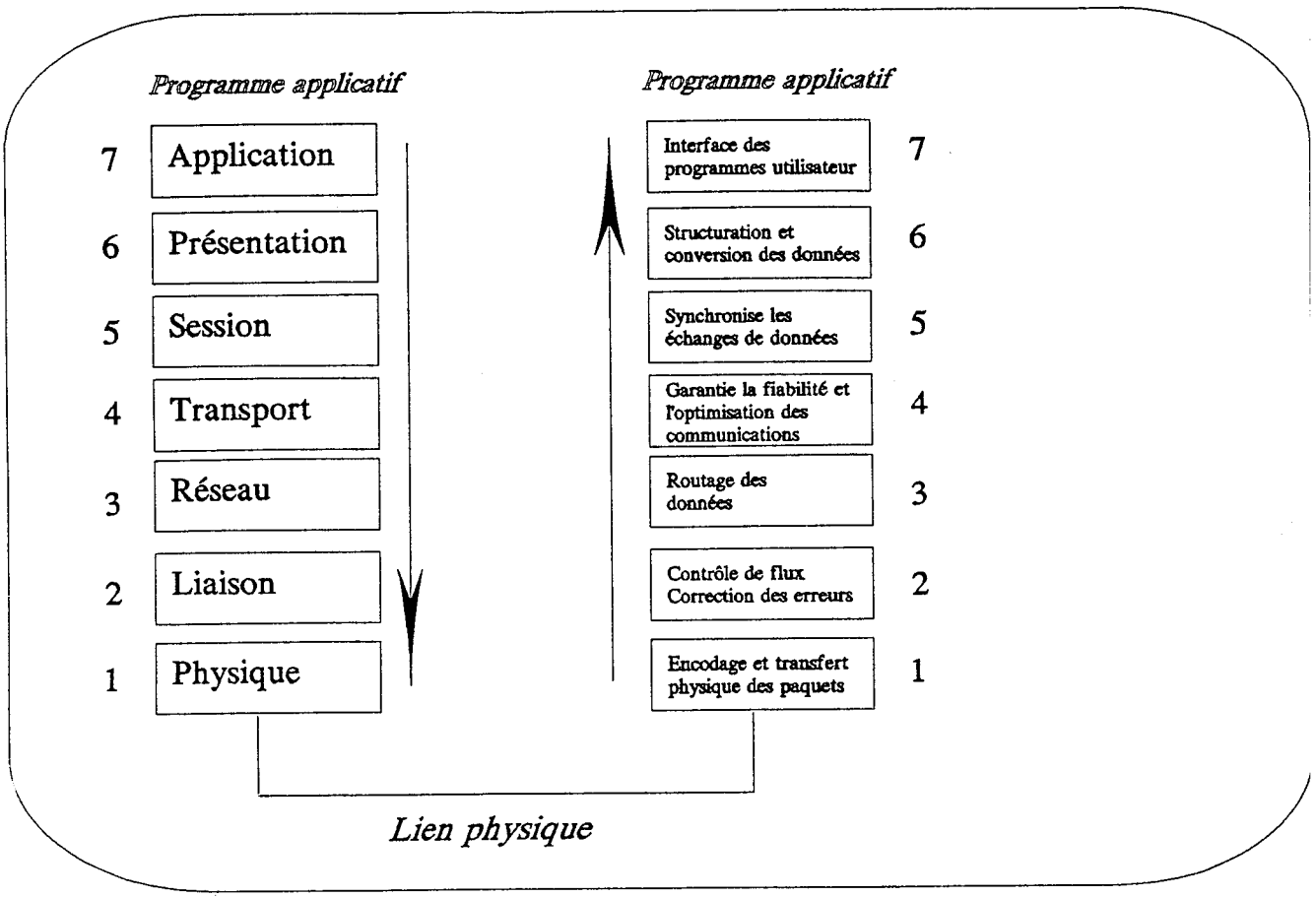


Figure 2.1 Modèle OSI

Les rôles attribués à chacune des couches sont les suivants d'après [32] :

. Couche 1 : **liaison physique**

Cette couche assure le codage physique d'un bit et la transmission physique des données. L'unité de donnée à ce niveau est le bit.

. Couche 2 : **liaison de données**

Cette couche assure le contrôle de flux, la détection et le contrôle des erreurs de transmission. L'unité de donnée est la trame, ensemble d'informations de contrôle tels que numérotation, code de détection d'erreurs et de données binaires.

. Couche 3 : **réseau**

Elle est responsable de l'acheminement des données d'un noeud d'origine à un noeud destination, les données pouvant traverser plusieurs noeuds intermédiaires. Cette couche se charge de prendre la distance la plus courte et sans encombrement.

L'unité de donnée est le paquet, il est constitué de plusieurs trames.

. Couche 4 : **transport**

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

La couche transport doit améliorer, si possible, le service fourni par la couche réseau et a un objectif supplémentaire qui est d'optimiser l'emploi des ressources de transmissions disponibles.

L'unité de donnée est le message.

. Couche 5 : **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. Cette couche garantie la cohérence des données transmises, elle assure que tous les messages d'une transaction sont traités correctement. Si ce n'est pas le cas, la transmission est recommencée.

. Couche 6 : **présentation**

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

Cette couche a pour objectif d'isoler le développeur par rapport :

- au matériel : par exemple, taille des entiers, ordre des bits, taille des caractères,...

- du langage de programmation utilisé : par exemple, type record ou array du Pascal ou type structure ou string du C,
- du compilateur: par exemple 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 de cryptage pour assurer la sécurité et la confidentialité de la transmission, des compressions de texte pour optimiser le volume des informations transitant sur le réseau.

. couche 7 : **application**

Cette couche propose des services de transfert d'informations tels que transferts de fichiers, accès à une base de données ou services de transfert particuliers demandés par une application (messagerie, annuaire).

Ce modèle OSI est plus ou moins respecté dans sa mise en application par les fournisseurs de protocoles de réseaux locaux notamment au niveau de l'implémentation des services à l'intérieur des différentes couches du modèle. Les gestionnaires de réseaux de différents fournisseurs (Lan Manager de Microsoft, Netware de Novell, Appletalk de Apple) certifient tous être basés sur le modèle OSI, cependant cela ne suffit pas à garantir l'interconnexion des réseaux mettant en oeuvre ces différents gestionnaires.

2.3.2. Les normes IEEE 802.X

Le comité 802 de l'IEEE (Institute of Electrical and Electronics Engineers) a été créé en 1980 avec pour mission d'élaborer un projet de normes concernant la technologie des réseaux locaux. Son objectif est de masquer les différences existantes alors dans les couches de bas niveau (1 et 2), de manière à se rapprocher du modèle de référence OSI.

La norme 802.3 définit les topologies de type bus utilisant la méthode d'accès CSMA/CD [32], le câble coaxial ou la paire torsadée comme support de transmission. Cette norme repose sur le protocole Ethernet. C'est la norme retenue dans le cadre du projet HERMES par Merlin Gerin.

La norme 802..4 définit les topologies de type bus, accessibles par la méthode du jeton.
La norme 802.5 concerne les topologies en anneau accessibles par la méthode du jeton.

Les paragraphes qui suivent ont pour objet de présenter les services offerts aux développeurs d'applications client-serveur par les deux principaux gestionnaires de réseaux locaux que sont Lan Manager et Netware.

2.3.3. Les interfaces de programmation des applications client-serveur

Les couches de 1 à 4 du modèle OSI sont généralement appelées les couches de transport. Elles sont en général transparentes pour le développeur qui utilise un gestionnaire de réseau de type Lan Manager ou Netware, qui sont majoritairement utilisés en informatique de gestion. Le fonctionnement de ces couches ne sera donc pas détaillé.

Le protocole couvrant les couches de niveaux 3 et 4 du modèle OSI est en général le protocole TCP/IP (Transmission Control Protocol/ Internet Protocol). Ce protocole est un standard, c'est à dire une 'norme de fait'. Ses spécifications n'émanent pas d'un organisme officiel de normalisation mais il est proposé par de nombreux fournisseurs de gestionnaires réseaux.

Les services d'accès distribués (couches 5 à 7) ont pour objectif de rendre les applications indépendantes des couches de transport (couches 1 à 4).

Les gestionnaires de réseaux (Lan Manager ou Netware) proposent des services de communication inter-processus (IPC Inter-Process Communication) qui remplissent la fonction de la couche 5 du modèle OSI :

. les canaux de communications nommés (Named Pipes),

. des services d'appel de procédures à distance (RPC Remote Procedure Call).

Ces services de communication inter-processus offrent aux développeurs les interfaces de programmation permettant d'établir des connexions logiques entre les applications fonctionnant sur un réseau. Ils permettent de masquer la complexité de la programmation de la communication entre machines.

Ces services sont appelés par le développeur à l'aide d'API (Application Program Interface).

Ces deux types de service sont présentés dans les paragraphes qui suivent.

2.3.3.1. Les Named Pipes

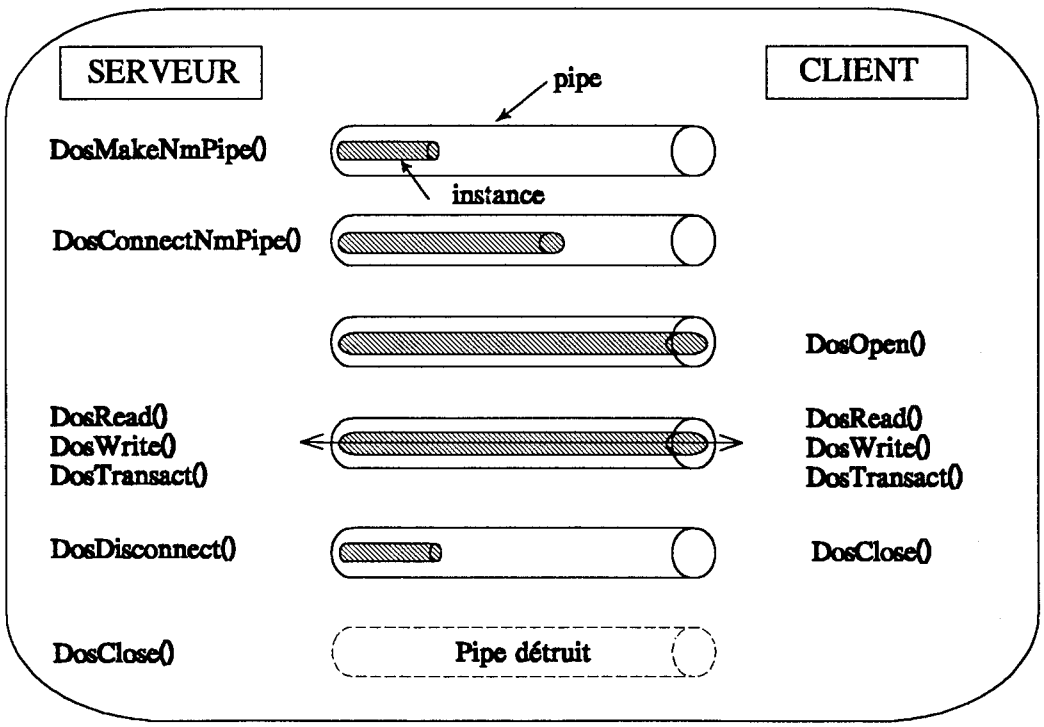


Figure 2.2 Cycle de vie des Named Pipes.

Fonctionnement des *Named Pipes* :

Le serveur crée le *pipe* et attend que le client l'ouvre.

Lorsque le *pipe* est ouvert, le client et le serveur peuvent dialoguer en écrivant ou en lisant les messages transmis par le *pipe*.

A la fin du dialogue le client ferme le *pipe* et le serveur peut le détruire.

Il suffit qu'un processus connaisse le nom d'un *Named pipe* et dispose des droits de sécurité suffisants pour qu'il puisse y accéder.

Ces services ont été utilisés pour développer le projet pilote (Cf. chapitre 4).

2.3.3.2. Les services d'appel de procédures à distance

Ces procédures utilisent un protocole standard qui permet au programme client de faire exécuter une procédure sans se préoccuper du fait que son exécution a lieu sur une autre machine. Le programme contient des appels de procédures distantes comme s'il appelait une procédure locale. Le client envoie un message au serveur qui contient l'appel de la procédure et des paramètres éventuellement. Le serveur exécute la procédure contenue dans le message et renvoie le résultat au client. Le système de communication est synchrone, le processus appelant est suspendu tant que le résultat n'est pas transmis.

2.3.4. L'architecture distribuée OSF/DCE

Face à la grande variété de matériels mis en oeuvre (grands systèmes, minis, micros), à la multiplicité des moyens de communication, (architectures, protocoles), à la diversité des environnements d'exploitation et des mécanismes de communication inter-process, des tentatives de normalisation d'outils de développement sont apparues.

L'OSF (Open Software Foundation) est un consortium international regroupant des fournisseurs de matériels et de logiciels. Il a pour vocation de définir des standards et de commercialiser des

produits réalisés à partir d'apport de ses membres (Hewlett Packard, IBM, Nixdorf, Digital, Bull, Apollo, Philips, Siemens). Les premières réalisations sont l'interface graphique utilisateur Motif et l'architecture distribuée DCE.

L'architecture DCE (Distributed Computing Environnement) est un ensemble d'interfaces élaborées par l'OSF qui définissent la technologie de base, permettant aux constructeurs de systèmes informatiques qui l'adopteront de proposer les éléments nécessaires aux développements d'applications distribuées en environnement hétérogène [21]. DCE est donc un ensemble de technologies de développement permettant de simplifier le travail des développeurs d'applications dans un environnement distribué en les isolant des couches réseaux. Cette architecture normalisée offre des services d'un niveau élevé qui correspondent aux services des couches 5,6,7 du modèle OSI.

Les fonctions clés de DCE sont les suivantes [20] :

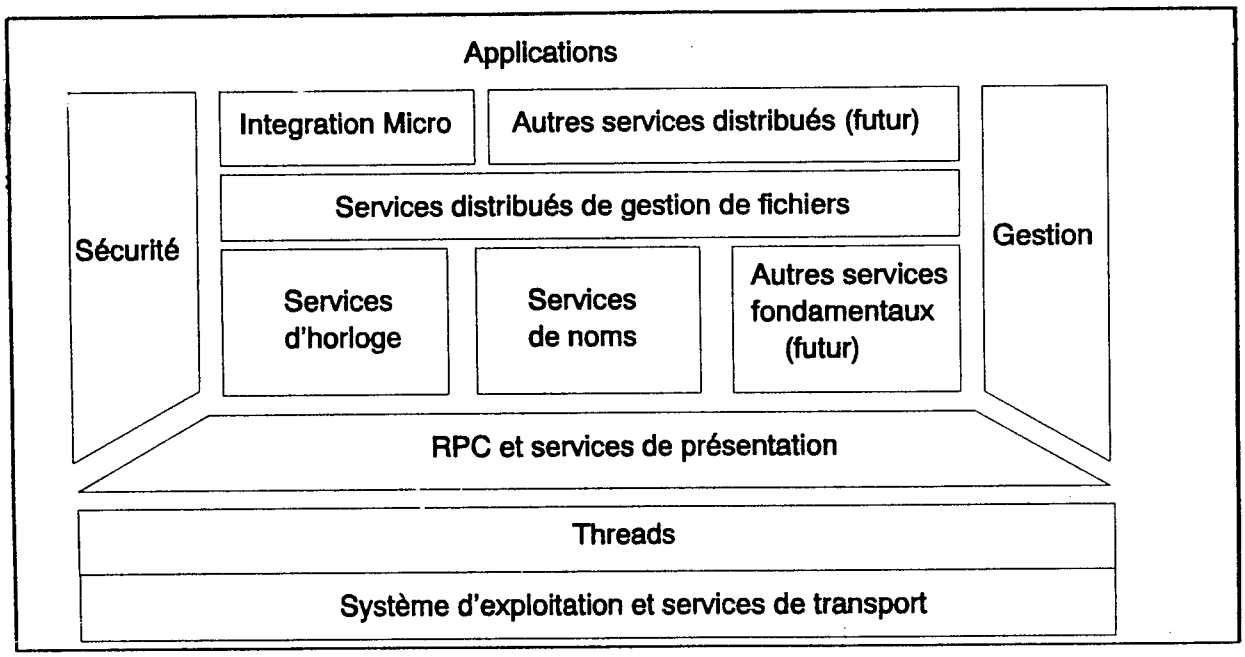


Figure 2.3 L'architecture OSF/DCE.

- **Services de présentation** (*Remote Procedure Call RPC*) : procédure d'appel à distance, service qui permet à une application d'appeler une procédure tournant sur une autre machine (Cf. 2.3.2.2),

- **Services de noms** (*Distributing Naming Service*) : permet d'avoir une désignation globale du système distribué, les ressources sont appelées par des noms, indépendamment de leur localisation dans le réseau.
- **Services d'horloge** (*Time Service)* : service qui permet une synchronisation dans le temps des différentes machines du réseau,
- **Sécurité** (*Security Service*) : service chargé de l'identification et des autorisations d'accès des utilisateurs de ressources sur le réseau,
- **Service de tâches** (*Threads service)* : chargé du support des tâches concurrentes,
- **Services distribués de gestion de fichiers** (*Distributed File System)* : permet d'avoir un accès à l'ensemble des fichiers du réseau comme s'il s'agissait de fichiers locaux, leur localisation est totalement transparente.

Cette norme est très récente et n'est pas implémentée sur tous les systèmes notamment sur le serveur HP3000 utilisé par Merlin Gerin. La société Hewlett Packard annonce l'architecture Encina [36], qui repose sur la norme DCE, pour mi-1993.

Nous n'avons donc pas pu expérimenter cette architecture normalisée DCE dans le cadre de ce mémoire. DCE nous aurait permis d'utiliser des outils de haut niveau pour développer des applications client-serveur. Elle nous aurait évité de développer les couches sessions qui prennent en charge la communication entre le client et le serveur, et le moniteur transactionnel (Cf. 2.4.1.4) pour le projet pilote (Cf. 4.4.7).
DCE nous aurait permis également de tester le développement d'application distribuée sur plusieurs serveurs. Sans outils particuliers à cette architecture, l'utilisation d'un seul serveur reste la règle générale dans les entreprises, les développements étant dans ce cas trop complexes et plus coûteux que pour une architecture traditionnelle.

2.3.5.Choix des différentes couches réseaux dans le cadre de Merlin Gerin.

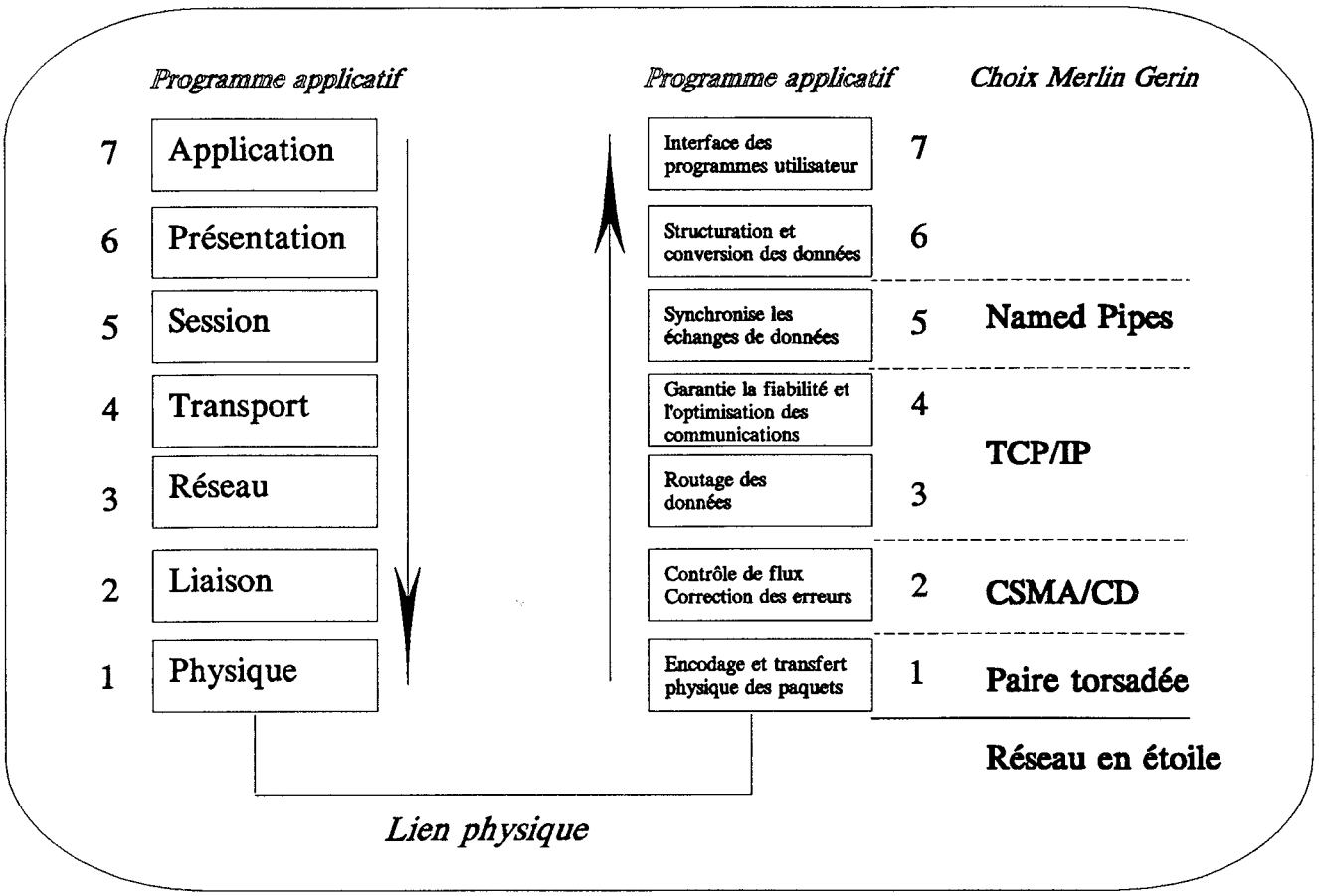


Figure 2.4 Architectureréseau choisie par le projet HERMES.

Le gestionnaire de réseau qui a été retenu à l'issue du projet HERMES (Cf.1.6.2) est Lan Manager avec le protocole TCP/IP.

Les couches de transport (1 à 4) sont transparentes pour le développeur et sont fournies par Lan Manager. Le développeur utilise directement les services de la couche 5 à savoir les Named Pipes, dans le contexte Merlin Gerin.

Les services de communication inter-processus retenus chez Merlin Gerin sont les *Named Pipes*. Ce sont des mécanismes standards, ils sont disponibles sur PC (DOS ou OS2) et sur HP (MPE et Unix) ou bien d'autres plates-formes encore.

Les services se trouvant au dessus de la couche de communication inter-processus et fournis par l'architecture normalisée DCE, détaillée dans le paragraphe précedent, n'ont pas été testés pour l'instant chez Merlin Gerin. Ils n'étaient pas disponibles lors du développement du projet pilote sur les machines utilisées (Cf.4.2.1).

2.4. Mise en oeuvre de l'architecture client-serveur pour les applications de gestion

Une application de gestion peut être découpée en trois modules au moins : [1]

. la logique de présentation, c'est à dire le contrôle des informations saisies, la présentation des données et l'interaction avec l'utilisateur,

. la gestion des données, qui comprend l'accès aux données, l'intégrité des données, la gestion des accès concurrents,

. la logique intrinsèque de l'application, composé de l'enchaînement des dialogues avec l'utilisateur, de l'enchaînement des traitements des deux autres modules (présentation, gestion des données) et des calculs, des éditions.

Exemple : système de réservation de vol.

Le système est utilisé par les agents de voyage dans diverses agences, par la direction et par le personnel du siège social de Paris. Ce système assure les fonctions suivantes :

. consultation des horaires des vols,

. recueillir les informations relatives aux clients,

. réserver les places dans les avions,

. calculer le montant à payer par le client.

Les traitements de l'application se répartissent dans les trois modules cités ci-dessus :

. Logique de présentation : les traitements seront chargés d'afficher les écrans de saisie, de contrôler que les données obligatoires sont saisies, que les valeurs saisies sont valides. Par exemple : faire apparaître le libellé de la donnée à saisie à gauche du

champ de saisie, contrôler le type de l'information (numérique, alphanumérique), afficher le montant saisi cadré à droite de la zone de saisie, contrôler que la valeur saisie est supérieure à une valeur donnée, faire apparaître un signe F à gauche des montants.

. Gestion des données : les traitements sont chargés d'exécuter les requêtes à la base de données centrale qui contient les informations relatives aux vols, et à la base locale par agence contenant les données des clients (spécifiques à l'agence).

. Logique intrinsèque de l'application : les traitements sont chargés de l'appel de procédures d'accès aux données, de l'enchaînement des écrans (saisie des critères de sélection de vols à partir de la demande du client, puis affichage de la liste des vols disponibles, puis écran de réservation d'un vol, puis écran de paiement), du calcul du montant à payer en prenant en compte une remise éventuelle, de l'édition des factures.

Les rôles des composants clients et serveurs sont les suivants pour ce type d'application :

. le **serveur** : il assure la gestion des données et l'accès physique aux données. Il peut également supporter une partie de la logique de l'application, c'est à dire une partie des traitements, dans ce cas il sera dit "serveur de programme" pour Arthur Andersen ou "serveur coopératif" pour HP.

. **le client:** il supporte le dialogue interactif avec l'utilisateur, tout ou une partie de la logique intrinsèque de l'application et la communication avec le serveur.

2.4.1. Différents types de coopération

La frontière entre le client et le serveur peut être située différemment selon le niveau de coopération. Trois types de coopération peuvent être distingués (figure 2.5):

1) Serveur d'application,

2) Serveur de données.

3) Serveur coopératif,

Ces trois types de coopération sont détaillés dans les paragraphes qui suivent et sont présentés dans l'ordre de la plus faible coopération entre le client et le serveur, vers la plus importante.

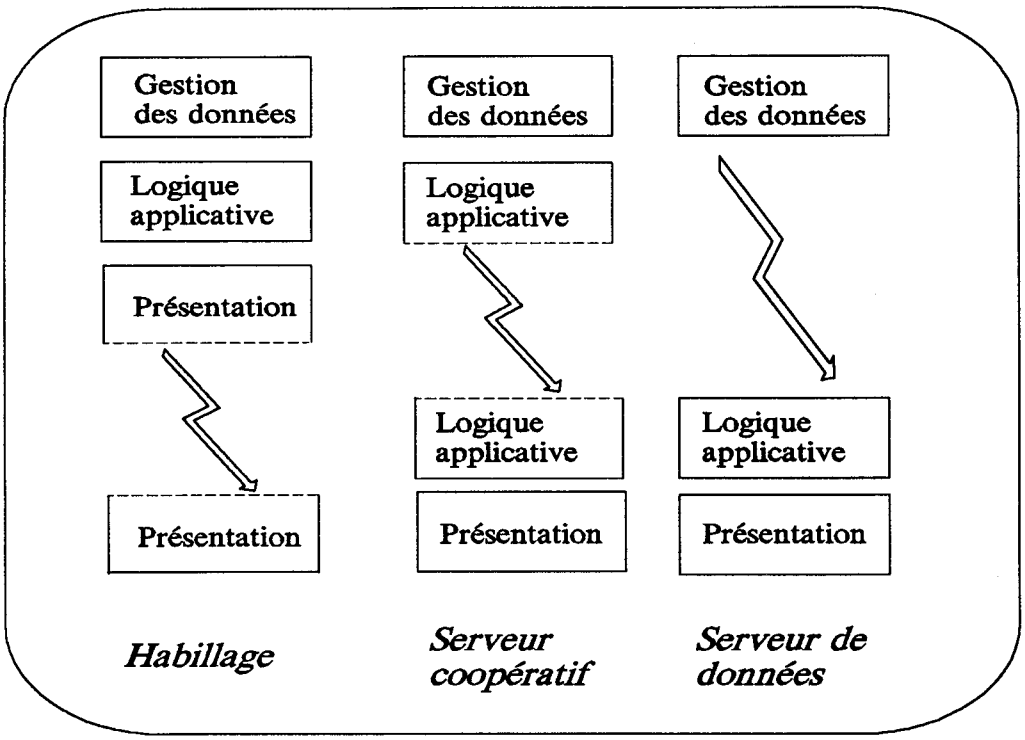


Figure 2.5 Les différents types de coopération.

2.4.1.1. Serveur d'application ou habillage des écrans

Sur le client	: logique de présentation.
Sur le serveur	: logique applicative + gestion des données.

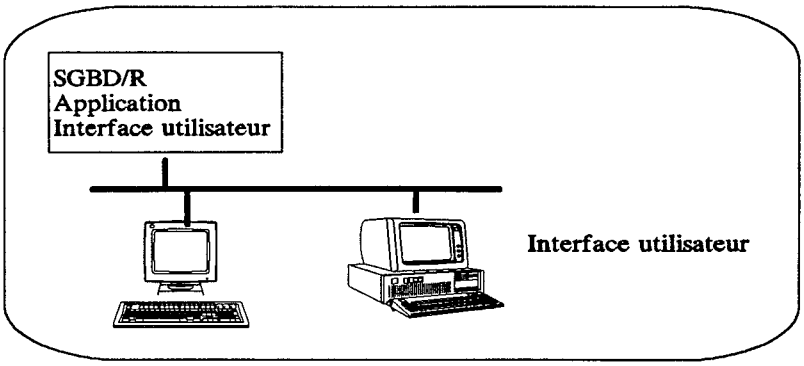


Figure 2.6 Habillage d'applications.

Il s'agit uniquement de fournir aux utilisateurs une interface plus conviviale.

L'interface utilisateur est déportée sur le client pour permettre l'utilisation d'une interface graphique de type Windows (Microsoft) ou Presentation Manager (Microsoft / IBM).

L'application continue à tourner sur machine centrale, les terminaux conservent leurs accès aux applications.

Les ordres du protocole de pilotage du terminal sont simplement traduits en ordres graphiques, permettant de mettre en oeuvre les mécanismes de fenêtrage, de menu déroulant.

C'est le serveur qui est maître de l'application contrairement aux principes de l'architecture client/serveur (Cf. 2.2.4).

Dans ce mode de fonctionnement, certains outils (par exemple VPLUS-WINDOWS de HP ou PAW de CGI) permettent de faire de l'habillage (revamping) d'applications tournant sur site

central. Cette opération consiste à modifier l'interface utilisateur d'une application, en la transférant sur micro-ordinateur sous interface graphique, sans modifier l'application elle-même. La conversion d'un écran 'passif' en écran sous interface graphique est entièrement à la charge du client. Le réseau transporte, comme dans une architecture centralisée, des 'maps' d'écran. Cette architecture permet d'intégrer des micro-ordinateurs dans un réseau d'entreprise existant et de moderniser les applications existantes sans investir dans la réécriture des applications centrales.

On ne peut véritablement parler d'application coopérative pour ce type d'architecture puisque les traitements supportés par le client se limitent à l'affichage des écrans et que le client n'a pas le contrôle de l'application..

2.4.1.2. Serveur de données

Sur le client	: logique de présentation + logique applicative
Sur le serveur	: gestion des données

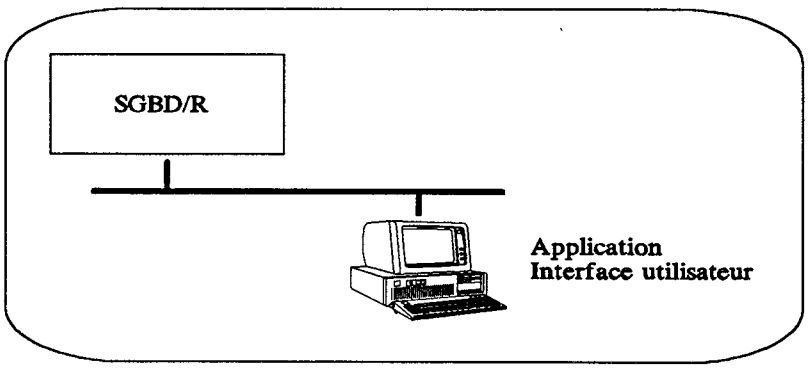


Figure 2.7 Serveur de données.

Le serveur est totalement dédié à la gestion des données tandis que l'ensemble de l'applicatif est exécuté par le poste client. La base de données se trouve sur le serveur, plus puissant et doté d'une capacité de stockage plus importante que le poste client.
Une interface SQL (Strutured Query Language : langage de requêtes normalisé qui permet d'accéder aux données stockées dans un Système de Gestion de Base de Données Relationnelle, SGBD/R) permet au client de définir et d'envoyer facilement une requête sur le réseau vers la base de données du serveur.
Ce type d'architecture repose sur l'utilisation de SGBD/R pour préserver les possibilités de portabilité des applications sur des machines hétérogènes par leurs systèmes d'exploitation. Les traitements sont exécutés par le client et contiennent essentiellement des requêtes d'accès SQL aux données. Le serveur reçoit directement la requête, traite la demande et renvoie le résultat demandé.

Les échanges se limitent, dans un sens, à des ordres SQL entre le client et le serveur et , en retour, à des données brutes que le serveur renvoie vers le client, ce qui génère un trafic réseau important puisque les demandes et les résultats transitent sur le réseau.
Le client est maître de l'application.
Les SGBD/R tels que SQLBASE de Gupta ou ALLBASE de HP permettent ce niveau de coopération.

Ce type de coopération, serveur de données, constitue actuellement la majorité des exemples d'applications client-serveur en informatique de gestion.

2.4.1.3. Serveur coopératif

Sur le client	: logique de présentation + une partie de la logique applicative
Sur le serveur	: une partie de logique applicative + gestion des données

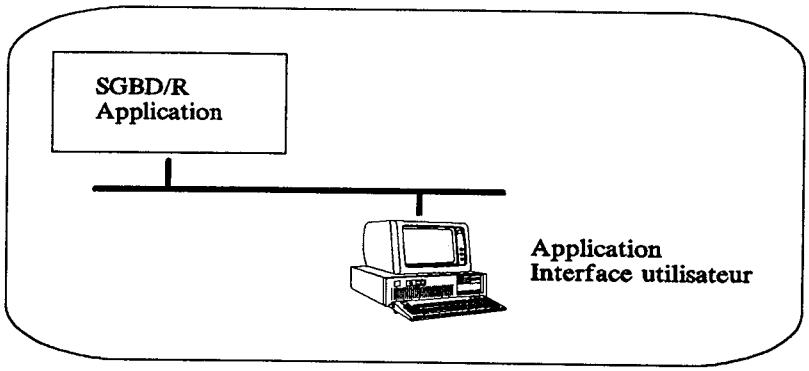


Figure 2.8 Serveur coopératif.

Pour éviter que le volume de données transitant sur le réseau soit important, certains SGBD/R tels que ORACLE, INGRES, SQLSERVER, proposent en standard la possibilité de faire exécuter des procédures stockées (procédures de code SQL stockées sur le serveur), ou de déclencher automatiquement des procédures stockées lors d'accès particuliers à la base de données (notion de triggers)[29]. Ces procédures évitent des allers-retours entre le client et le serveur puisque les procédures stockées sont exécutées par le serveur et permettent de conserver l'intégrité référentielle de la base dans le cas des triggers. Les SGBD/R ne contrôlent pas tous directement l'intégrité référentielle à partir des déclarations faites dans le schéma de la base, contrairement à DB2 [35].

Les procédures stockées permettent, par exemple, de prendre en compte le problème suivant : deux entités sont reliées par une dépendance multi-valuée de type 1 à N, par exemple un client

relié à ses N commandes. Chaque commande contient la clé *N° de client* de son client, c'est une clé étrangère. Dans le cas d'une suppression de client, les options suivantes doivent pouvoir être gérées automatiquement par le serveur :

- vérifier que le client n'a plus de commande, sinon interdire la suppression,
- si un client est supprimé, toutes les commandes sont supprimées en même temps,
- si un client est supprimé, son *N° de client* est positionné à la valeur NUL dans toutes ses commandes.

Ce mécanisme de procédure stockée évite donc des échanges entre le client et le serveur puisque les contrôles peuvent être automatiquement exécutés sur le serveur avant que celui-ci ne renvoie le résultat au client.

Cette architecture (serveur coopératif) peut également être mise en oeuvre à partir d'outils de développement du type Foundation Coopératif de la société Arthur Andersen (cet AGL est présenté dans le chapitre 3) ou tout simplement en développant des procédures serveurs en Cobol sur le serveur, et des procédures client sous Windows en C, ou avec des outils plus simples à utilisés tels que Visual Basic de Microsoft ou NS-DK de Nat-Systèmes (produit utilisé pour le développement du projet pilote). Dans ce cas il faut développer des couches de communication permettant le dialogue entre le client et le serveur, reposant sur les services proposés par les gestionnaires de réseau tels que Lan Manager de Microsoft ou Netware de Novell.

Dans ce type d'architecture, une partie de la logique applicative est supportée par le serveur puisque celui-ci prend en charge les traitements afférents aux données. Le client exécute les traitements de contrôle ou de calcul, et des traitements liés à l'interface graphique.
Le client et le serveur dialoguent toujours par l'intermédiaire de messages.
Cependant, c'est le client qui est toujours maître de l'application.

2.4.1.4. Moniteur transactionnel

Le serveur doit être capable de recevoir des demandes de service en provenance de plusieurs clients simultanément, de les exécuter et de renvoyer la réponse au client correspondant. Ces fonctions sont assurées par un moniteur transactionnel.
Le moniteur transactionnel est un processus qui est exécuté par le serveur et qui est à l'écoute des demandes des postes clients.
Les principales fonctions d'un moniteur transactionnel sont les suivantes :

. gérer la file d'attente des demandes de services envoyés par les clients,

. assurer l'exécution des services demandés,

. renvoyer un code retour et/ou les données aux postes clients,

. gérer la sécurité d'accès au serveur et aux services.

Certains constructeurs fournissent en standard un moniteur transactionnel. C'est le cas de IBM qui fournit le moniteur CICS en standard. Hewlett Packard proposera bientôt l'architecture Encina [36], qui comporte un moniteur transactionnel, sur toutes ces machines.

Les SGBD/R tels que Oracle ou Sybase fournissent les moniteurs serveurs avec leurs produits. Dans la version 6 d'Oracle, à chaque processus client, correspondait un processus serveur, ce qui entraînait une consommation de ressources (mémoire et CPU) importante. Dans la version 7, un seul processus serveur traite toutes les demandes client [29].

Lorsque le moniteur n'est pas fourni par le constructeur du serveur ni pas le fournisseur de SGBD/R, il est nécessaire de développer la fonction de moniteur transactionnel. C'est le cas du projet pilote (Cf.4.4.3) qui utilise le SGBD/R SQLBASE de Gupta sur serveur OS/2.

De la bonne performance du moniteur dépend la performance des applications développées. Ce sera un point d'autant plus critique que le nombre de clients d'un même serveur est élevé.

2.4.1.5. Conclusion

La position de la frontière entre le client et le serveur est dépendante du type d'application et doit être examinée au cas par cas. La volonté de développer des applications client-serveur traduit dans la majorité des cas, la volonté d'optimiser la gestion des ressources existantes mais aussi de rendre les applications plus conviviales. La limite entre le client et le serveur déterminera la qualité des applications, en terme de temps de réponse et de disponibilité des données, et la performance des applications.

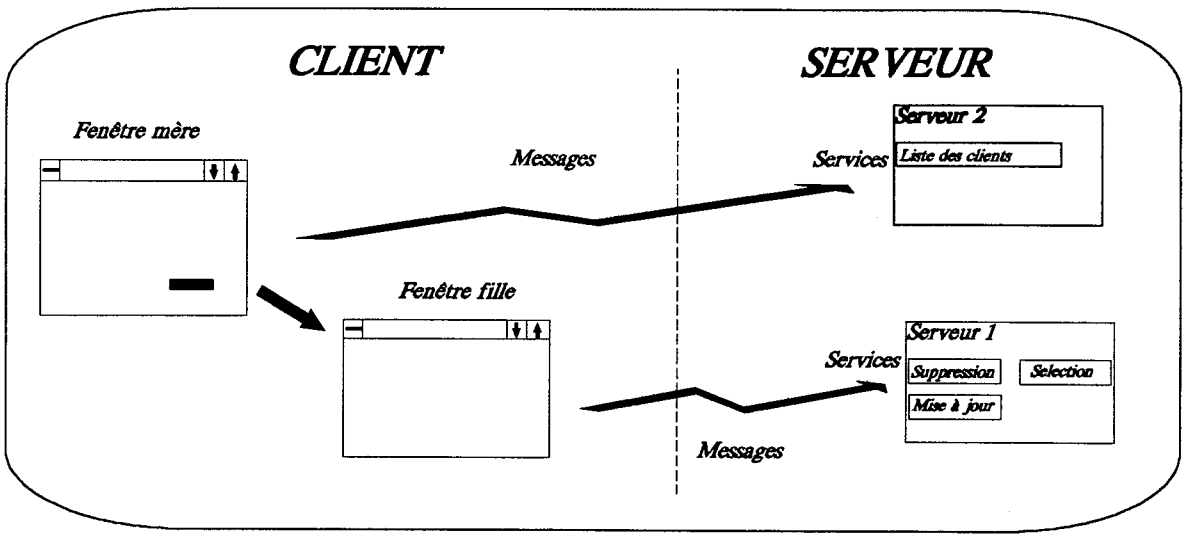


Figure 2.9 Dialogue entre le client et le serveur.

2.4.2. Evolution de la programmation des applications de gestion

2.4.2.1. Interface graphique

L'utilisation d'interfaces graphiques de type Windows ou Presentation Manager est permise par l'architecture client-serveur. De ce fait, on associe souvent interface graphique et concept client-serveur. Il est vrai que dans la plupart des cas le client étant un micro-ordinateur, l'interface utilisateur développée utilise un des avantages de ce type de machine c'est à dire l'interface graphique.
Le type de programmation des applications, dans le cas du niveau de coopération maximum (Cf. 2.4.1.3) s'en trouve alors modifié.
A partir de chaque fenêtre, le choix proposé à l'utilisateur est beaucoup plus riche que dans les applications traditionnelles. A chaque action (click de souris, pression sur une touche du clavier) est associé un événement qui fera appel à une procédure client, par exemple une validation inter-champs, ou qui fera appel à un service du serveur, par exemple pour accéder à la base de données.

Les programmes ne sont plus écrits de façon séquentielle mais ils sont découpés en procédures qui correspondent à chaque événement, que l'utilisateur peut déclencher à partir d'une fenêtre de l'application. La programmation est dite *événementielle*.

2.4.2.2. Développement modulaire et meilleure réutilisation

Les concepteurs devront identifier les actions possibles, disponibles pour les utilisateurs et quels services du serveur ou quelles procédures client leurs sont rattachés.
Les traitements sont isolés les uns des autres en fonction de leur type : attachés aux données, liés à la présentation, purement applicatifs, chargés de la navigation entre les fenêtres de l'application.

L'architecture client-serveur est bien adaptée au développement modulaire des fonctions. Un développeur pourra être amené à développer seulement la partie client ou seulement la partie serveur ou encore une fonction particulière de l'application mettant en oeuvre le client et le serveur. Il convient de définir l'organisation la plus efficace possible.

Cette architecture permet le développement d'applications modulaires et autorise ainsi la réutilisation de services communs. Elle permet la mise en oeuvre de la notion d'encapsulation, c'est à dire de la réutilisation maximum du code des applications, en ce qui concerne les traitements liés aux données d'une part mais aussi les traitements liés aux composants graphiques.

. **Encapsulation des traitements liés aux données** : la modularité des applications permise par l'architecture client-serveur autorise la réutilisation de services communs dans la gestion des données. L'ensemble des contraintes associées aux données est gérée de manière unique et indépendante des fonctions de gestion, les services du serveur sont utilisables par plusieurs applications accédant aux mêmes données, dans le cas où la coopération entre le client et le serveur est maximum (Cf.2.4.1.3).

. **Encapsulation des traitements liés aux composants graphiques** : les composants graphiques et leurs procédures associées représentent des éléments réutilisables qui peuvent être partagés par plusieurs fenêtres avec les mêmes caractéristiques de comportement quelle que soit la fenêtre.

2.5. Enrichissement nécessaire du modèle d'information

Le nouveau modèle d'information adapté à l'architecture client-serveur devra intégrer l'ensemble des entités nécessaires à la description d'une telle application : d'une part les éléments constitutifs d'une interface graphique (fenêtres, menu, list-box, radio-boutons, navigation ...) et d'autre part la prise en compte de la distribution des traitements sur les clients et les serveurs.

A ce jour il n'existe pas de méthode de conception éprouvée ni d'outils de modélisation disponible, c'est à dire que les entreprises puissent mettre en oeuvre, parfaitement adaptés à la modélisation des applications client-serveur. La conception d'une application client-serveur relève donc encore d'une démarche expérimentale dans le domaine de l'informatique de gestion.

2.5.1. Les méthodes de conception

Les méthodes de modélisation traditionnelles telles que Merise, Yourdon ont été développées pour les systèmes à architectures centralisées et ne sont pas adaptées à l'architecture client-serveur :

- elles ne permettent pas la modélisation de la distribution des traitements et données,
- elles ne proposent pas de méthode particulière pour la conception d'interface graphique (utilisée en général dans les applications de gestion de type client-serveur), qui ne peut être séparée du reste de l'application et qui suit une logique événementielle.

Les méthodes de conception traditionnelles utilisées en informatique de gestion, et en particulier Merise, sont en pleine évolution et s'adaptent progressivement aux nouvelles architectures [11] :

- Des évolutions de Merise sont en cours pour prendre en compte les architectures client-serveur. La société Sema propose une variante de Merise, Merise/2 qui propose une adaptation pour les applications distribuées. La modélisation de la répartition des données et des traitements permet de représenter graphiquement leur localisation géographique , le mode de transport.
- L'AFCET (Association Française pour la Cybernétique Economique et Technique) propose d'intégrer les interactions données-traitements dès le niveau conceptuel dans Merise pour permettre leur répartition ultérieure dans les systèmes distribués.

Les sociétés telles que CGI, dont l'AGL repose sur la méthode Merise, ou Arthur Andersen qui utilise la méthode Yourdon, ne proposent pas vraiment d'évolutions en ce qui concerne la spécifications des applications client-serveur. Ces sociétés ont pour objectif de faire fonctionner leurs outils avant tout, et la méthode est encore artisanale pour ce qui concerne la modélisation de la distribution des traitements et de la navigation dans les applications.

Des recherches sont en cours dans les laboratoires des universités et s'orientent vers l'utilisation de modèles objets [14]. " Les méthodes objets... combinent des spécifications détaillées centrées sur la notion d'objet regroupant structures de données et traitements, et des spécifications globales centrées sur les liaisons statiques et dynamiques inter-objets" [17]. Les méthodes objets remettent en cause la séparation de la modélisation des données de la modélisation des traitements (ce qui est une des contraintes de MERISE) et sont donc mieux adaptée à l'architecture client-serveur. Les méthodes orientées objet permettent la modélisation de la notion d'encapsulation (Cf.2.4.2.2) qui est mise en oeuvre par les architectures client-serveur.

L'objectif de ces méthodes est de prendre en compte dès la phase d'analyse conceptuelle, l'architecture du système informatique pour faciliter la répartition des processus de l'application dans une architecture client-serveur.
La méthode COOPE proposée par [14] est événementielle c'est à dire pilotée par les événements et permet une prise en compte plus facile de l'évolution des systèmes d'information. Cette méthode propose une modélisation des applications faisant apparaître le type de coopération mis en oeuvre par les différentes unités de traitements du système d'information.

Cependant il faudra attendre les outils mettant en oeuvre ces nouvelles méthodes pour qu'elles puissent être utilisées efficacement par les entreprises.

2.5.2. Les outils de conception

La société CGI annonce une nouvelle version de l'AGL PACLAN [9] (utilisé pour développer le projet pilote (Cf. 4.3.1) et qui nous intéresse donc particulièrement dans le cadre de MERLIN GERIN), Paclan/NT, pour fin 1993. Ce nouveau produit est le seul à notre connaissance à ce jour à proposer un AGL aussi complet et aussi bien adapté à l'architecture client-serveur. Cependant il restera à vérifier que les annonces alléchantes deviennent bien réalités.

Cette nouvelle version apportera les outils de conception et de développement spécifiques à l'architecture client-serveur, mais reste très proche de MERISE. Le concepteur aura les outils nécessaires à la définition des modèles suivants :

. modèle de distribution des traitements,

. modèle logique de données qui décrit l'ensemble des données et les contraintes qui y sont attachées,

. modèle navigationnel des données qui permet de décrire les vues logiques de données utilisables par une application,

. modèle d'enchaînement des fenêtres pour la spécification de l'interface graphique utilisateur,

.le modèle de donnée intègre les entités nécessaires à la modélisation d'une interface graphique, le méta-modèle de données se trouve page suivante (figure 2.10).

Ces modèles devraient permettre une description complète des applications client-serveur.
En ce qui concerne le développement des applications, la nouvelle version de PACLAN propose un mode de développement basé sur la spécification d'une architecture logique client-serveur, et permet la production de tous les composants nécessaires (services de présentation, services applicatifs, services de navigation, services de données) indépendamment de leur distribution physique. C'est uniquement lors de la génération des programmes que se fera la distinction entre les procédures client et les procédures serveur.

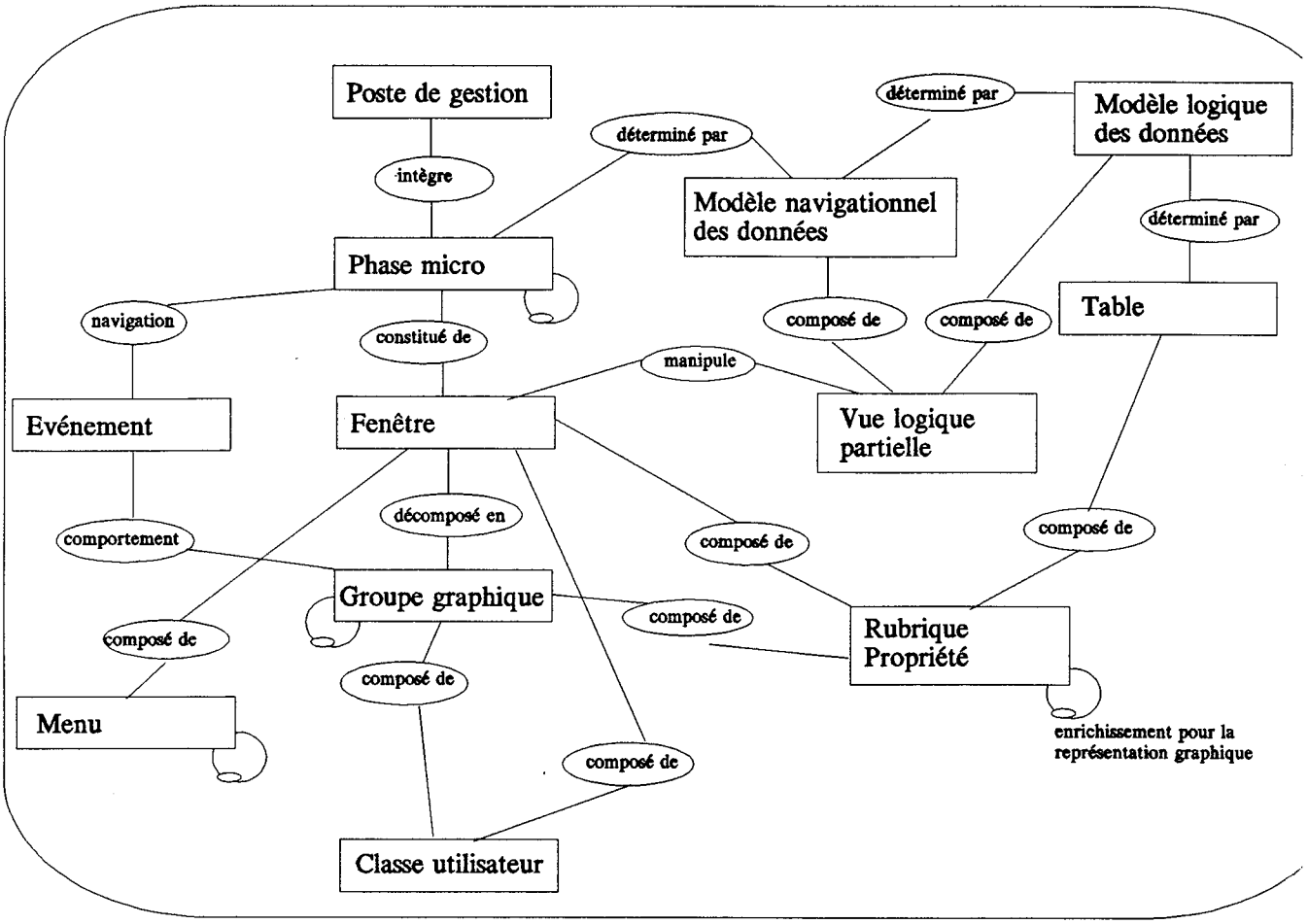


Figure 2.10 CGI : modèle d'information client d'après [9].

2.6. Synthèse

2.6.1. Les avantages de l'architecture client-serveur.

L'architecture client-serveur amène la flexibilité et l'ergonomie de l'interface utilisateur sur la station de travail tout en bénéficiant de la puissance d'un serveur central. Ils permettent d'utiliser au mieux les ressources de chaque système participant au traitement.

Les avantages de ce type d'architecture sont nombreux dans le cas où la coopération entre le client et le serveur est maximum, c'est à dire dans le cas du *serveur coopératif* (Cf. 2.4.1.3) :

Interface graphique

. L'architecture client-serveur est bien adaptée à l'utilisation d'interface graphique (Cf. 2.4.3.1). En général les applications client-serveur en informatique de gestion mettent en oeuvre une interface graphique et bénéficient donc de tous les avantages de l'interface graphique, à savoir : facilité d'apprentissage et d'utilisation des applications par les utilisateurs, meilleure acceptation des applications, réduction des coûts de formation.

Trafic réseau

. Réduction du trafic réseau, le réseau véhicule un minimum de données (demandes de services et résultats). Le contrôle des informations saisies s'effectuant en local, le client transmet au serveur des informations déjà contrôlées ce qui minimise les échanges. Dans le cas du serveur coopératif, le serveur transmet directement les résultats et non pas toutes les données sélectionnées comme dans le cas du serveur de données (Cf. 2.4.1.3).

Réduction du codage des applications

. Partage des traitements communs aux applications : les services d'accès aux données assurés par les serveurs sont partageables par plusieurs programmes clients. Ceci permet une réutilisation du code, ce qui diminue les temps de développement et facilite la maintenance.

. Les règles de cohérence sont centralisées, et sont prises en charge par le serveur.(Cf.2.4.1)

"Le meilleur des deux mondes" [25]

. La distribution des traitements permet une optimisation des ressources machines. Chaque système est utilisé pour ses avantages sans ses inconvénients, par exemple l'interface

graphique des micro-ordinateurs et la capacité de stockage et de traitement des ordinateurs centraux.

Evolutivité

. Cette fragmentation des applications permet également de préserver les investissements informatiques des entreprises. En effet, les évolutions technologiques peuvent être prises en compte sans remettre en cause l'ensemble du système informatique ; un changement de système d'exploitation, d'environnement graphique, de système de gestion de base de données ne touche qu'une partie du système.

. L'architecture client-serveur repose sur la séparation entre les traitements applicatifs liés aux opérations de gestion et les services attachés aux données de l'entreprise qui sont indépendantes des applications. Cette architecture permet une transparence de la localisation des données et donc un système d'information évolutif.

2.6.2. Les difficultés de mise en oeuvre.

A entendre tous les fournisseurs se situant sur le marché du concept client-serveur, cette nouvelle architecture présente des avantages attendus importants mais il nous est apparu un inconvénient majeur : le coût élevé de développement de telles applications par rapport aux développements d'applications dites traditionnelles.

Ce nouveau type d'application, qui n'est pas encore maîtrisé, nécessite des temps de développement plus longs aujourd'hui, pour les raisons suivantes :

Conception et développement plus complexes

. L'interface graphique (mise en oeuvre dans la plupart des applications client-serveur dans le domaine de l'informatique de gestion) d'une part et la répartition des traitements d'autre part

rendent plus difficiles et plus longs le développement et la conception de ce type d'application.

Il existe encore peu de méthodologie de conception et de développement prenant en compte les nouvelles caractéristiques de ces applications sur le marché, qui est encore immature. Peu d'outils sont disponibles aujourd'hui.(Cf.2.5.1).

Formation importante aux nouvelles techniques

. Ce type d'application nécessite une formation des concepteurs et des développeurs à de nouvelles techniques telles que la programmation événementielle, de nouveaux langages de programmation pour développer les parties client, une nouvelle méthodologie de conception et de développement [7].

Technologie de pointe

. Des nouveaux risques de non fonctionnement des applications produites apparaissent. Ils sont dus à notre manque d'expérience de ce type d'architecture, du point de vue des performances (charge du réseau, typologie du réseau), de la définition des types de serveur et de leur répartition.

Maîtrise multi-environnements

. Des compétences client (micro-ordinateur), interface graphique, protocoles réseau, serveur, deviennent nécessaires pour mener à bien un projet de développement d'une application client-serveur. Le chef de projet va devoir faire intervenir une multiplicité d'intervenants qui risquent de se 'renvoyer la balle' lorsqu'un problème technique surgira. Pour pouvoir diagnostiquer la cause d'un problème (mauvaise performance d'une application, blocage, message d'erreur), l'ensemble des compétences serveur, micro-ordinateur, réseau, SGBD/R) sera nécessaire.

Maintenance multi-sites

. Des outils commencent à apparaître mais ils sont peu nombreux et peu fiables (car très récents), dont la fonction est la maintenance des traitements des postes clients et la télédiffusion des nouvelles versions sur de nombreux postes de travail répartis géographiquement [10] [28].
Pour certaines mises à jour ayant donné lieu à des modifications à la fois des programmes serveurs et des programmes clients, les changements de version doivent être pris en compte simultanément sur tous les postes clients et serveurs pour garantir la cohérence entre les programmes serveurs et clients.

. L'évolution des logiciels, des matériels est tellement rapide qu'il est très difficile pour un parc de machines élevé de gérer la cohérence entre les versions des applicatifs, des cartes réseaux, des systèmes d'exploitation, des outils bureautiques divers.

. La difficulté provient de la nécessité de gérer plusieurs machines délocalisées sur des sites géographiquement distincts. Pour les postes clients, les sauvegardes de fichiers, les mises à niveau du système d'exploitation, des applications seront à la charge des utilisateurs qui n'ont pas particulièrement de compétence en la matière et dont ce n'est pas le métier.[5]

Sécurité d'accès

.Les points d'entrée sur l'ensemble du système informatique étant plus nombreux que sur un système centralisé, il convient de mettre en place des outils gérant la confidentialité sur toutes les entités communiquant à travers le réseau à tous les niveaux (réseau, SGBD/R, application). L'utilisation de mot de passe constitue la méthode la plus courante. Au niveau réseau, les mots de passe permettent de contrôler l'accès au réseau mais aussi d'associer à une catégorie d'utilisateur des droits et des privilèges particuliers, par exemple l'accès (en écriture ou modification) à un sous-ensemble de fichiers. Les principaux gestionnaires de réseaux tels que Lan Manager ou Netware proposent ces mécanisme de protection.

2.7. Evolutions de l'architecture coopérative

2.7.1. La recherche

La recherche est très active sur les environnements systèmes répartis : deux projets, peuvent être cités en exemple, ce sont les projets Gothic [5] et Guide [4] [26]. Leur objectif est d'intégrer un langage, permettant la programmation d'applications distribuées, à un système d'exploitation. Ces langages reposent sur des modèles objets et permettent la création d'objets sur toutes les unités de traitement se trouvant sur le réseau. L'accès aux objets répartis est totalement transparent pour les applications et se trouve à la charge du système. Le langage mis à la disposition des développeurs se comporte comme si les objets accédés étaient situés localement. Il est donc facile à utiliser et l'application peut être transférée sur une autre unité de traitement sans aucune modification.

Le choix de langage orienté objet est justifié par les raisons suivantes [4]:

. encapsulation des données avec le code ce qui permet une réutilisation du code commun pour la gestion des données,

. indépendance de l'interface développeur et de l'implémentation système ce qui permet le développement d'applications en milieu distribué hétérogène.

Les versions expérimentales de ces systèmes tournent sur des stations de travail UNIX se trouvant sur un réseau local.

2.7.2. Les outils en informatique de gestion

Dans l'industrie, il existe encore peu d'expérience de développement et de mise en oeuvre d'application client-serveur. Des expériences sont plus nombreuses mais récentes dans les banques et les administrations.

Les outils de développement disponibles sur le marché n'offrent pas toutes les fonctionnalités souhaitées et le développement d'application client-serveur en informatique de gestion relève encore essentiellement d'une démarche expérimentale. Cependant les annonces des fournisseurs, les travaux en cours des organismes de normalisation, et les recherches en cours font espérer de rapides progrès.

CGI annonce son produit PACLAN/NT (Cf.2.5.1), il apportera les outils de conception et de développement spécifiques à l'architecture client-serveur[9].
De nombreux constructeurs annoncent l'implémentation de l'architecture normalisée de l'OSF : DCE. C'est le cas de Hewlett Packard qui annonce que l'environnement de développement Encina de la société Transarc (qui est à la norme DCE)[36], sera distribué avec toutes les machines à partir de 1994, et qui apportera des outils permettant de développer facilement et d'exploiter des applications distribuées en environnement hétérogène.
Les utilisateurs seront donc assurés (à priori) d'une certaine normalisation permettant une inter-opérabilité globale.

Le coût des matériels diminue et leur puissance est croissante.
Les réseaux locaux s'améliorent en fiabilité et performance. La compatibilité entre les différents protocoles est aujourd'hui réelle et ne pose plus de difficulté, du moins pour les couches les plus basses.

L'architecture client-serveur et plus largement les architectures coopératives sont certainement destinées à un avenir prometteur et permettront une informatique plus souple, plus facile à utiliser pour les utilisateurs finaux et moins coûteuse.

Pour nous permettre d'évaluer concrètement les avantages et les difficultés de mise en oeuvre d'une architecture client-serveur, nous avons développé le projet pilote PROSTIVE (Cf. Chapitre 4) mettant en oeuvre les outils choisis dans le chapitre suivant.

CHAPITRE 3.

CHOIX DES OUTILS

3.1. Introduction

Le projet ANAIS a pour objectif de spécifier les techniques et les outils appropriés (AGL) pour produire des applications coopératives ou traditionnelles en respectant au maximum les choix qui ont été faits par les deux autres projets SIGMA et HERMES. De ce fait, le choix a été limité aux systèmes de bases de données relationnelles et aux outils de développement (AGL).

Le projet devra donc fournir des normes et standards pour le développement et la conception des applications. En effet, le respect de normes communes est fondamental dans le contexte d'architecture ouverte pour permettre l'évolution des applications vers d'autres types de serveurs.

L'architecture applicative doit être dissociée de l'architecture technique. Une même application doit pouvoir fonctionner soit sur une machine unique, soit dans un contexte coopératif.

La localisation des machines de type serveur ainsi que la répartition de l'applicatif entre les machines de type serveur et les machines de type client doivent être évolutives pour une même application.

3.2. Choix du système de base de données relationnelle

3.2.1. Introduction

Le choix d'un système de base de données relationnelle permet d'assurer l'indépendance entre les développements et les bases de données.

La normalisation du langage de description et de manipulation des données a permis le développement des applications client-serveur. Les besoins de portabilité des applications sont résolus en utilisant un langage normalisé d'accès aux données, le SQL (le langage SQL est un langage défini par l'ANSI et l'ISO).
Il est donc important de développer les accès aux bases de données par des requêtes SQL conformes aux normes.

L'utilisation du SQL normalisé représente un des critères de choix, doté d'un poids très fort, du SGBD/R. C'est le seul moyen de garantir l'indépendance des applications par rapport au moteur SGBD/R.

3.2.2. L'offre du marché

Dans un premier temps une étude de l'offre du marché a été faite. Deux types de produit sont disponibles : les SGBD/R proposés par les constructeurs de machines et les SGBD/R multi-constructeurs fournis par des sociétés de services. Les trois principaux SGBD/R multi-constructeurs nous ont été présentés : Oracle, Sybase, Ingres, tous ces systèmes fonctionnant sur des plates-formes multiples. Ils présentent, pour le contexte Merlin Gerin, un coût élevé et ne proposent pas d'avantage important par rapport aux SGBD/R constructeurs : **Oracle** propose une solution logicielle complète constituée d'un atelier de génie logiciel et . d'un SGBD/R, mais la solution est totalement fermée : en amont l'AGL n'est interfacé qu'avec la base de données Oracle, en aval, Oracle utilise un protocole réseau spécifique.
Sybase et **Ingres** proposent des concepts nouveaux tels que triggers et procédures cataloguées, mais ils sont non normalisés.
Pour ces trois produits un runtime (partie du produit assurant la communication) payant est nécessaire sur chaque serveur et chaque client.

Ces trois solutions sont chères pour le groupe compte tenu du nombre de sites à installer.

Les SGBD/R constructeurs tels que ALLBASE de HP, DB2 d'IBM sont aujourd'hui performants, ils sont à la norme SQL et suivent les évolutions de cette norme, ils sont beaucoup mieux intégrés aux systèmes d'exploitation que les SGBD/R multi-cibles et ne nécessite pas de runtime payant sur les clients et serveurs.

Il a donc été décidé par l'ensemble des membres du projet ANAIS de choisir les SGBD/R constructeurs correspondant aux machines retenues dans le cadre du projet HERMES (Cf.1.6.2).

3.2.3. SGBD/R retenus

3.2.3.1. Serveur HP3000 / MPEXL

Le serveur applicatif choisi étant le HP 3000/MPEXL, le système de base de données relationnelle **HP Allbase SQL** est retenu sur ce serveur.
Ce produit est à la norme SQL. Il est aussi performant que la base de données Réseau HP (Turbo Image), à condition toutefois que le modèle physique soit bien conçu et que les données soient bien gérées.
Il est livré **gratuitement** avec toutes les machines, c'est le critère qui a été déterminant dans le choix. En effet le SGBD/R retenu devra être installé dans toutes les filiales, une cinquantaine à ce jour , ce qui a un impact économique non négligeable.

3.2.3.2. Autres serveurs

Pour le serveur **IBM** le choix est DB2, il existe déjà de nombreuses applications développées avec ce système sur la machine centrale.

Pour les micro-ordinateurs, le SGBD/R retenu est SQLBase de Gupta. C'était le seul à l'époque du choix à fonctionner aussi bien en DOS que sous OS2.

La portabilité des applications au niveau SQL, pour les trois produits retenus, a été vérifiée et ne pose pas de problème majeur.

3.3. Choix de l'AGL

3.3.1. Les motivations

Le choix d'un outil s'est imposé pour répondre aux objectifs fixés, à savoir :

. recherche de la productivité des concepteurs et réalisateurs,

. réduction des coûts de développement,

. amélioration de la qualité du code produit, diminution de la maintenance,

. génération de code sur plusieurs machines cibles.

Un AGL doit permettre également une automatisation la plus complète possible du cycle de développement des applications, et procurer un support méthodologique.

Rappel de l'objectif principal du projet ANAIS :

proposer un AGL permettant de générer des applications en mode coopératif et traditionnel (mini + terminaux) avec les composants suivants :

. poste client sous DOS/Windows 3 ou terminal,

. serveur applicatif HP MPEXL,

. SGBD/R Allbase SQL,

. réseau préconisé par HERMES,

. utilise et génère du L3G (COBOL).

Devront été éliminés les produits utilisant un Langage de 4ème génération (L4G). Ce type de langage n'est pas normalisé, il est non portable, non évolutif, il est donc considéré comme jetable à terme.

3.3.2. La démarche

Une première étape a consisté à étudier **l'offre du marché**. A l'issue de cette étude deux produits ont été retenus. Un tableau de critères de choix a été envoyé à chaque fournisseur pour que ceux-ci puissent donner tous les éléments nécessaires à la comparaison des offres.
Ce tableau de critères se trouve en annexe 1.

Une **synthèse** des avantages et inconvénients des offres a permis de décider de **tester deux produits sur un projet pilote**.

3.3.2.1. Etude de l'offre du marché

A l'issue de cette étude les produits suivants ont été éliminés :

Oracle/case	n'est pas indépendant du SGBD/R. Fonctionne uniquement avec le SGBD ORACLE.
Powerhouse	n'utilise pas le SQL et accède physiquement aux bases de données constructeur, ce qui ne garanti pas la pérennité du produit. Ce produit est accessible par un L4G.
IEW	Ernst and Young : architecture cible d'exécution des applications uniquement IBM.

Les fournisseurs répondant le mieux au cahier des charges sont :

1) Arthur Andersen avec les produits :

. Foundation qui permet d'automatiser la génération d'applications "traditionnelles" c'est à dire avec une architecture mini + terminaux sur machine IBM/CICS.

. Foundation coopératif qui permet de générer des applications en mode coopératif avec des postes clients OS2 ou DOS et des serveurs OS2 ou IBM/CICS.

La transposition de la partie serveur sur HP MPEXL, ainsi que la possibilité de développer des applications traditionnelles sur HP ont été demandées par Merlin Gerin, puisque cette plate-forme n'est pas au catalogue Arthur Andersen.

2) CGI avec le produit :

. Paclan qui permet de générer des applications sur HP ou 30 autres machines cibles. Le produit doit évoluer pour permettre de générer des applications en mode coopératif poste client sous DOS ou OS2 et serveur sous HP MPE XL ou UNIX. En attendant CGI propose pour développer la partie client un produit d'une autre société Nat Systèmes, NS-DK qui pourra être interfacé avec Paclan.

3.3.2.2. Synthèse

Aucun des deux produits retenus ne satisfait parfaitement aux exigences de Merlin Gerin.

Les points forts de CGI :

. L'offre est disponible pour une trentaine d'architectures cibles, ce qui est stratégique pour Merlin Gerin.

. Très fort partenariat avec HP tout en n'excluant pas les autres offres.

. Outils de gestion des sources et des versions dès le niveau conceptuel.

. Le référentiel paraît techniquement meilleur.

. Il est possible de passer tous les développeurs sur Paclan dès aujourd'hui.

Les points faibles de CGI :

. Pas d'expérience du mode client-serveur.

Les points forts de Arthur Andersen :

. En avance pour le mode coopératif.

Les points faibles de Arthur Andersen :

. Pour les applications traditionnelles (mini + terminaux) la solution sera spécifique à Merlin Gerin et risque de ne pas être cohérente, les outils seront différents, avec Foundation Coopératif pour les applications client-serveur.

Il a été décidé de tester l'offre de CGI en premier puisque c'est l'offre qui couvrait le plus de critères au moment où le choix a été fait.

CHAPITRE 4.

PROJET PILOTE

4.1. Objectif du projet pilote PROSTIVE

Le principal objectif du projet pilote PROSTIVE est de montrer la capacité de l'AGL Paclan à développer des applications client-serveur mais également à vérifier que l'utilisation de l'AGL amène les avantages pressentis, à savoir : productivité des développeurs et concepteurs, qualité des applications, portabilité des applications.
L'expérimentation de la conception et du développement de ce nouveau type d'application client-serveur est bien sûr un argument de poids en faveur de la réalisation de ce projet pilote.

Pour que le test soit plus probant et plus proche de la réalité, une partie d'un projet en cours de spécification chez Merlin Gerin (projet STIVE) a été extraite et développée dans le cadre du projet pilote.

4.2. Présentation du projet STIVE

L'objectif du projet STIVE (Système Tiers Vente) est de spécifier et de construire un système d'information commercial dynamique capable de procurer aux forces de vente du groupe Merlin Gerin en France, une partie des outils de suivi et d'analyse clientèle, dont elles ont besoin pour conduire leurs actions sur le terrain. La petite partie du projet STIVE , qui a été extraite est la fonction : *gérer tiers*.

La définition d'un tiers pour Merlin Gerin étant : toute personne, établissement ou société ayant une influence plus ou moins directe sur les achats de produits ou de services au groupe Merlin Gerin. Les fournisseurs ne sont pas concernés.

4.2.1. Architecture technique cible

Les sites concernés par le projet STIVE sont répartis géographiquement selon le schéma suivant :

- 1 site central France, données opérationnelles + besoins de type infocentre,
- 4 sites régions, besoins de type infocentre
- 40 sites agence, données opérationnelles + besoins de type infocentre,
- 10 postes de travail autonomes (portables) en moyenne par site agence.

Les données sont réparties de la façon suivante :

- 30% des données d'une agence sont communes à toutes les agences,
- 70% des données des agences d'une région sont nécessaires à la région.

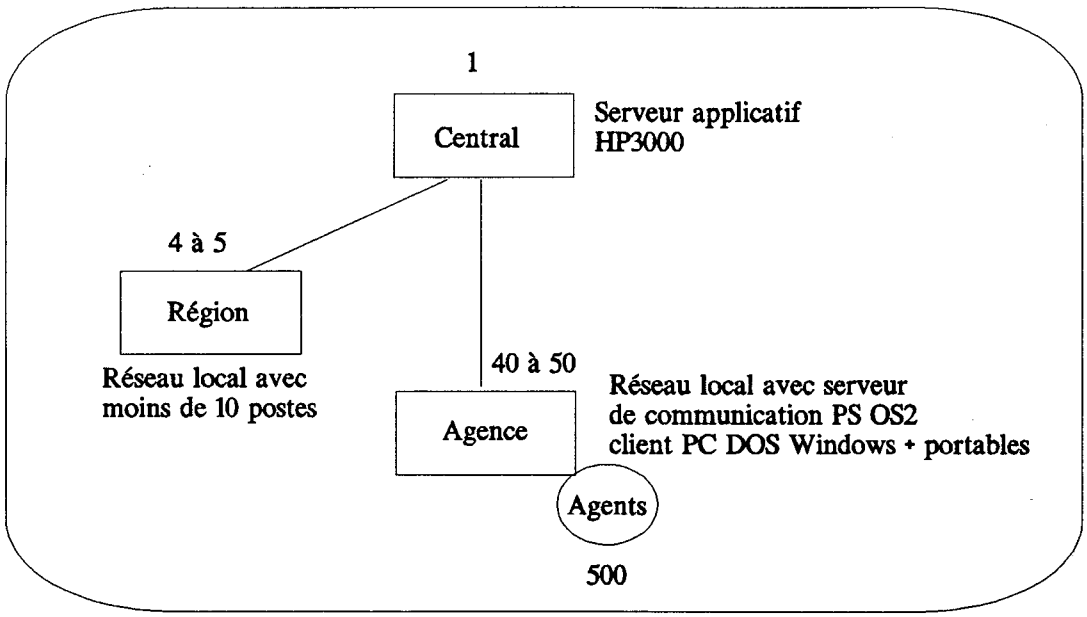


Figure 4.1 Architecture technique cible du projet STIVE.

Les utilisateurs ont exprimé les demandes suivantes :

- une bonne ergonomie des postes de travail avec interface graphique,
- la délocalisation des données possible pour rendre les agences le plus autonome possible,
- le système mis en place en France doit être portable à l'étranger, sur des machines cibles qui peuvent être différentes,
- l'application doit être la même en agence et sur les portables destinés aux agents.

L'ensemble de ces besoins ont conduit à définir une **architecture technique de type client-serveur** pour ce projet pour les raisons suivantes :

- L' utilisation de l'interface graphique Windows est la principale raison du choix de l'architecture client-serveur, en effet, on ne sait pas aujourd'hui développer dans le contexte Merlin Gerin des applications traditionnelles Cobol avec interface graphique.
- La souplesse de l'architecture qui permet de délocaliser progressivement les données et les traitements qui y sont attachés : dans un premier temps, pour répondre au besoin urgent des agents commerciaux, l'application sera développée, en mode client-serveur sur des postes OS2 portables autonomes qui joueront à la fois le rôle de client et de serveur. L'architecture client-serveur permettra ensuite de déporter, dans un deuxième temps, la partie serveur sur la machine centrale France, et la partie client sur les postes de travail des agences, sans modifier l'application.
  La délocalisation des données se fera ensuite progressivement sur les sites régions.

Pour le projet pilote, il a été décidé de tester l'application développée dans les deux environnements qui suivent :

- un PC DOS pour les clients et un serveur OS2,
- un monoposte PS OS2 portable (car seul l'OS2 permet le multitâche aujourd'hui) qui joue le rôle du client et du serveur.

La phase d'analyse détaillée du projet STIVE n'étant pas commencée lors du démarrage du projet pilote, les développements faits dans le cadre de ce projet pilote ont été considérés comme "jetables" . Les spécifications n'étaient pas complètes (pas de maquette d'écran par exemple) et certaines options ont été prises dans le cadre du projet pilote sans concertation avec les utilisateurs.

4.3. Présentation des outils

La solution que CGI a préconisée était donc d'utiliser l'AGL Paclan pour développer la partie serveur et le produit NS-DK pour développer la partie client. (Cf.3.3.2.1)

4.3.1. L'AGL

4.3.1.1. Les outils de l'AGL

Paclan prend en charge la gestion du cycle de vie des applications depuis la phase de conception jusqu'à la phase de maintenance.
L'ensemble des informations manipulées pendant tout le cycle de vie des applications est stocké dans une base unique : la base de spécifications ou dictionnaire.
Les spécifications sont définies de manière unique et sont liées par des références croisées automatiquement mises à jour.
Les spécifications sont organisées en un modèle entités/relations et reposent sur la méthodologie Merise.

Paclan a une interface utilisateur en mode caractère de type écran 3270.
L'AGL est également disponible sur station de travail sous Windows ou Presentation Manager.
Deux modules sont intégrés dans cette station de travail : Pacdesign et Pacbench.

Pacdesign est le poste de travail du concepteur et **Pacbench** est le poste de travail spécialisé pour la réalisation.

Page suivante se trouve un schéma permettant de situer ces outils par rapport à la démarche méthodes Merlin Gerin

.

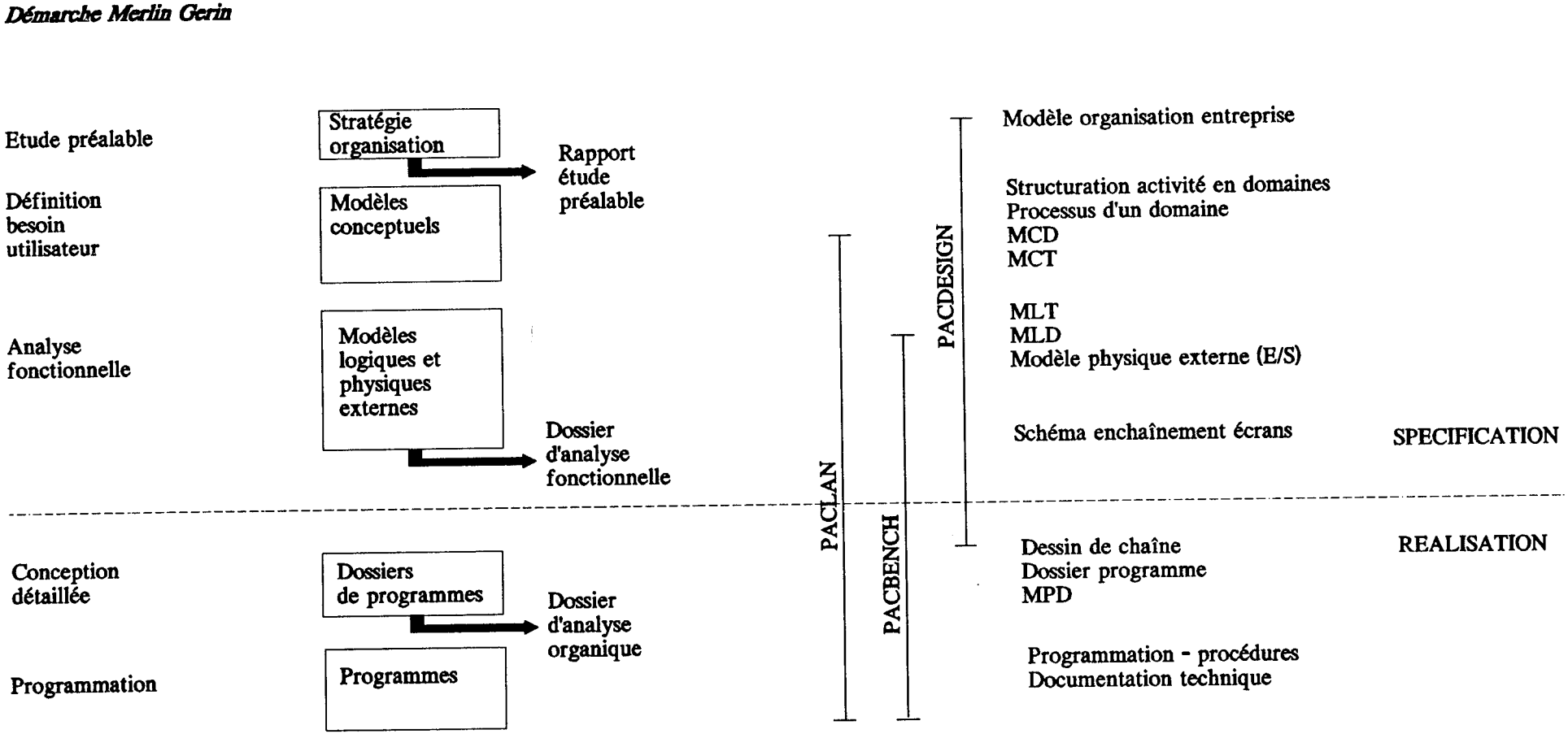


Figure 4.2 Couverture de la démarche méthodes MG par l'AGL.

4.3.1.2. Architecture client-serveur de l'AGL

L'AGL utilise pour son fonctionnement propre une architecture client-serveur.
La station de travail se comporte en client du serveur sur lequel se trouve la base de spécifications. Les rôles respectifs du client et du serveur sont les suivants :

. le client :

- gère entièrement et automatiquement le recours au serveur, qui est totalement "transparent" pour l'utilisateur,
- utilise temporairement ses capacités de stockage pour les éléments ne mettant pas en jeu la cohérence de la base de spécifications.

. le serveur :

- gère l'intégrité de la base de spécifications,
- assure la cohérence permanente des données,
- permet le partage des données entre plusieurs clients,
- est chargé des traitements de masse tels que archivage, sauvegarde...

4.3.1.3. Plate-forme de développement du projet pilote

Page suivante se trouve le schéma de l'architecture de la plate-forme de développement utilisant conjointement l'outil de développement sous Windows, NS-DK, et l'AGL Paclan.

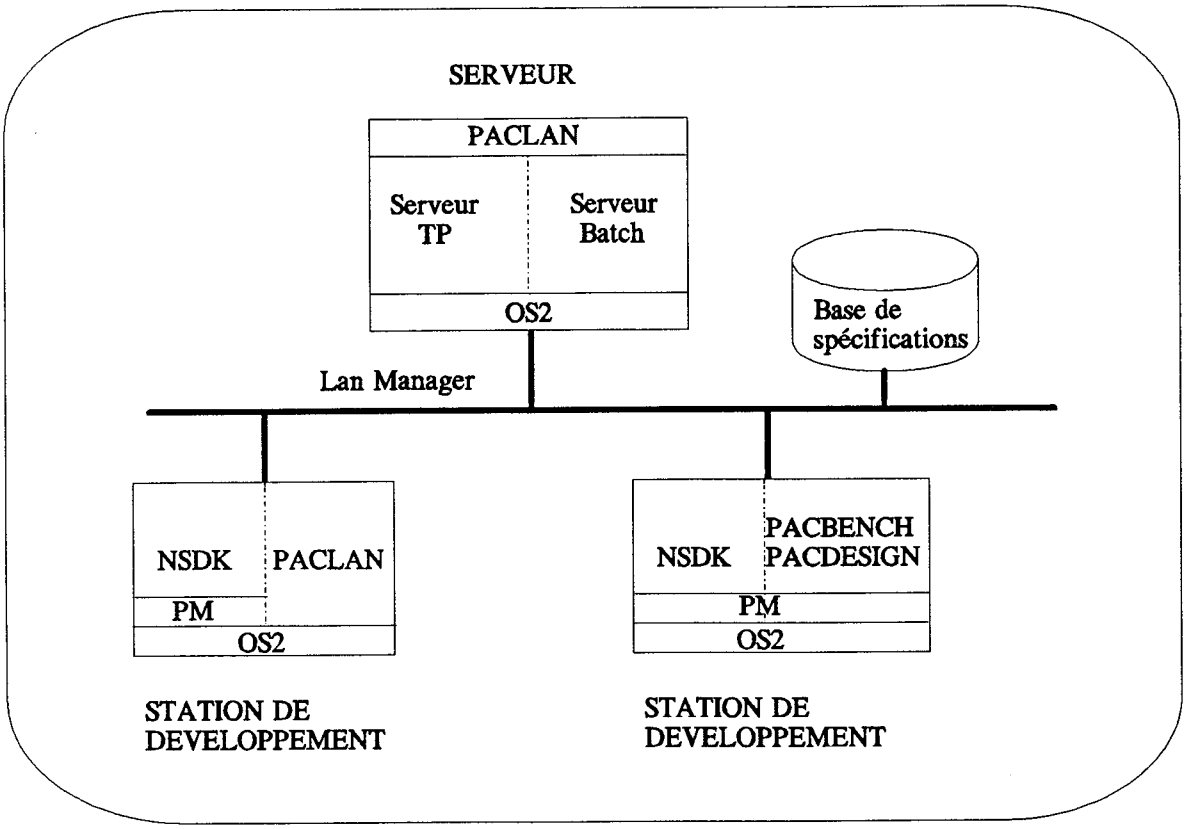


Figure 4.3 Architecture de la plate-forme de développement pour le projet pilote.

4.3.1.4. Langage structuré de l'AGL

L'AGL génère automatiquement les traitements standards tels que gestion de la conversation (affichage des écrans et récupération des saisies utilisateurs), accès aux fichiers...

Les traitements spécifiques fonctionnels sont à insérer dans une structure de squelette Cobol proposée par défaut.

Pour introduire les traitements spécifiques il faut utiliser le langage structuré Pacbase qui est composé de codes raccourcis Cobol. Exemples : M pour Move, IT pour If Then.

Le temps d'apprentissage est long, de l'ordre de 3 à 6 mois pour maîtriser totalement l'AGL, pour la partie développement, mais les gains en temps de développement et de maintenance sont importants.

Selon les clients que nous sommes allés voir, Descours et Cabaud - Lyon, Caisse d'Epargne (CT4R) - Lyon, Caisse Régionale du Crédit Agricole (CRCA) - Lyon, lorsque l'AGL est maîtrisé, les gains en temps de développement peuvent être de 20 à 30% et les gains en maintenance de 50 à 80%.

4.3.2. Développement de la partie client

NS-DK est un produit qui offre des outils permettant de développer des applications, sous DOS ou OS2, ayant une interface graphique de type Windows ou Presentation Manager.
Ces principales fonctionnalités sont :

- dessiner les écrans : le dessin se fait très facilement à l'aide d'une boîte à outils accessible avec la souris et ne nécessite aucune programmation.
- programmer les traitements : le langage de programmation est spécifique à NS-DK. C'est un mélange de C , de Cobol et de Basic. Il est très facile à apprendre, un journée de formation suffit pour démarrer. Il est possible d'ajouter des instructions C ou Cobol au milieu d'instructions NS-DK.
  Le comportement des contrôles et des fenêtres est géré par programmation événementielle. A chaque contrôle sont associés des traitements correspondant à des événements donnés intervenant sur le contrôle.
- paramétrer les enchaînements des écrans : c'est un excellent outil de maquettage.

4.4. Organisation du développement du projet

L'équipe de développement était constituée de deux personnes de Merlin Gerin (une personne faisant partie du groupe de projet STIVE et moi-même) et d'une personne de CGI chargée de développer le moniteur spécifique pour OS2 et des utilitaires particuliers en C.
Une seconde personne de CGI avait le rôle de chef de projet.
Une formation rapide de 4j sur Paclan et 1j sur NS-DK nous a été dispensée.

Toutes les phases de développement d'un projet ont été traitées avec l'AGL Paclan et le produit NS-DK pour développer la partie client.
Il n'a pas été utilisé de méthode de conception spécifique au type d'application client-serveur . Cependant, un de nos objectifs était de nous servir de cette première expérience pour établir des recommandations pour le projet complet STIVE.
Les normes de développement ont été construites pendant le développement, CGI et Merlin Gerin n'ayant pas d'expérience de ce nouveau type de développement.

La solution étant composée de deux produits qui ne sont pas intégrés, Paclan et de NS-DK , des ponts ont dû être développés entre eux pour éviter des ressaisies. Toutes les données nécessaires au développement de l'interface client ont été transférées sans problème sous NS-DK, ce produit étant très ouvert.

4.4.1. Modélisation

La modélisation avait déjà été faite sous l'outil IEW, outil de modélisation qui est le standard pour l'instant chez Merlin Gerin. Seule la modélisation des données a été traitée. Les diagrammes d'action saisis sous IEW n'ont pas été retranscris sous Paclan.
MCD et MLD ont été ressaisis manuellement sous Pacdesign. Les modèles IEW n'ont pu être transférés directement, il a fallu une demi-journée de réflexion pour transcrire ces modèles de type Yourdon sous Merise. Une automatisation paraît difficilement réalisable pour passer de modèle conceptuel à modèle conceptuel, les formalismes des deux méthodes étant très différents, (les cardinalités sont symbolisées de façons différentes, les entités associatives d'IEW n'existent pas sous Merise).
Un pont au niveau des modèles logiques paraît plus facile à réaliser.

Les MCD et MLD de Prostive ainsi qu'un extrait du Dossier d'Analyse Fonctionnelle sont fournis en annexe 4.

4.4.2. Les ponts entre Paclan et NS-DK

Les attributs du MLD saisis sous Paclan sont générés sous forme de *template* pour NS-DK. Une *template* est un groupe de contrôles, par exemple un libellé et une zone de saisie, une combo-box ou des radio-boutons.

Un attribut donne naissance à plusieurs *templates* en fonction des caractéristiques d'affichage qui peuvent être : champ protégé ou pas, zone de saisie obligatoire ou non, libellé long, court, sans libellé.

Les segments de données sous NS-DK ont été générés automatiquement à partir des segments définis dans le dictionnaire Paclan.

4.4.3. Le moniteur

Le moniteur transactionnel (Cf. 2.4.1.4) est chargé de recevoir les demandes de service en provenance des différents clients, de les exécuter et de renvoyer la réponse au client correspondant.
La machine OS2 n'ayant pas de moniteur transactionnel, un moniteur a été développé pendant les deux mois de test par CGI. La communication entre les clients et le moniteur se fait par l'intermédiaire du mécanisme de Named Pipes fourni par Lan Manager (Cf.2.3.2.1).

C'est le point faible de la solution. Celui-ci a été développé rapidement pour nos propres besoins et il est assez "rustique". Il nécessite quelques améliorations pour être opérationnel sur un site d'exploitation.
Le moniteur crée un fichier résultat sur le serveur, le nom de ce fichier est envoyé au client correspondant. Le client récupère le fichier résultat au moyen des fonctions standards de partage de fichier de Lan Manager.

4.4.3.1. Critiques sur le fonctionnement du moniteur

Le moniteur n'est pas **performant**. Il teste les pipes ouverts par les clients les uns après les autres et traite les demandes de services séquentiellement, ce qui peut générer un temps de réponse important pour le client qui est le dernier de la file d'attente lorsque le nombre de clients est important.

Le moniteur n'utilise pas toutes les potentialités du système multitâches OS2. Il faudrait qu'il ouvre un thread (sous-tâche déclenchée par un process, qui s'exécute simultanément et partage le

même espace mémoire que le process père), par pipe ouvert ce qui permettrait de traiter simultanément les demandes de services des postes clients, et d'obtenir un temps de réponse équivalent quel que soit le client.

Le moniteur ne gère pas la **sécurité** d'accès au serveur, il n'y a aucune vérification des accès au serveur ni aux services.
Le moniteur ne vérifie pas la **compatibilité des programmes clients avec les programmes serveurs**, ce qui n'est pas acceptable pour un déploiement important des applications sur de nombreux sites.

Toutes ces améliorations ont été demandées à CGI.

4.4.4. Identification des fenêtres et des services

Une fenêtre principale a été créée par entité du MCD.
A partir de ces fenêtres principales, des fenêtres filles peuvent être ouvertes à partir d'une option du menu ou d'un bouton "Détail".

Les fenêtres développées sont les suivantes :

. généralités d'un tiers,
. deux fenêtres pour les détails de tiers particuliers,
. critères de recherche,
. coordonnées d'un tiers,
. relations tiers-tiers.

Voir en annexe 2 une copie des fenêtres "coordonnées d'un tiers" et "relation tiers-tiers" .

Dans un premier temps, la liste des événements possibles par fenêtre a été définie, c'est à dire que les actions possibles à l'utilisateur, les options du menu, les boutons poussoir affichés ont été recensés.
Des procédures ont été ensuite associées à ces événements.

Certaines de ces procédures font appel à des services du serveur. C'est le cas pour les procédures de mise à jour et de recherche de données.
Un service de mise à jour par entité du MLD a été développé, au minimum.
Un service de recherche d'une occurrence et un service de recherche d'une liste d'occurrences ont été développés pour certaines fenêtres.

Exemple de la fenêtre Relation tiers-tiers :

Evénement	Procédure	Service
Click sur option Relation tiers-tiers du menu	CT-Init	CTRECHERCH
Click sur bouton Ajouter	CT-I-Ajouter	CTRECHERCH CTMISEAJOU
Click sur bouton Modifier	CT-I-Modifier	CTMISEAJOU
Click sur bouton Supprimer	CT-I-Supprimer	CTMISEAJOU
Click sur bouton Rechercher	CT-I-Rechercher	CTRECHERCH
Click sur bouton OK	CT-I-Close	

Le service CTRECHERCH permet de rechercher une liste d'occurrences.
Le service CTMISEAJOU permet de mettre à jour les données ou de rechercher une seule occurrence. C'est le code action, passé en paramètre au serveur, qui détermine le type d'accès aux données (C création, M modification, A suppression, R recherche).

Exemple de la fenêtre Coordonnées :

Evénement	Procédure	Service
Click sur bouton Coordonnées de la fenêtre Tiers	CO-Init	
Click sur bouton Ajouter	CO-I-Ajouter	COMISEAJOU
Click sur bouton Modifier	CO-I-Modifier	COMISEAJOU
Click sur bouton Supprimer	CO-I-Supprimer	COMISEAJOU
Click dans la combo-box contenant le type de coordonnée	CO-I-TCclick	COMISEAJOU
Click sur bouton OK	CO-I-Close	

4.4.5. Localisation des données

La liste mémorisée des tiers particuliers à chaque agent a été rapatriée sur les postes clients. Ces données sont stables dans le temps et ne sont accédées qu'en lecture. Pour éviter un accès long au serveur lors du démarrage de l'application, ces données ont été recopiées sur les clients.
Toutes les autres données sont stockées sur le serveur à l'aide du SGBD/R SQLBase.

4.4.6. Localisation des traitements

Les traitements ayant accès à la base de données située sur le serveur ont été développés avec Paclan sur le serveur. C'est le cas des traitements de mise à jour et de recherche. Tous les autres

développements de l'application ont été développés sur le client, à savoir l'affichage, le contrôle des données saisies et l'accès aux données locales.

Cette application est du type *serveur coopératif* (Cf.2.4.1.3)

4.4.7. Le développement

Les procédures rattachées à chaque fenêtre se trouvent dans des librairies séparées, une par fenêtre principale, c'est à dire une librairie par entité.

Une librairie par projet regroupe les procédures et les déclarations de variables communes à toutes les fenêtres du projet.

Une librairie commune à tous les projets, regroupe toutes les procédures de communication avec le serveur et les utilitaires communs.

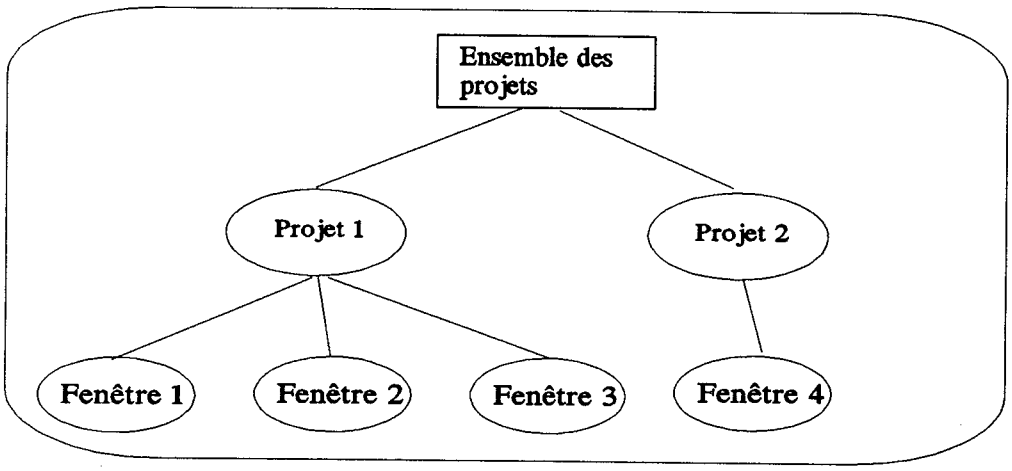


Figure 4.4 Arborescence des librairies.

Nous avons défini l'architecture logicielle suivante qui a été découpée en trois couches : applicative, enchaînement, session. Chaque couche du client et du serveur est détaillée dans les pages qui suivent.

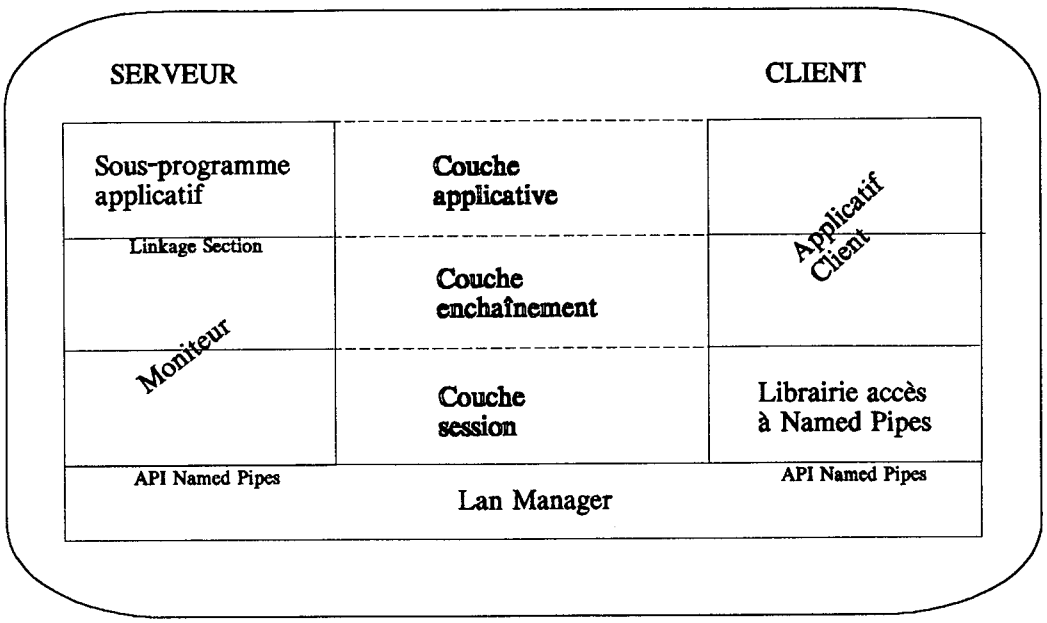


Figure 4.5 Architecture logicielle.

4.4.7.1. Les couches du client

Couche session du client

Les procédures suivantes se trouvent dans une librairie commune à tous les projets et utilisable par les couches supérieures :

. **Connexion** : cette procédure permet au client de demander une connexion au serveur. Elle est utilisée à chaque début de conversation avec le serveur. Elle renvoie un numéro de session à utiliser pour tous les appels ultérieurs à la couche session.
La procédure de connexion est libre, deux options sont proposées :
- première solution : le client demande une connexion à chaque appel de service du serveur.

- deuxième solution, le client demande une connexion au début de l'application et coupe la communication à la fin.

La première solution n'est pas envisageable dans le cas de serveur très éloigné des postes clients, puisque cette solution provoque une surcharge du réseau.
Avec la deuxième solution, beaucoup de clients risquent d'être connectés simultanément ce qui pourra nécessiter des ressources sur le serveur plus importantes.
Il est à noter que Lan Manager ne supporte pas plus de 250 connexions simultanées.
Dans le cas du projet pilote, nous avons opté pour que le client se connecte à l'initialisation de l'application, le nombre de postes clients dans un premier temps étant limité (fonctionnement monoposte au départ).
Appel à la fonction DosConnect (API Named Pipes)

. **Ecriture** : permet d'envoyer vers le serveur un bloc de données. Les paramètres sont les suivants :

- numéro de session (renvoyé lors de la connexion),
- taille des données à envoyer.

Les données auront été préalablement stockées dans un buffer d'écriture.
Appel à la fonction DosWrite (API Named Pipes)

. **Lecture** : permet de lire les données envoyées par le serveur. Les paramètres sont les suivants :

- numéro de session en entrée (renvoyé lors de la connexion),
- taille des données reçues en sortie.

Les données reçues sont stockées dans un buffer de lecture. La procédure de lecture effectue une boucle jusqu'à écoulement d'un temps limite ou jusqu'à ce qu'une réponse soit disponible, elle rend ensuite la main au processus appelant.

Appel à la fonction DosRead (API Named Pipes)

. **Déconnexion** : coupe la communication entre le client et le serveur.

Le paramètre est :

- numéro de session.

Appel à la fonction DosDisconnect (API Named Pipes)

Couche enchaînement du client :

Une seule procédure REQUETE permet d'envoyer une demande de service au serveur. Les paramètres sont les suivants :

- code du service à exécuter en entrée,
- nombre de caractères des données en entrée,
- nombre de caractères des données en sortie.

Les données en entrée et en sortie sont stockées dans des buffers lorsqu'il s'agit de données constituant une seule occurrence.
Lorsqu'il s'agit de listes d'occurrences, celles-ci sont stockées dans un fichier sur le serveur.
Cette procédure est synchrone pour les demandes de service sur une occurrence (elle garde la main tant que le serveur n'a pas répondu), elle est asynchrone pour les demandes de service portant sur une liste d'occurrences, l'utilisateur n'est pas bloqué en attendant la réponse du serveur.

Couche applicative du client

Les procédures de la partie client ont été découpées en plusieurs couches :

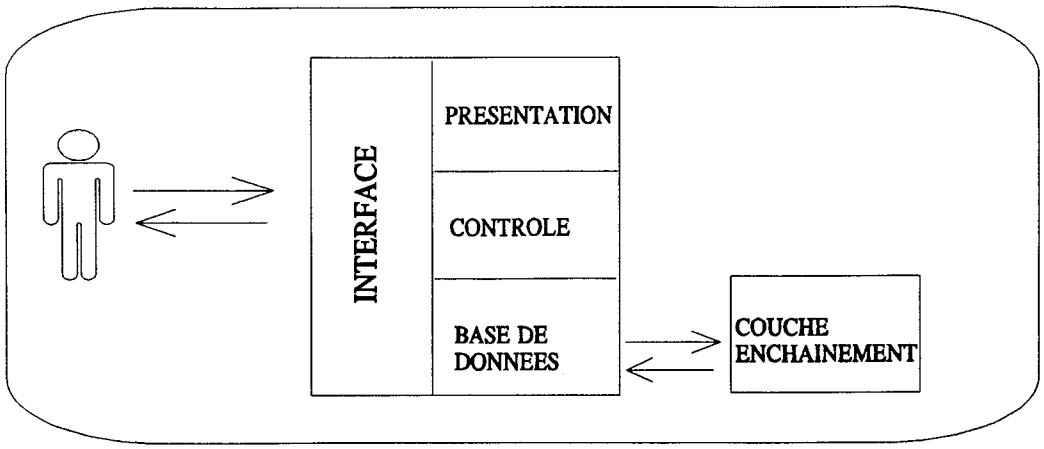


Figure 4.6 Couche applicative de la partie client.

. Présentation : couche chargée des initialisations des fenêtres, des sauvegardes des données dans des buffers, les données saisies ou modifiées n'étant effectivement envoyées au serveur que lors de la validation de la fenêtre principale.

. Contrôle : couche chargée du contrôle des saisies des utilisateurs. Une fonction par fenêtre est développée.

. Base de données : couche chargée de l'interface entre les buffers (segments) et la base de données. Deux types de requêtes ont été développés :

- la mise à jour comprenant la création, la modification, la suppression dans la base de données,

- la recherche qui rend soit un segment contenant une occurrence recherchée, soit une liste d'occurrences sélectionnées dans un fichier situé sur le serveur.

. Interface : couche chargée de l'ouverture et de la fermeture des fenêtres, elle traite les actions de l'utilisateur et appelle les procédures des autres couches. Il y a une fonction par item de menu et une fonction par bouton poussoir.

4.4.7.2. Les couches du serveur

Couche session du serveur

Le moniteur se met en écoute des demandes de connexion au serveur. Il gère une table des sessions en cours.

. **Lecture** : il lit tous les pipes ouverts par les clients à tour de rôle. Lorsque le résultat de la lecture est positif, cas où le client a envoyé des données, il mémorise ces données dans un buffer de lecture. Ce buffer est unique pour tous les utilisateurs. Le nombre de caractères présents dans le buffer est stocké à part.
Appel à la fonction DosRead (API Named Pipes).

. **Ecriture** : à chaque itération de sa boucle principale le moniteur examine un indicateur, contenant le nombre de caractères à envoyer vers le client. Si ce nombre est non nul, les données correspondantes, contenues dans un buffer d'écriture, sont envoyées vers le client au moyen de la fonction DosWrite (API Named Pipes). Le numéro de la session courante est lue dans la table des sessions.

. **Déconnexion** : le moniteur supprime la session correspondante de la table des sessions en cours. Il coupe la session en appelant la fonction DosDisconnect (API Named Pipes).

Couche enchaînement du serveur:

Une table de correspondance codes requêtes - codes sous-programmes applicatifs permet de lancer le sous-programme correspondant au service demandé.

Dans la *Linkage Section* partagée avec le sous-programme, le moniteur charge les données du message reçu.
Après exécution du sous-programme, le moniteur écrit les données constituant la réponse (à partir de la *Linkage Section*) dans un buffer d'écriture et met à jour le nombre de caractères à envoyer au client. La couche session est chargée de renvoyer effectivement ces données au client.

Couche applicative du serveur

Les sous-programmes applicatifs constituant les services ont été développés avec l'AGL Paclan. Ils sont appelés par le moniteur qui leur transmet les informations suivantes par la *Linkage Section* :

- les données envoyées par la couche applicative client : ce sont des données pour mise à jour ou extraction,
- le contexte utilisateur qui permet de mémoriser des données utilisées lors de plusieurs exécutions du sous-programme ou de plusieurs sous-programmes différents.

Seules les couches applicatives du client et du serveur sont visibles pour le développeur. Les autres couches sont totalement transparentes pour lui.

4.5. Bilan

Le projet pilote a été réalisé avec succès dans les délais impartis, l'objectif a été atteint.
La solution préconisée par CGI, l'utilisation conjointe de l'AGL Paclan et du produit NS-DK, a montré sa capacité à développer des applications client-serveur.

Ce projet a été mené sans méthode de conception spécifique à l'architecture client-serveur, a confirmé qu'il était nécessaire avant de démarrer un projet réel à plus grande échelle, d'établir des normes et d'utiliser une méthode enrichie, plus adaptée à ce nouveau type d'application.

Une méthode de conception plus efficace doit prendre en compte la conception d'interfaces graphiques (organisation de la navigation entre les fenêtres, contenu des fenêtres par exemple) et la distribution des données et des traitements.

4.6. Suites

A l'issue de ce projet pilote une consultation à été lancée auprès de plusieurs sociétés de service pour la réalisation du projet complet STIVE.
Le coût de développement de ce projet en architecture client-serveur annoncé par ces sociétés de service était de 30% plus cher et demandait une charge supplémentaire de 20% en temps de développement, par rapport à une architecture traditionnelle. Ceci nous a confirmé, contrairement à ce qu'affirment les articles de presse et les animateurs des différents séminaires auxquels nous avons assistés ([20], [25], [1]), que le développement d'applications client-serveur est aujourd'hui plus coûteux que le développement bien maîtrisé des applications 'traditionnelles' centralisées.

Les raisons de ce coût plus élevé sont les suivantes :

- le développement de l'interface graphique est plus complexe parce que l'interface proposée à l'utilisateur est plus riche que les écrans de type caractères s'enchaînant de façon séquentielle,
- il nécessite la formation des développeurs qui connaissent tous le Cobol (langage aujourd'hui utilisé majoritairement dans les applications de gestion) mais peu la programmation événementielle nécessaire pour développer des interfaces graphiques sous Windows,
- la mise en oeuvre d'une nouvelle technologie et de nouveaux outils comporte des risques plus importants d'allongement des temps de développement,
- il est nécessaire de faire intervenir des compétences différentes plus nombreuses, au niveau réseau (LAN MANAGER), maquettage sous interface graphique (WINDOWS), systèmes d'exploitation (MS-DOS, OS2, HP3000),
- la société de service doit acquérir plusieurs machines (portables, serveur OS2, serveur HP).

Le choix de l'architecture client-serveur a donc été remis en cause pour le projet STIVE. Les utilisateurs (qui financent les projets chez Merlin Gerin) ont préféré choisir une architecture correspondant moins à leurs exigences mais moins coûteuse et comportant moins de risque à savoir une architecture centralisée sur serveur HP3000 et accessible par terminaux.

Le développement du projet pilote avec le deuxième AGL (FCP de Arthur Andersen) a donc été annulé, son principal point fort étant l'architecture client-serveur. Ce produit ne présente alors plus d'argument pour être retenu dans un contexte de développement d'applications à architecture centralisée, avec les contraintes Merlin Gerin. Seule la formation au produit a eu lieu pour nous permettre d'enrichir nos connaissances dans le développement des applications client-serveur, de

nous faire une idée précise du produit rapidement, et d'être plus critiques par rapport à l'offre de CGI.

Suite à ce projet pilote, l'AGL PACLAN de la société CGI a été retenu pour développer des applications traditionnelles multi-cibles à savoir HP3000 et AS400, qui sont les principaux serveurs du groupe Merlin Gerin. L'évolution annoncée de ce produit vers l'architecture client-serveur en 1993, nous permettra de remettre au goût du jour l'architecture client-serveur chez Merlin Gerin, puisque les outils annoncés devraient faciliter le développement de telles applications et donc en réduire le coût.
Les outils de télédiffusion et de télémaintenance seront alors plus fiables et plus complets; ce qui permettra une exploitation plus facile des applications client-serveur.

CONCLUSION

Ce mémoire a été l'occasion d'une étude très intéressante sur un sujet à la pointe de la technologie en informatique de gestion. Le nombre de séminaires organisés sur le sujet témoigne de l'intérêt que portent aujourd'hui les entreprises à l'architecture client-serveur et plus largement aux architectures coopératives. Cependant il m'a été difficile de déterminer ce qu'est une application client-serveur, une application distribuée ou coopérative. Les informations disponibles dans les services de développement de gestion des entreprises et particulièrement de Merlin Gerin sont limitées aux articles de presse. De façon générale, la presse informatique de gestion pêche par omission, fait de nombreux amalgames qu'il faut ensuite démêler car les articles reprennent les discours marketing des fournisseurs sans véritablement approfondir le sujet.
Contrairement à la presse générale et aux médias qui mettent en valeur ce qui ne marche pas, pour la presse informatique tout marche bien, les outils annoncés sont déjà opérationnels. Il a donc été très difficile dans ce contexte de faire la part des choses et d'arriver à avoir une vue précise de l'état de l'art en matière d'architecture client-serveur dans le domaine de l'informatique de gestion.

Le projet pilote a été un excellent moyen pour concrétiser les notions théoriques du chapitre 2.

Ce travail a été très enrichissant et m'a demandé d'explorer de nombreux domaines nécessaires pour appréhender l'architecture client-serveur dans sa globalité.
Plusieurs composantes matérielles ont été mises en oeuvre : serveur mini HP3000, serveur PC OS2, poste client OS2/Presentation Manager, poste client DOS/Windows, réseau Lan Manager.
Plusieurs composantes logicielles ont nécessité un apprentissage particulier : méthodologie de conception, outils de développement sous Windows, outils de développement sur les serveurs, bases de données relationnelles.

ANNEXES

La mise en place de l'architecture client-serveur a été stoppée pour le moment par Merlin Gerin. Je regrette donc de n'avoir pas pu concrétiser davantage l'expérience du projet pilote.

Dans les mois qui viennent, l'installation de l'AGL Paclan pour plusieurs projets qui vont prochainement passer en phase de développement, sera mon principal objectif.

Nous espérons que l'année 1993, permettra de poursuivre le travail effectué pour ce mémoire, à savoir le développement et la mise en place de plusieurs applications client-serveur. Les outils, tant du point de vue de la conception et du développement, que du point de vue de l'exploitation de telles applications, seront certainement prêts alors.

ANNEXE 1 : critères de choix de l'AGL

LES APPLICATIONS DEVELOPPEES AVEC L'AGL

	FOUNDATION	PACLAN
Données		
.SGBD/R indépendant de l'AGL	-	+
.Permet l'utilisation SQL normalisé	oui	+ Allbase
.Permet l'utilisation de fichiers 'classiques'		Turbo-Image fin 03/92
Architecture d'exécution des applications		
Architecture client/serveur :		
.La machine "client" tourne sous		
DOS	non	X
DOS/Windows3	01/92	01/92
DOS/Windows3/Newwave	'	'
OS2/PM	X	X
OS2	X	Cobol Microfocus
X/MOTIF		
.La machine "serveur" tourne sous		
HP MPEXL	fin 92 ?	X
HP UNIX	fin 92 ?	X
OS2	X	X
Monoposte micro autonome		
.Le micro tourne sous		
DOS	non	X
DOS/Windows3	non	X avec NS-DK
DOS/Windows3/Newwave	'	'

OS2/PM	oui	X
X/MOTIF		
Architecture traditionnelle mini+ terminaux		
.Génération du logiciel sur	solution MG	s'intègre
HP MPEXL	92 ??	à Menu
UNIX natif	Béta 06/92	X
IBM/MVS	X	X
AS400	06/92	X
.Outils de revamping : permet un habillage Windows des applications traditionnelles	non	X
Réseau en mode client/serveur		
.Supporte Lan Manager (Named Pipes)	attaque TCP/IP	X
Mise en oeuvre des maintenances applicatives		
Outils disponibles autour de l'AGL :		
.télémaintenance : solution à la norme OSF		
.télédiffusion		
.gestion du temps : produit qui permet le déclenchement de tâches.		
Coût du Runtime si existant		
.pour la machine "client"	1 KF	Pas de
.pour la machine "serveur"	20KF>10	Runtime
	4KF>100	payant
L'AGL		
Mode de fonctionnement de l'AGL		
Architecture client/serveur :		
.La machine "client" tourne sous		
DOS	X	X
DOS/Windows3	en 92	X
DOS/Windows3/Newwave	'	'
OS2/PM	X	X

.La machine "serveur" tourne sous		
HP MPEXL		
HP UNIX		X
OS2	X	X
Monoposte micro autonome		
.Le micro tourne sous		
DOS		
DOS/Windows3		
DOS/Windows3/Newwave		
OS2/PM	X	X
Architecture traditionnelle mini+ terminaux		
.L'AGL fonctionne sur		
HP MPEXL		
UNIX natif		
IBM/MVS	X	X
Outils CASE et méthodologie cycle-projet		
. Lower/case : spécification du modèle logique par modélisation.	X	X
.Utilise la norme IRDS : norme qui définit les échanges entre AGL au niveau des encyclopédies	ESF Norme IBM	X
.Autre norme suivie		
.Assistance au passage au modèle logique MCx ----> MLx automatique ?		Assistance oui Automatisme version suivante
. MLx ---> MPx	non	oui
Référentiel		
.Connexion à ADW au niveau du dictionnaire	X	X
.Quels sont les modèles récupérables depuis ADW	MCD	MCD
.Sont-ils utilisables ou destinés à la documentation seulement	peu	peu
.Navigation possible entre le référentiel et ADW,	oui avec pertes	oui avec pertes
dans les deux sens		X

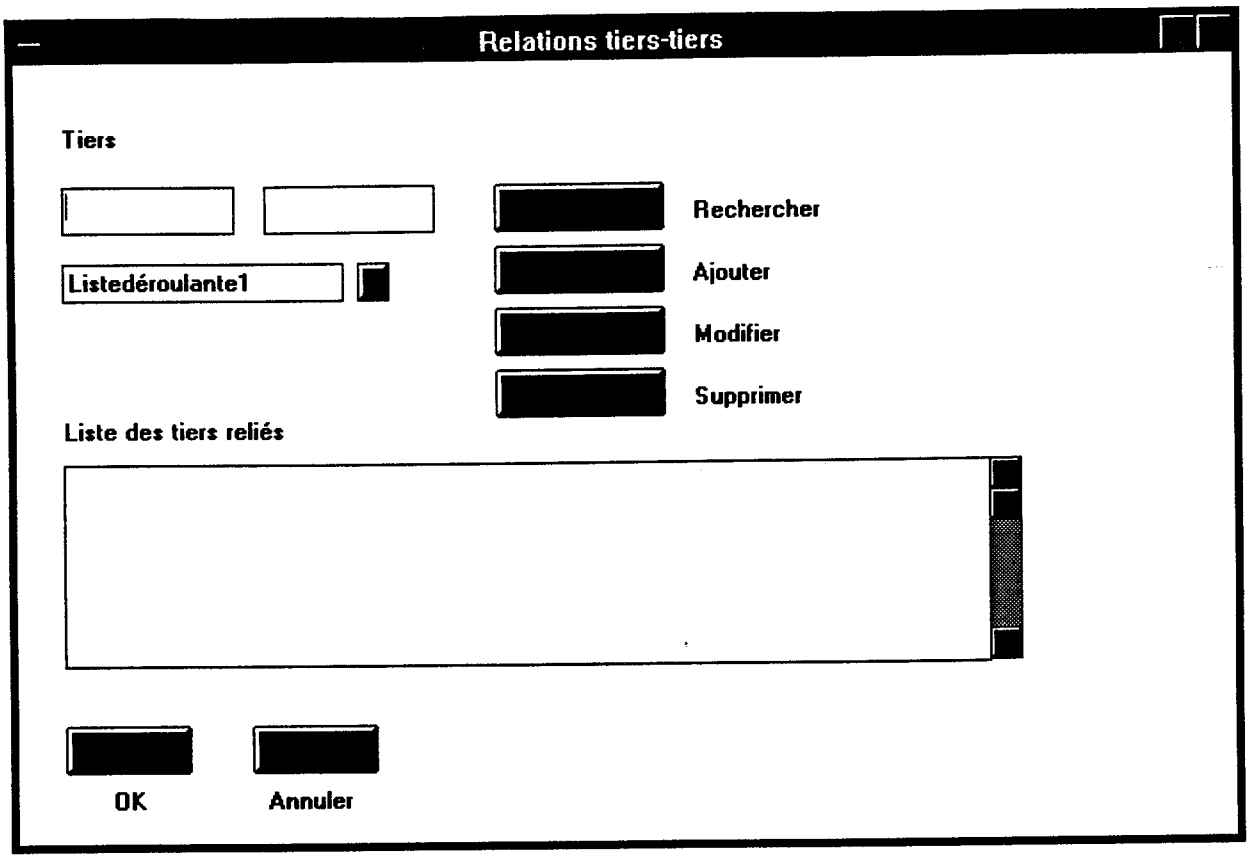
.Le référentiel est-il		
. centralisé		X
. réparti		X
.On-line pendant le cycle de développement /maintenance	X	pas pour toutes les fonctions
.Possibilité de croissance en volume		X
.Gestion interne du référentiel :		
. processus de SIGN-OUT, SIGN-IN	non	X
. contrôle des changements compte-tenu du nombre		
d'utilisateurs.		X
. accès concurrents		X
.Multi-langues	X +	X
.Méta modèle ouvert	X +	X -
.L'AGL intègre-t-il une interface de type API (Application Program Interface)	X	X
Qualité		
.Outils pour contrôler la qualité des modèles,	X	X
des codes générés		
Génération de code		
.Génère du L3G (Cobol ou C)	X	X
.Codage manuel se fait en L3G ou langage spécifique	C Cobol	Paclan
.Permet la réutilisabilité du code	X	X
.Aide à la réutilisation (en donnant des critères de sélection retrouver des modules dans le référentiel)	X	requêtes SQL +
.Modèles/squelettes standard L3G fournis au départ	X	X
.Gestion des versions	non	X
.Gestion de configurations	non	X
.Prototypage		
. possible	X	X
. récupérable	X	X
. dynamique	X	X

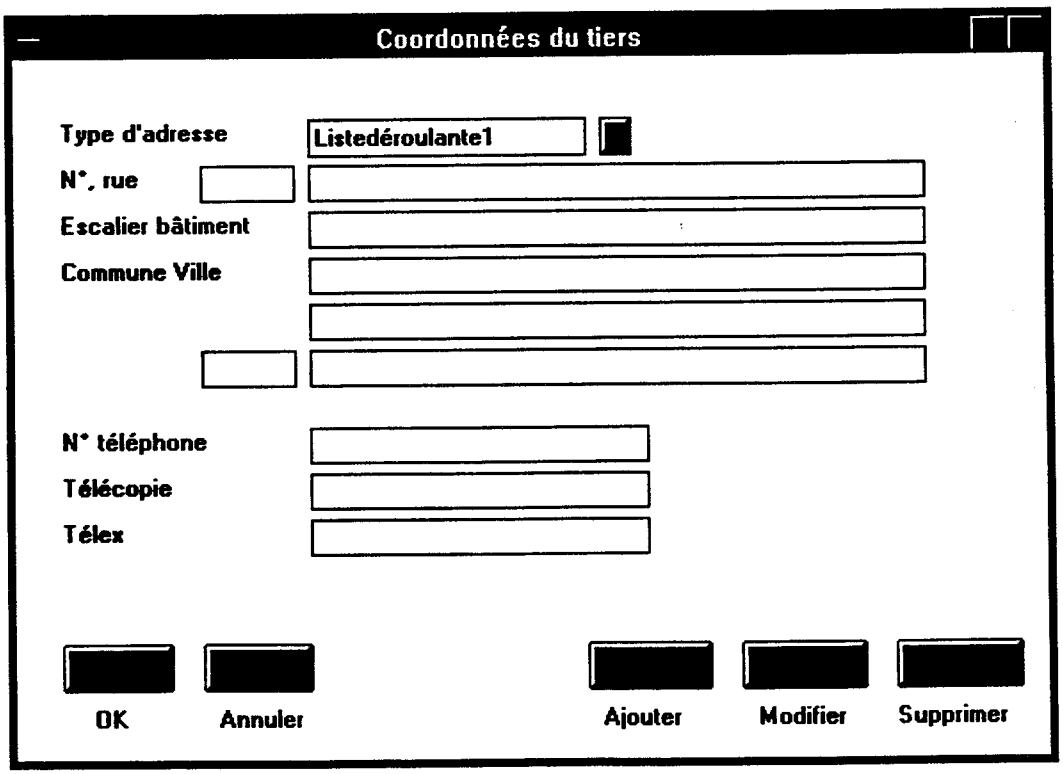
Tests		
.Outils de tests	oui Foundat. non FCP	non
.Outils de génération de jeux de tests	oui Foundat non FCP	non
Méthodologie		
.Méthodologie supportant l'AGL		
. RAD		
. MERISE	X	X
. Autre	Yourdon	Yourdon
.Méthodologie spécifique à la programmation événementielle, à l'approche orientée objet	Béta	non
.Méthodologie spécifique à l'architecture client/serveur	Béta 18 mois d'expérience	
.Reverse engineering :	non	Pacreverse
. outils	en cours	oui
. méthodologie		
.Permet la génération d'applications multi-langues	X	X
Formation pour le projet pilote		
.Existence d'une formation	X	X
.Combien de jours (démarrage)	5j	5j
.Méthode d'apprentissage	monitorat	monitorat
.Accompagnement par le fournisseur	400 KF les softs sont à MG	350 KF les softs sont prêtés
.Existence d'un support technique	X	X
Combien de personnes en France sur le produit	50	80
.Temps pour devenir autonome (délai)	2 mois	2 mois

Pérennité du produit/société		
.Adhésion aux normes ISO / OSF	X	X
.Nombre de clients consultables en France	Found. 60	Pacbase 450 Paclan 7
.Etat du produit : annoncé, en béta test, prêt	prêt C/S OS2	prêt MPE XL
.Le fournisseur a-t-il une stratégie		
. HP/MPEXL	non	oui
. compétences SSII	peu	beaucoup
Mise en oeuvre de l'AGL		
.Charge initiale d'implémentation pour MG	6 mois	6 mois
.Faut-il prévoir un administrateur de l'AGL en plus de l'administrateur des données	2 pers	2 pers
.Charge en temps par développeur	2 mois/ h	2 mois/h
.Outils d'exploitation de l'AGL		X
Coûts		
.Coût de la station de développement (logiciels)	70 KF	100 KF
.Coût de la maintenance	15 % du prix	catalogue
.Fréquence de la maintenance	1 an	1 an
Gains		
.Productivité escomptée par rapport à un développement "traditionnel" en temps exprimée en ratio		Gain de 30 à 40 % pour Pacbase
.Réduction des coûts de développement par rapport à un développement "traditionnel" exprimée en ratio		
.Réduction du temps de maintenance par rapport à un développement "traditionnel" exprimée en ratio.		Gain de 300 à 400%

ANNEXE 2 : exemples de fenêtres du projet pilote

Exemples de fenêtres développées pour le projet pilote : coordonnées du tiers et relations tiers-tiers.





ANNEXE 3 : exemples de librairies développées avec NS-DK

Coordonnées du tiers

```
;Constantes de la fenêtre

CONST kCOMajReq$ "COMISEAJOU"
CONST kErrCO001% 1001
CONST kErrCO001$  "Le code département est obligatoire"
CONST kErrCO002% 1002
CONST kErrCO002$  "Le nom de la commune est obligatoire"

; Variables de la fenêtre

GLOBAL F_CONTEXTE CO_CTX
GLOBAL SGES06 ES06
GLOBAL SGB_ES06 B_ES06

;=========================================
;    Couche Interface
;=========================================

;_____________________________________________________
INSTRUCTION CO_INIT    ; Bouton coordonnées dans STITI_00
;_____________________________________________________

LOCAL wDial%

SETPTR GETSPTR%(SPTR_WAIT%);  Affichage du sablier

OPENH STICO_00,0,pWinTab%[kFeCO%]
MOVE pWinTab%[kFeCO%] TO wDial%

; Chargement du n° de tiers actif
MOVE NUTITI% TO STICO_00(wDial%).NUTITI_0

MOVE kNoAction TO CO_CTX.Action
MOVE NUTITI% TO ES06.NUTITI
MOVE 1 TO ES06.CDTCTC
MOVE 1 TO STICO_00(wDial%).CDTCTC_0
MOVE CO_B_MISEAJOUR% TO wErr%
IF wErr% = kNoErr%
   CO_P_AfficheResul
ELSE CO_P_AfficheBlanc
ENDIF

SETDATEFORMAT DD_MM_YY%

SHOW wDial%
SETFOCUS wDial%
SETPTR GETSPTR%(SPTR_ARROW%)    ; Affiche la flèche

ENDINSTRUCTION   ;   Fin de CO_INIT
```

```
;_____________________________________________
INSTRUCTION CO_I_Ajouter   ; Bouton Ajouter
;_____________________________________________
LOCAL wDial%
LOCAl wErr%

SETPTR GETSPTR%(SPTR_WAIT%);  Affichage du sablier

ES06_INIT

MOVE pWinTab%[kFeCo%] TO wDial%

;Màj de la date de modification
SETDATESEPARATOR "/"

 MOVE STRING$(CURRENTDATE%,"dd/mm/yy") TO STICO_00(wDial%).DTXTDM_0

 MOVE CO_P_00SAVE% TO wErr%   ; Sauvegarde de la fenêtre STICO_00

 ;Création  dans la base

 MOVE kCreation TO CO_CTX.Action
 IF wErr% = kNoErr%
    MOVE CO_B_MISEAJOUR% TO wErr%
 ENDIF

 SETPTR GETSPTR%(SPTR_ARROW%)     ; Affiche la flèche

 ENDINSTRUCTION


 ;_____________________________________________
 INSTRUCTION CO_I_Modifier   ; Bouton Modifier
 ;_____________________________________________

 LOCAL wDIAL%
 LOCAL wErr%

 SETPTR GETSPTR%(SPTR_WAIT%)

 MOVE pWinTab%[kFeCO%] TO wDial%

 MOVE STRING$(CURRENTDATE%,"dd/mm/yy") TO STICO_00(wDial%).DTXTDM_0

 ; Sauvegarde de la fenêtre dans ES06
 MOVE CO_P_00SAVE% TO wErr%

 ; Màj de la base

 MOVE kModification TO CO_CTX.Action
 IF wErr% = kNoErr%
    MOVE CO_B_MISEAJOUR% TO wErr%
 ENDIF

 SETPTR GETSPTR%(SPTR_ARROW%)

 ENDINSTRUCTION
```

```
;_____________________________________
INSTRUCTION CO_I_Supprimer     ; Bouton Supprimer
;_____________________________________

LOCAl wErr%
LOCAL wDIAL%

SETPTR GETSPTR%(SPTR_WAIT%)

; Sauvegarde de la fenêtre dans ES06
MOVE CO_P_00SAVE% TO wErr%

; Màj de la base

MOVE kSuppression TO CO_CTX.Action

IF wErr% = kNoErr%
   MOVE CO_B_MISEAJOUR% TO wErr%
ENDIF

IF wErr% = kNoErr%

   CO_P_AfficheBlanc
ENDIF

SETPTR GETSPTR%(SPTR_ARROW%)

ENDINSTRUCTION

;_____________________________________
INSTRUCTION CO_I_TCClick   ;Modification du type de coordonnée par click
;                           Dans la combo-box
;_____________________________________

LOCAL wDial%
LOCAL LineSel%
LOCAL wErr%

SETPTR GETSPTR%(SPTR_WAIT%) ; Affichage du sablier

MOVE pWinTab%[kFeCO%] TO wDial%

MOVE kNoAction TO CO_CTX.Action   ; Recherche dans la base si existe déjà
MOVE STICO_00(wDial%).NUTITI_0 TO ES06.NUTITI
MOVE STICO_00(wDial%).CDTCTC_0 TO ES06.CDTCTC
MOVE CO_B_MISEAJOUR% TO wErr%
IF wErr% = kNoErr%
   CO_P_AfficheResul

ELSE CO_P_AfficheBlanc

ENDIF

SETPTR GETSPTR%(SPTR_ARROW%)

ENDINSTRUCTION
```

```
;=========================================
;     Couche PRESENTATION
;=========================================


;_____________________________________
FUNCTION CO_P_00SAVE%  ; Sauvegarde de la fenêtre STICO_00 dans ES06
;_____________________________________

LOCAL wErr%
LOCAL wDial%
LOCAL Temp$
LOCAL Temp%


MOVE CO_C_CONTROLE% TO wErr%

MOVE pWinTab%[kFeCO%] TO wDial%

IF wErr% = kNoErr%
   MOVE STICO_00(wDial%).NUTITI_0  TO ES06.NUTITI


   MOVE STICO_00(wDial%).CDTCTC_0  TO Temp$
   MOVE INT(Temp$) TO Temp%
   MOVE Temp% TO ES06.CDTCTC

   MOVE STICO_00(wDial%).DTXTDM_0  TO Temp$
   MOVE DATE%(Temp$) TO Temp%
   MOVE STRING$(Temp%,"ddmmyy") TO Temp$
    MOVE Temp$ TO ES06.DTXTDM

    MOVE STICO_00(wDial%).ADCOAB_0  TO ES06.ADCOAB
    MOVE STICO_00(wDial%).ADCOAA_0  TO ES06.ADCOAA
    MOVE STICO_00(wDial%).CDCOAR_0  TO ES06.CDCOAR
    MOVE STICO_00(wDial%).ADCOBV_0  TO ES06.ADCOBV
    MOVE STICO_00(wDial%).NUCOBP_0  TO ES06.NUCOBP
    MOVE STICO_00(wDial%).ADCOCP_0  TO ES06.ADCOCP
    MOVE STICO_00(wDial%).LBCODP_0  TO ES06.LBCODP
    MOVE STICO_00(wDial%).NUCOAD_0  TO ES06.NUCOAD
    MOVE STICO_00(wDial%).NUCOTE_0  TO ES06.NUCOTE
    MOVE STICO_00(wDial%).NUCOTB_0  TO ES06.NUCOTB
    MOVE STICO_00(wDial%).NUCOTD_0  TO ES06.NUCOTD
    MOVE STICO_00(wDial%).NUCOTX_0  TO ES06.NUCOTX
    MOVE STICO_00(wDial%).CDCOCP_0  TO ES06.CDCOCP

ENDIF

RETURN wErr%
ENDFUNCTION
```

```
;_____________________________________
INSTRUCTION CO_P_AfficheResul
;_____________________________________
LOCAL wDial%
LOCAL Temp$
LOCAL Temp%


MOVE pWinTab%[kFeCO%] TO wDial%

   ; Traitement de la date

   MOVE ES06.DTXTDM   TO Temp$
   SETDATESEPARATOR ""
   MOVE DATE%(Temp$)  TO Temp%
   SETDATESEPARATOR "/"
   MOVE STRING$(Temp%,"dd/mm/yy")   TO STICO_00(wDial%).DTXTDM_0

   MOVE ES06.ADCOAB    TO STICO_00(wDial%).ADCOAB_0
   MOVE ES06.ADCOAA    TO STICO_00(wDial%).ADCOAA_0
   MOVE ES06.CDCOAR    TO STICO_00(wDial%).CDCOAR_0
   MOVE ES06.ADCOBV    TO STICO_00(wDial%).ADCOBV_0
   MOVE ES06.NUCOBP    TO STICO_00(wDial%).NUCOBP_0
   MOVE ES06.ADCOCP    TO STICO_00(wDial%).ADCOCP_0
   MOVE ES06.LBCODP    TO STICO_00(wDial%).LBCODP_0
   MOVE ES06.NUCOAD    TO STICO_00(wDial%).NUCOAD_0
   MOVE ES06.NUCOTE    TO STICO_00(wDial%).NUCOTE_0
   MOVE ES06.NUCOTB    TO STICO_00(wDial%).NUCOTB_0
   MOVE ES06.NUCOTD    TO STICO_00(wDial%).NUCOTD_0
   MOVE ES06.NUCOTX    TO STICO_00(wDial%).NUCOTX_0
   MOVE ES06.CDCOCP    TO STICO_00(wDial%).CDCOCP_0


ENDINSTRUCTION
;_____________________________________
INSTRUCTION CO_P_AfficheBlanc; remet à blanc les zones de saisie
;_____________________________________
LOCAL wDial%


MOVE pWinTab%[kFeCO%] TO wDial%

   MOVE ""    TO STICO_00(wDial%).DTXTDM_0
   MOVE ""    TO STICO_00(wDial%).ADCOAB_0
   MOVE ""    TO STICO_00(wDial%).ADCOAA_0
   MOVE ""    TO STICO_00(wDial%).CDCOAR_0
   MOVE ""    TO STICO_00(wDial%).ADCOBV_0
   MOVE ""    TO STICO_00(wDial%).NUCOBP_0
   MOVE ""    TO STICO_00(wDial%).ADCOCP_0
   MOVE ""    TO STICO_00(wDial%).LBCODP_0
   MOVE 0     TO STICO_00(wDial%).NUCOAD_0
   MOVE ""    TO STICO_00(wDial%).NUCOTE_0
   MOVE ""    TO STICO_00(wDial%).NUCOTB_0
   MOVE ""    TO STICO_00(wDial%).NUCOTD_0
   MOVE ""    TO STICO_00(wDial%).NUCOTX_0
   MOVE ""    TO STICO_00(wDial%).CDCOCP_0

ENDINSTRUCTION
```

```
;=======================================
;  Couche Contrôle
;=======================================

;_______________________________________
FUNCTION CO_C_CONTROLE%          ; Contrôles de la fenêtre CO
;_______________________________________
LOCAL wErr%
LOCAL wStr$
LOCAL wDial%

MOVE kNoErr% TO wErr%
MOVE pWinTab%[kFeCO%] TO wDial%

MOVE "" TO wStr$
MOVE FILLER$(ASC%(" "),LENGTH STICO_00(wDial%).CDCOCP_0) TO wStr$
IF STICO_00(wDial%).CDCOCP_0 = wStr$ AND wErr% = kNoErr%
   MOVE kErrCO001% TO wErr%
   SETFOCUS  STICO_00(wDial%)          ; on fait passer STITI_00 au premier plan
   MESSAGE kErrgen$, kErrCO001$
ENDIF

MOVE "" TO wStr$
MOVE FILLER$(ASC%(" "),LENGTH STICO_00(wDial%).ADCOCP_0) TO wStr$
IF STICO_00(wDial%).ADCOCP_0 = wStr$ AND wErr% = kNoErr%
   MOVE kErrCO002% TO wErr%
   SETFOCUS  STICO_00(wDial%)          ; on fait passer STITI_00 au premier plan
   MESSAGE kErrgen$, kErrCO002$
ENDIF

RETURN wErr%

ENDFUNCTION
```

```
;=====================================
;   Couche BASE DE DONNEES
;=====================================

;_____________________________________
FUNCTION CO_B_MISEAJOUR%
;_____________________________________

LOCAL wNBCarIn%(2), wNbCarOut%(2)
LOCAL TBuffer wBuffer
LOCAL wErr%

; Envoi de la requête

MOVE kNoErr% TO wErr%
MOVE "0" TO B_ES06.BOARIE ; Initialisation indicateur d'erreur
MOVE CO_CTX.Action TO B_ES06.CDPBAO   ; Alimentation du code action
ES06TOB_ES06  ; Alimentation des rubriques fonctionnelles

MOV @B_ES06,@wBuffer,SIZEOF B_ES06
MOVE SIZEOF SGB_ES06 TO wNbCarIn%


MOVE REQUETE%(kCOMajReq$,wNbCarIn%,wNbCarOut%,@wBuffer) TO wErr%

; Détection des messages applicatifs
IF wErr% = kNoErr%
     MOVE U_AfficheErreursAppli (@wBuffer) TO wErr% ; Affichage d'erreurs applicatives
ENDIF

IF wErr% = kNoErr%
       MOV @wBuffer,@B_ES06,wNbCarOut%
       B_ES06TOES06
ENDIF

RETURN wErr%

ENDFUNCTION
```

Librairie Relation Tiers-Tiers

```
; Constántes de la fenêtre

CONST kCTMajReq$ "CTMISEAJOU"
CONST kCTRechReq$ "CTRECHERCH"
;CONST kCTResuRech$ "CTRESULT.TXT"
CONST kCTResuRech$ "D:\"

; Constantes d'erreur

; Variables de la fenêtre

GLOBAL F_CONTEXTE CT_CTX
GLOBAL SGES07 ES07
GLOBAL SGB_ES07 B_ES07
GLOBAL SGRS07 RS07
GLOBAL SGB_RS07 B_RS07
GLOBAL SGLS07 LS07
;GLOBAL  SGB_ES07 B_LS07

GLOBAL  CT_DIAL%
GLOBAL CONTROL Ctrl
GLOBAl aLine$
GLOBAL CTResuRech$(65)
```

STICTLIB 27/02/92

```
;================================================
;    Couche INTERFACE
;================================================


; ______________________________________________
INSTRUCTION CT_INIT              ; Ouverture et chargement de la fenêtre Relations tiers-tiers
;______________________________________________

LOCAL wErr%

SETPTR GETSPTR%(SPTR_WAIT%)

; Défini le format de date
SETDATEFORMAT DD_MM_YY%

;ouverture de la fenêtre qui reste cachée
OPENH STICT_00,0,CT_DIAL%

; Chargement du dernier n° de tiers actif
MOVE NUTITI%  TO STICT_00(CT_DIAL%).NUTITI_0

;Initialisation adresse du contrôle Liste
MOVE STICT_00(CT_DIAL%).LICTLI_0 TO @Ctrl


IF NUTITI% <> 0
     ;Chargement de la liste des relations tiers-tiers
    MOVE NUTITI% TO RS07.NUTITI
    MOVE CT_B_RECHERCHE% TO wErr%
    IF wErr% = kNoErr%
       LS_LOAD_CTRL Ctrl, kCTResuRech$ & CTResuRech$
    ENDIF
ENDIF

SETPTR GETSPTR%(SPTR_ARROW%)
SETFOCUS STICT_00(CT_DIAL%).NUTITI_0
SHOW CT_DIAL%                          ; visualise STICT_00
```

```
ENDINSTRUCTION
;
;_____________________________________________
INSTRUCTION  CT_I_Ajouter                ;Bouton Ajouter
;_____________________________________________

LOCAL wErr%

SETPTR GETSPTR%(SPTR_WAIT%)
ES07_INIT

SETDATESEPARATOR "/"
MOVE STRING$(CURRENTDATE%,"dd/mm/yy") TO STICT_00(CT_DIAL%).DTXTDC_0
MOVE STRING$(CURRENTDATE%,"dd/mm/yy") TO STICT_00(CT_DIAL%).DTXTDM_0

MOVE CT_P_00SAVE% TO wErr% ;  Sauvegarde de la fenêtre STICT_00 dans ES07

IF STICT_00(CT_DIAL%).NUTITI_0 <> NUTITI%
     MOVE CT_B_RECHERCHE% TO wErr%
     IF wErr% = kNoErr%
        LS_LOAD_CTRL Ctrl, kCTResuRech$ & CTResuRech$
        MOVE STICT_00(CT_DIAL%).NUTITI_0 TO NUTITI%
     ENDIF
ENDIF

IF wErr% = kNoErr%
   MOVE kCreation TO CT_CTX.Action
   MOVE CT_B_MISEAJOUR% TO wErr%
ENDIF

IF wErr% = kNoErr%
   FenTOListe
   LS_AjoutDsLI Ctrl, aLine$
ENDIF

SETPTR GETSPTR%(SPTR_ARROW%)
ENDINSTRUCTION

;_____________________________________________
INSTRUCTION  CT_I_Modifier               ;Bouton Modifier
;_____________________________________________
LOCAL wErr%
LOCAL I%

NOUPDATE Ctrl
SETPTR GETSPTR%(SPTR_WAIT%)
; Récupère le numéro de ligne sélectionnée
MOVE SELECTION%(Ctrl) TO I%

IF I% <> -1        ; si une ligne est selectionnée
      MOVE STRING$(CURRENTDATE%,"dd/mm/yy") TO STICT_00(CT_DIAL%).DTXTDM_0
      MOVE CT_P_00SAVE% TO wErr%
      IF wErr% = kNoErr%
         MOVE kModification TO CT_CTX.Action
         MOVE CT_B_MISEAJOUR% TO wErr%
      ENDIF
      IF wErr% = kNoErr%
         FenTOListe
         LS_ModifDsLi Ctrl, aLine$, I%
      ENDIF
ELSE
   MESSAGE "ATTENTION","Aucune ligne sélectionnée"
ENDIF   ; Fin Si une ligne selectionnée
```

```
  UPDATE Ctrl
  SETPTR GETSPTR%(SPTR_ARROW%)

  ;
  ENDINSTRUCTION



  ;_____________________________________________
  INSTRUCTION CT_I_AfficheSelLigne     ; Affiche dans les zones de saisies
  ;                                      la ligne selectionnée
  ;_____________________________________________
  LOCAL NuLi%


  ; Récupère le numéro de ligne sélectionnée
  MOVE SELECTION%(Ctrl) TO NuLi%

  ; Copie la ligne selectionnée de la liste dans les zones de saisies
  ListeTOFen NuLI%

  ENDINSTRUCTION
  ;_____________________________________________
  INSTRUCTION  CT_I_Supprimer                ;Bouton Supprimer
  ;_____________________________________________
  LOCAL wErr%
  LOCAL I%

  NOUPDATE Ctrl

  SETPTR GETSPTR%(SPTR_WAIT%)

  ; Récupère le numéro de ligne sélectionnée
  MOVE SELECTION%(Ctrl) TO I%

  IF I% <> -1        ; si une ligne est selectionnée
     MOVE CT_P_OOSAVE% TO wErr%
     IF wErr% = kNoErr%
        MOVE kSuppression TO CT_CTX.Action
        MOVE CT_B_MISEAJOUR% TO wErr%
     ENDIF
     IF wErr% = kNoErr%
        DELETE AT I% FROM Ctrl
     ENDIF
  ELSE
     MESSAGE "ATTENTION","Aucune ligne sélectionnée"
  ENDIF

  UPDATE Ctrl
  SETPTR GETSPTR%(SPTR_ARROW%)

  ENDINSTRUCTION
  ;_____________________________________________
  INSTRUCTION  CT_I_Rechercher               ;Bouton Rechercher
  ;_____________________________________________
  LOCAL wErr%

  SETPTR GETSPTR%(SPTR_WAIT%)

  MOVE STICT_OO(CT_DIAL%).NUTITI_O  TO RSO7.NUTITI
  MOVE CT_B_RECHERCHE% TO wErr%
  IF wErr% = kNoErr%
     LS_LOAD_CTRL Ctrl, kCTResuRech$ & CTResuRech$
  ENDIF
```

```
MOVE STICT_00(CT_DIAL%).NUTITI_0 TO NUTITI%

SETPTR GETSPTR%(SPTR_ARROW%)

ENDINSTRUCTION

;_____________________________________________
INSTRUCTION  CT_I_CLOSE                ;Bouton Close
;_____________________________________________

IF CT_DIAL% <> kNoHandle%
   HIDE CT_DIAL%
   CLOSE CT_DIAL%
ENDIF

ENDINSTRUCTION


;=============================================
;  Couche PRESENTATION
;=============================================

;_____________________________________________
FUNCTION CT_P_00SAVE%     ; Sauvegarde de la fenêtre STICT_00 dans ES07
;_____________________________________________

LOCAL wErr%
LOCAL Temp$
LOCAL Temp%

MOVE CT_C_CONTROLE% TO wErr%

IF wErr% = kNoErr%
     MOVE STICT_00(CT_DIAL%).NUTITI_0  TO  ES07.NUTITI  ; chargement ES07 à partir de STICT_00
     MOVE STICT_00(CT_DIAL%).NUTIT1_0  TO  ES07.NUTIT1
     MOVE STICT_00(CT_DIAL%).CDCTRE_0  TO ES07.CDCTRE

     MOVE STICT_00(CT_DIAL%).DTXTDC_0  TO Temp$
     MOVE DATE%(Temp$) TO Temp%
     MOVE STRING$(Temp%,"ddmmyy") TO Temp$
     MOVE Temp$ TO ES07.DTXTDC

     MOVE STICT_00(CT_DIAL%).DTXTDM_0  TO Temp$
     MOVE DATE%(Temp$) TO Temp%
     MOVE STRING$(Temp%,"ddmmyy") TO Temp$
     MOVE Temp$ TO ES07.DTXTDM

 ENDIF

 RETURN wErr%
 ENDFUNCTION                ; fin de la fonction CT_P_00SAVE

 ;_____________________________________________
 INSTRUCTION CT_P_AfficheBlanc
 ;_____________________________________________

 MOVE 0 TO STICT_00(CT_Dial%).NUTIT1_0
 MOVE "" TO STICT_00(CT_Dial%).CDCTRE_0
 MOVE "" TO STICT_00(CT_Dial%).DTXTDM_0
 MOVE "" TO STICT_00(CT_Dial%).DTXTDC_0
 ENDINSTRUCTION

 ;_____________________________________________
```

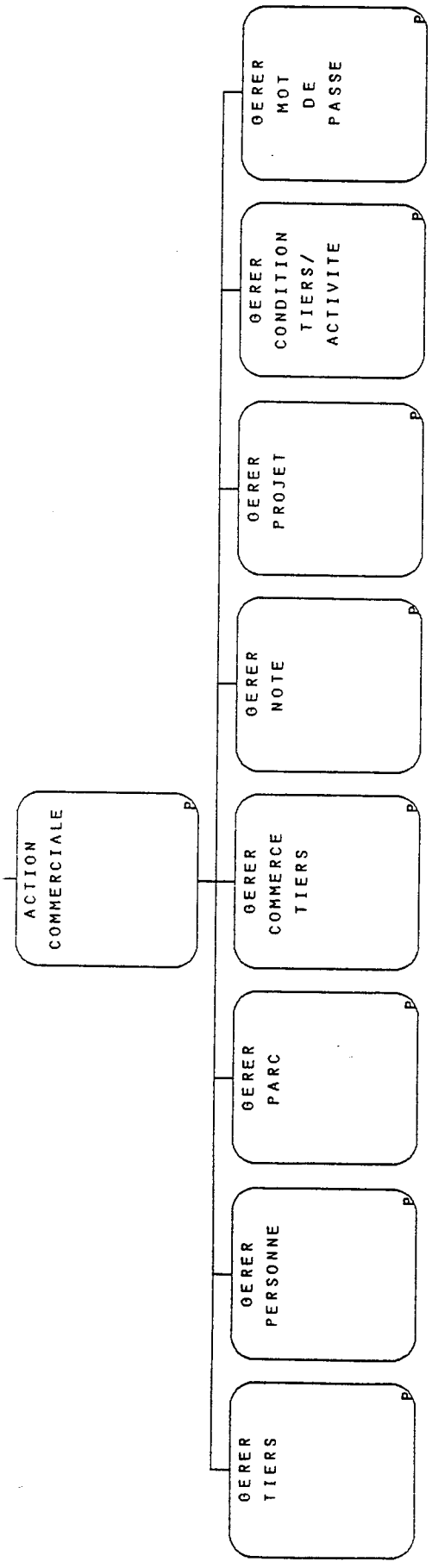
```
INSTRUCTION FenTOListe   ; Concatène les zones de saisie dans une ligne
;_____________________________________________

LOCAL A$,B$,C$,D$,E$,F$
LOCAL Temp%

;MOVE STICT_00(CT_DIAL%).NUTITI_0 TO A$
MOVE U_ARRAYTOSTRING (@B_ES07.NUTITI,006) TO A$
;MOVE STICT_00(CT_DIAL%).NUTIT1_0 TO B$
MOVE U_ARRAYTOSTRING (@B_ES07.NUTIT1,006) TO B$
MOVE STICT_00(CT_DIAL%).CDCTRE_0 TO C$
MOVE ES07.RSTIRS                 TO D$

MOVE STICT_00(CT_DIAL%).DTXTDC_0 TO E$
MOVE DATE%(E$) TO Temp%
MOVE STRING$(Temp%,"ddmmyy")     TO E$
MOVE STICT_00(CT_DIAL%).DTXTDM_0 TO F$
MOVE DATE%(F$) TO Temp%
MOVE STRING$(Temp%,"ddmmyy")     TO F$


; Ajoute des séparateurs définis dans le contrôle Liste de STICT_00
; pour définir les tabulations

MOVE A$ & '?' & B$ & '?' & C$ & '?' & D$ & '?' & E$ & '?' & F$ TO aLine$

ENDINSTRUCTION
;_____________________________________________

;_____________________________________________
INSTRUCTION ListeTOFen  NuLi% ; Copie la ligne selectionnée de la liste dans les zones de saisies
;_____________________________________________

LOCAL A$[7]
LOCAL K%,P%
;Initialisations
MOVE 1 TO K%

; Récupération de la ligne selectionnée
MOVE Ctrl[NuLi%] TO aLine$

;Extraction des différentes zones

MOVE POS%('?' , aLine$) TO P%

WHILE P% <> 0

      MOVE aLine$(0..P%-2) TO A$[K%]
      MOVE DELETE$(aLine$,1,P%) TO aLine$
      MOVE K% + 1 TO K%
      MOVE POS%('?' , aLine$) TO P%

ENDWHILE

MOVE aLine$ TO A$[K%]

MOVE A$[1] TO STICT_00(CT_DIAL%).NUTITI_0
MOVE A$[2] TO STICT_00(CT_DIAL%).NUTIT1_0
MOVE A$[3] TO STICT_00(CT_DIAL%).CDCTRE_0

MOVE STRING$(INT(A$[5]),"00/00/00") TO STICT_00(CT_DIAL%).DTXTDC_0
;MOVE A$[5] TO STICT_00(CT_DIAL%).DTXTDC_0
MOVE STRING$(INT(A$[6]),"00/00/00") TO STICT_00(CT_DIAL%).DTXTDM_0
```

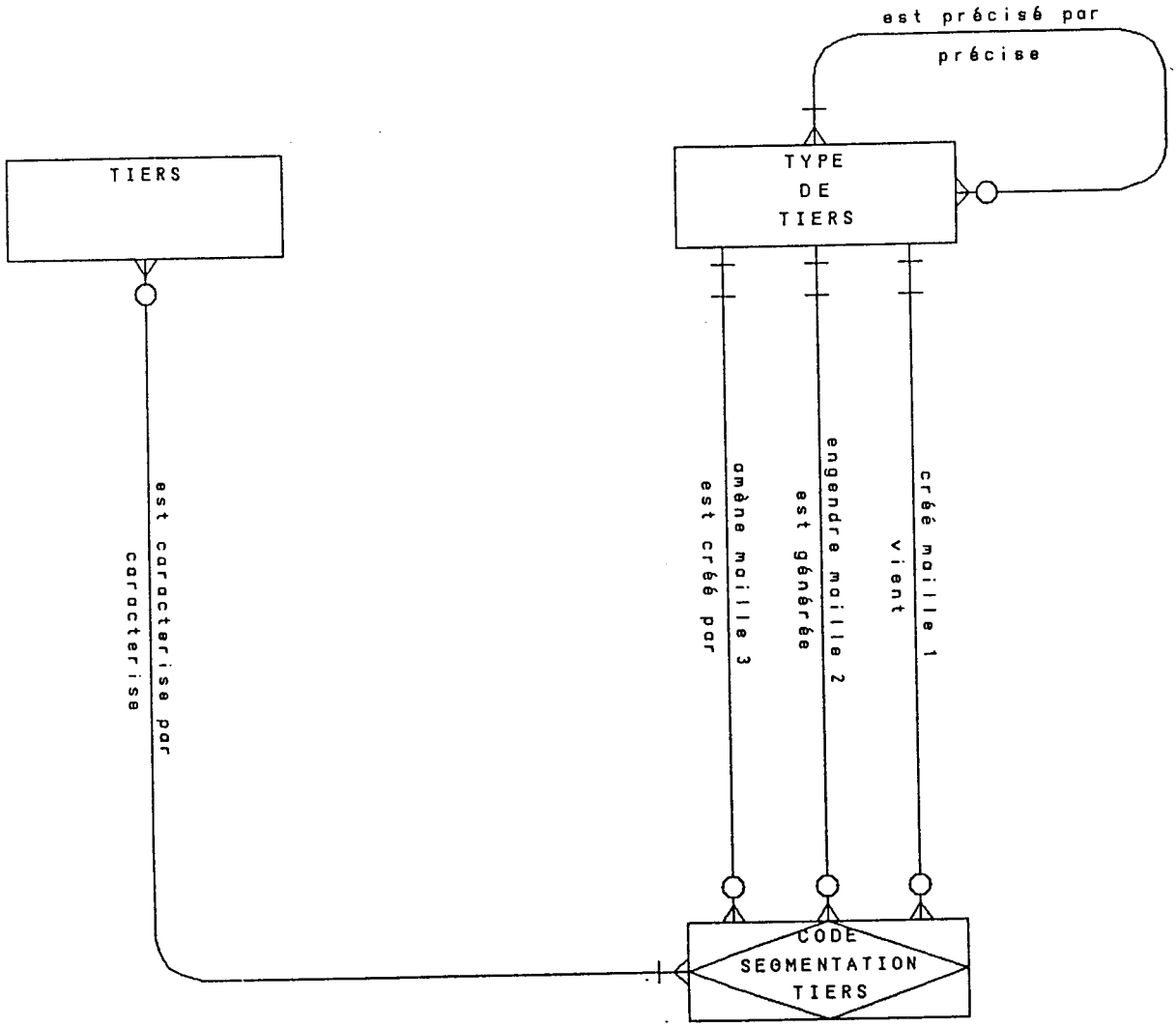
```
ENDINSTRUCTION
;_____________________________________________


;=============================================
;  Couche CONTROLE
;=============================================
;_____________________________________________
FUNCTION CT_C_CONTROLE%          ; Contrôles de la fenêtre CT
;_____________________________________________

LOCAL wErr%
MOVE kNoErr% TO wErr%
IF STICT_00(CT_DIAL%).NUTITI_0 = 0
   MOVE 1 TO wErr%
   MESSAGE "ATTENTION","Numéro de tiers manquant"
   SETFOCUS STICT_00(CT_DIAL%).NUTITI_0
ENDIF

IF STICT_00(CT_DIAL%).NUTIT1_0 = 0 AND wErr% = kNoErr%
   MOVE 1 TO wErr%
   MESSAGE "ATTENTION","Numéro de tiers manquant"
   SETFOCUS STICT_00(CT_DIAL%).NUTIT1_0
ENDIF
IF wErr% = kNoErr%
   IF STICT_00(CT_DIAL%).CDCTRE_0 = ""  OR     \
      STICT_00(CT_DIAL%).CDCTRE_0 = " "
      MOVE 1 TO wErr%
      MESSAGE "ERREUR","code relation manquant"
      SETFOCUS STICT_00(CT_DIAL%).CDCTRE_0
   ENDIF
ENDIF

RETURN wErr%
ENDFUNCTION


;=============================================
;  Couche BASE DE DONNEES
;=============================================
;_____________________________________________
FUNCTION CT_B_MISEAJOUR%
;_____________________________________________

LOCAL wNbCarIn%(2) , wNbCarOut%(2)
LOCAL wErr%,wErr2%
LOCAL TBuffer wBuffer

; Envoi de la Requête
;- - - - - -- - - - - - - - - --
MOVE kNoErr% TO wErr%
MOVE "0" TO B_ES07.BOARIE;     Initialisation indicateur d'erreur
MOVE CT_CTX.Action TO B_ES07.CDPBAO  ; Alimentation du code action
ES07TOB_ES07   ; Alimentation des rubriques fonctionnelles
MOVE SIZEOF SGB_ES07 To wNbCarIn%
MOV @B_ES07,@wBuffer,SIZEOF SGB_ES07
MOVE REQUETE% (kCTMajReq$,wNbCarIn%,wNbCarOut%,@wBuffer) To wErr%

;Détection des messages applicatifs
IF wErr% = kNoErr%
       MOVE U_AfficheErreursAppli (@wBuffer) TO wErr%;  Affichage d'erreurs applicatives
```

```
ENDIF  ;    Fin du wErr% <> kNoErr%

;alimentation de la variable objet
IF wErr% = kNoErr%
        MOV @wBuffer,@B_ES07,wNbCarOut%
        B_ES07TOES07
ENDIF

RETURN wErr%
ENDFUNCTION;   Fin de la fonction TI_B_MISEAJOUR%

;_____________________________________________
FUNCTION CT_B_RECHERCHE%
;_____________________________________________

LOCAL wErr%
LOCAL wNbCarIn%(2), wNbCarOut%(2)
LOCAL TBuffer wBuffer
LOCAL NUTITIW%

MOVE RS07.NUTITI TO NUTITIW%
MOVE "0" TO B_RS07.BOARIE
MOVE kRecherche TO B_RS07.CDPBAO
RS07TOB_RS07
MOVE SizeOf SGB_RS07 TO wNbCarIn%
MOV @B_RS07,@wBuffer,SIZEOF SGB_RS07
Move Requete%(kCTRechReq$,wNbCarIn%,wNbCarOut%,@wBuffer) TO wErr%

IF wErr% = kNoErr%
    IF wBuffer.Buffer [0] <> 0
        MESSAGE "Relation Tiers/Tiers","Pas de relation pour le tiers" && NUTITIW%
    ENDIF
ENDIF

MOVE U_ArrayToString(@wBuffer +10,50) TO CTResuRech$

RETURN wErr%
ENDFUNCTION;   Fin de CT_B_RECHERCHE%
;=============================================
```

ANNEXE 4 : extrait du dossier d'analyse fonctionnelle du projet STIVE

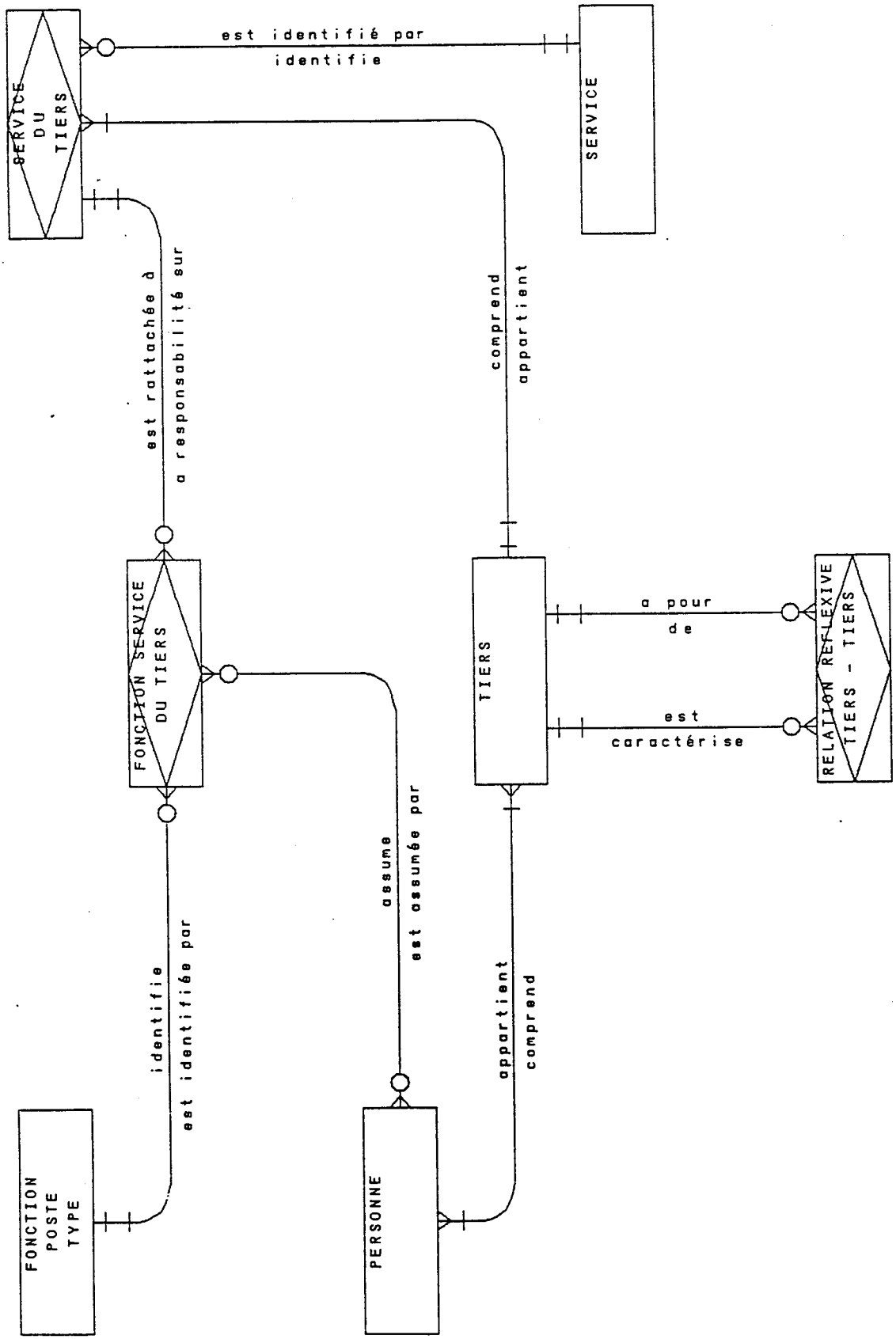
Extrait du dossier d'analyse fonctionnelle du projet STIVE à partir duquel a été réalisé le projet pilote Prostive, suivi du MCD et MLD du projet pilote.

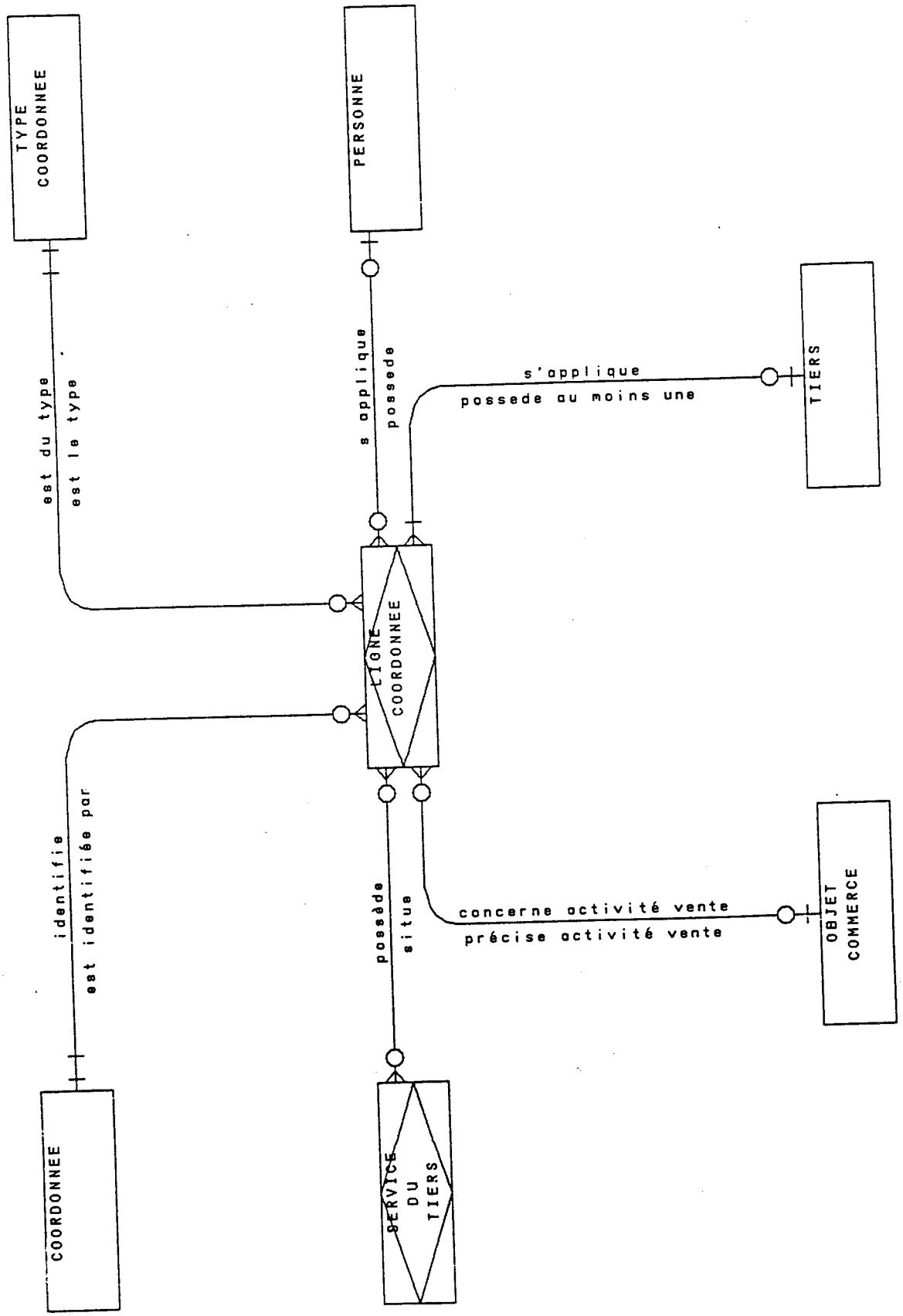

ACTION COMMERCIALE
GERER TIERS
GERER PERSONNE
GERER PARC
GERER COMMERCE TIERS
GERER NOTE
GERER PROJET
GERER CONDITION TIERS/ ACTIVITE
GERER MOT DE PASSE
ACTION COMMERCIALE
January 30, 1992 9:33:28



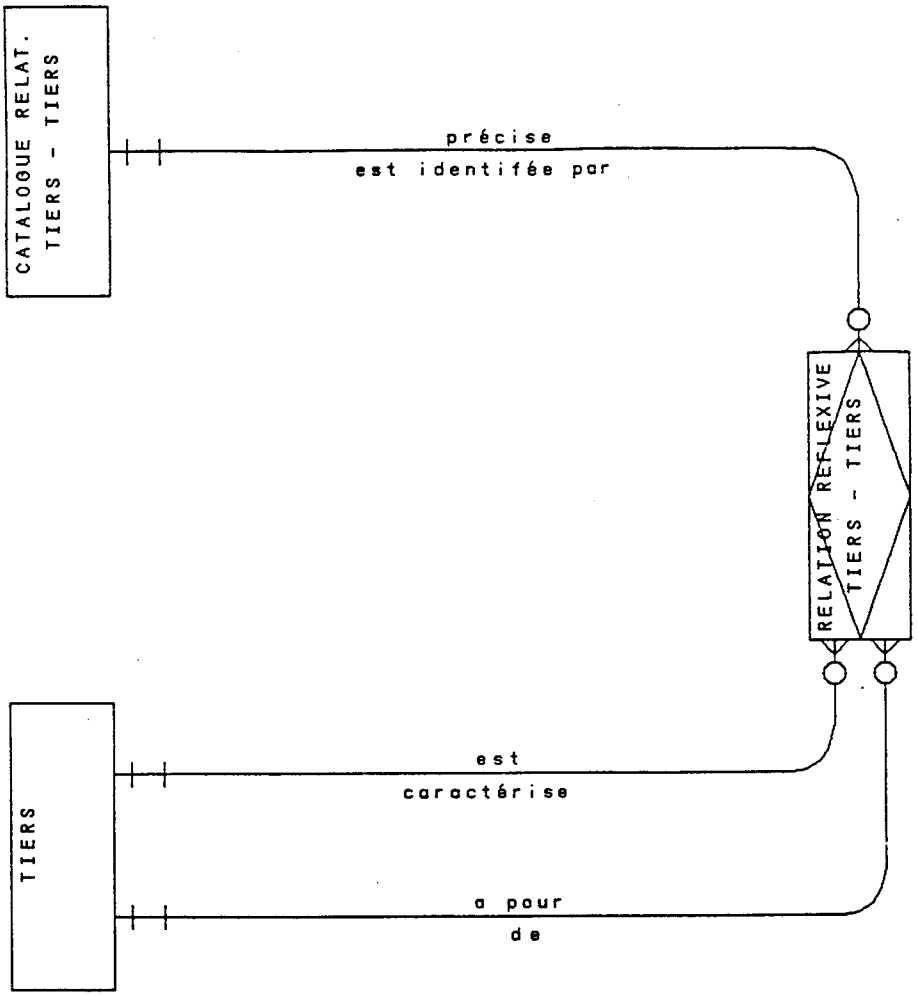
coractérisation tiers

November 12, 1991 10:36:28


SERVICE DU TIERS
est identifié par
identifie
SERVICE
est rattachée à
a responsabilité sur
comprend
appartient
FONCTION SERVICE DU TIERS
TIERS
a pour
de
est
caractérise
RELATION REFLEXIVE TIERS - TIERS
identifie
est identifiée par
assume
est assumée par
appartient
comprend
FONCTION POSTE TYPE
PERSONNE

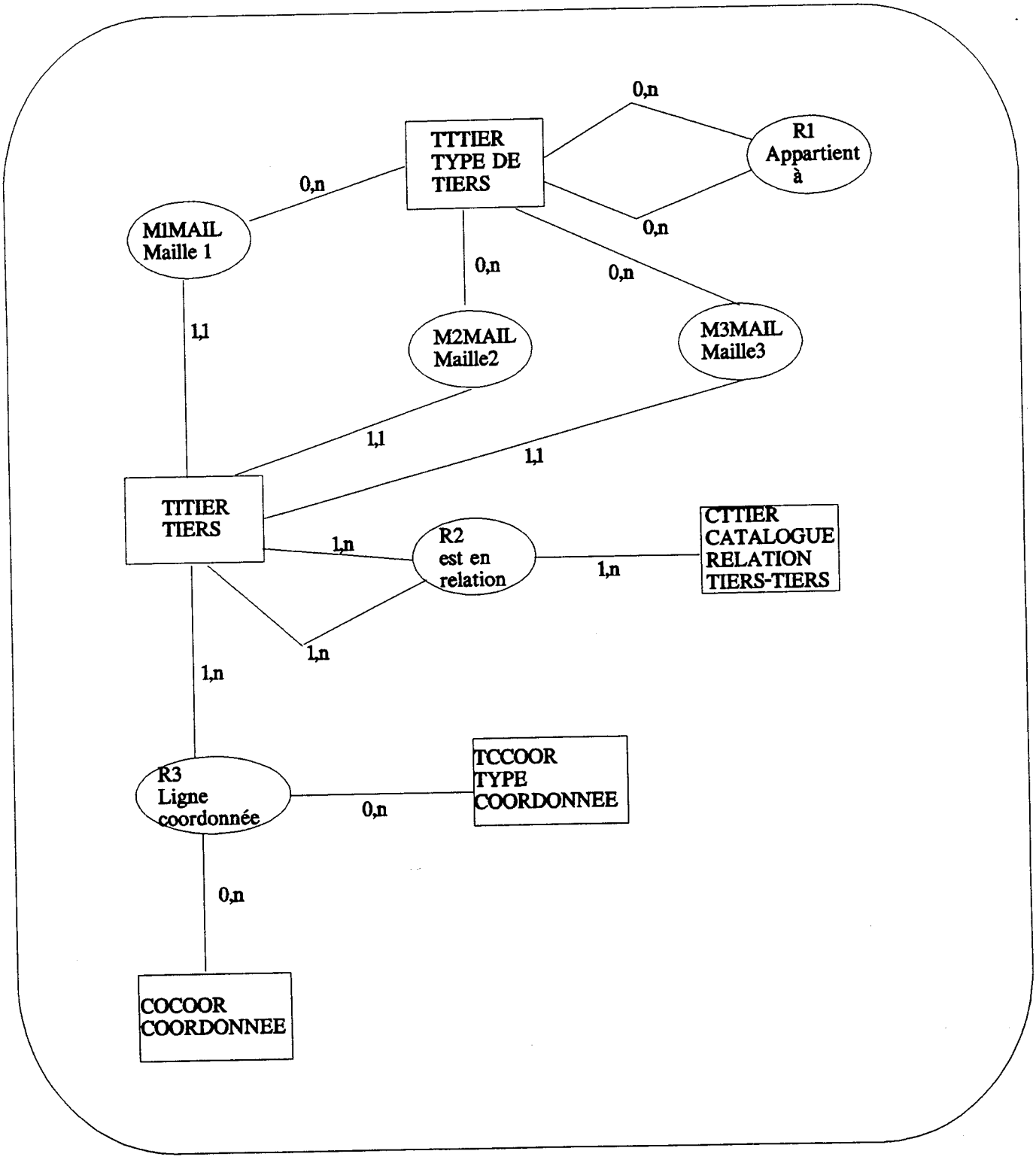

TYPE COORDONNEE
PERSONNE
TIERS
est du type
est le type
s applique
possede
s'applique
possede au moins une
LIGNE COORDONNEE
identifie
est identifiée par
possède
situe
concerne activité vente
précise activité vente
OBJET COMMERCE
COORDONNEE
SERVICE DU TIERS

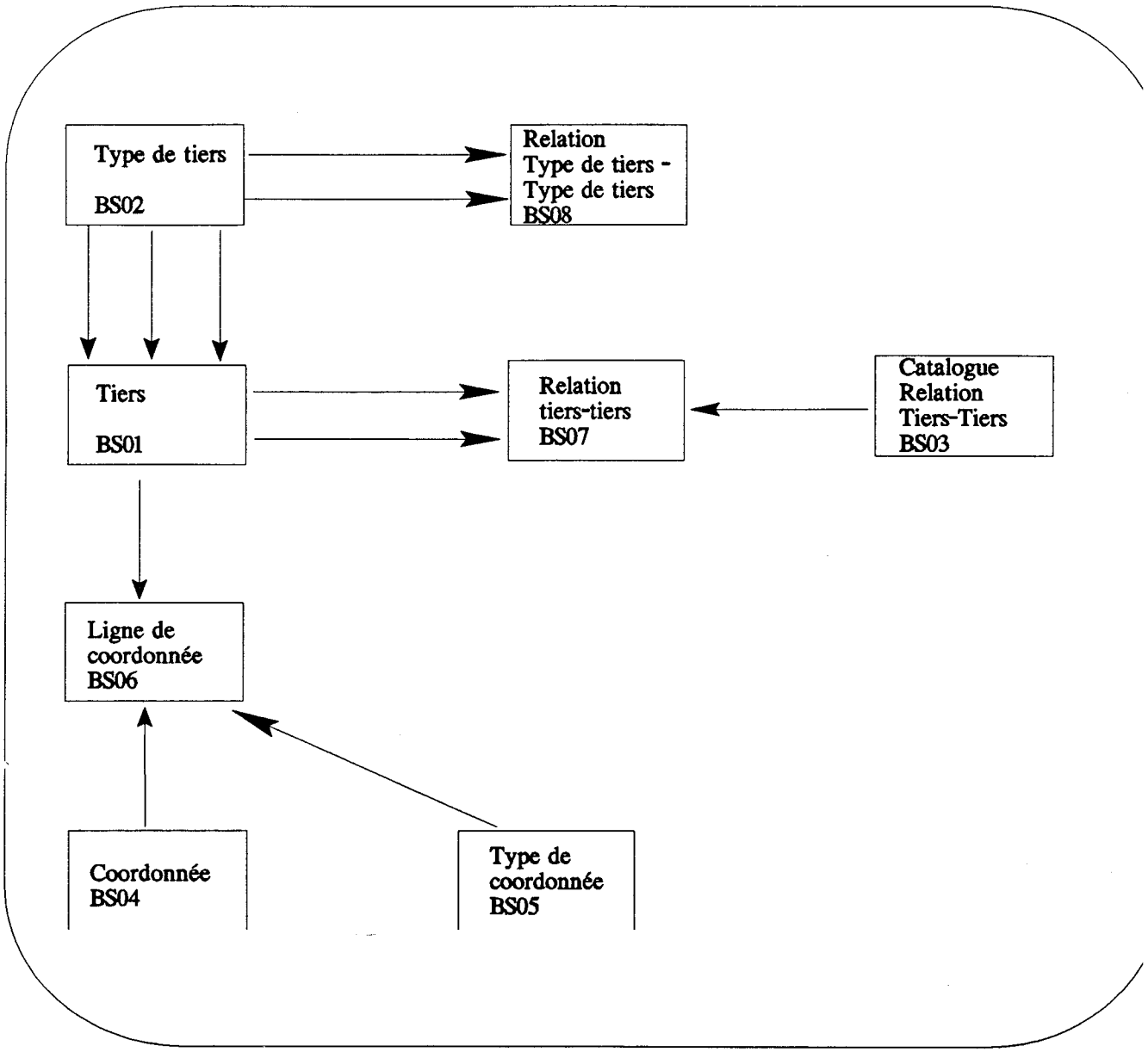
coordonnée



tiers renvois

November 14, 1991 15:24:17


TTTIER
TYPE DE
TIERS
R1
Appartient
à
0,n
0,n
0,n
0,n
0,n
M1MAIL
Maille 1
M2MAIL
Maille2
M3MAIL
Maille3
1,1
1,1
1,1
TITIER
TIERS
1,n
R2
est en
relation
1,n
CTTIER
CATALOGUE
RELATION
TIERS-TIERS
1,n
1,n
R3
Ligne
coordonnée
0,n
TCCOOR
TYPE
COORDONNEE
0,n
COCOOR
COORDONNEE


Type de tiers
BS02
Relation
Type de tiers -
Type de tiers
BS08
Tiers
BS01
Relation
tiers-tiers
BS07
Catalogue
Relation
Tiers-Tiers
BS03
Ligne de
coordonnée
BS06
Coordonnée
BS04
Type de
coordonnée
BS05

GLOSSAIRE

ATELIER DE GENIE LOGICIEL ou AGL

Un ensemble intégré d'outils, conçus pour être utilisés conjointement afin d'automatiser le cycle de vie complet du logiciel : l'analyse, la conception, la réalisation et les tests.

CASE

Computer Aided Software Engineering

Terme générique désignant l'ensemble des progiciels susceptibles d'apporter une aide durant le développement d'une application, sa maintenance, et la gestion de projet associé.

CUA

Common User Access - Interface commune d'accès

Composante de l'architecture unifiée d'IBM.

"Il s'agit d'une série de spécifications standards concernant la conception et l'utilisation d'écrans ainsi que des techniques d'interaction avec l'utilisateur ayant pour but de faciliter l'utilisation de ces applications et le transfert de connaissances de l'une à l'autre." [24]

DCE

Distributed Computing Environment

Ensemble de services d'interfaces élaborées par l'OSF qui définissent la technologie de base, permettant aux constructeurs qui l'adopteront de proposer les éléments nécessaires aux développements d'applications distribuées en environnement hétérogène.

EVENEMENT

Stimulation ou action à laquelle une procédure réagit dans le cadre d'une application interactive. Les événements de niveau contrôle comprennent les pressions sur un bouton ou les saisies dans un champ de données. Les événements de niveau fenêtre comprennent la réception d'un message en provenance d'une autre fenêtre.

INTEROPERABILITE

Aptitude, pour des systèmes de fournisseurs différents, à échanger de l'information entre eux.

ISO

International Standards Organization

Organisation internationale de normalisation. Institut de l'ONU créé en 1946 pour prendre en charge les questions de normalisation internationale dans tous les domaines sauf la technologie électrique et électronique.

Cette organisation regroupe 91 pays.

NORMES

Les normes et recommandations fournissent des spécifications détaillées concernant la structure et les fonctionnalités d'un matériel ou d'un logiciel. Ces spécifications sont définies par des organismes internationaux officiels. (Voir aussi Standard).

OSI

Open System Interconnection

Spécification d'un protocole de communication en 7 niveaux. C'est le modèle de référence pour l'interconnexion de systèmes ouverts. Il est défini par le norme ISO 7498.

OSF

Open Software Foundation

Consortium international regroupant des fournisseurs de matériels et de logiciels. Il a pour vocation de définir des standards et de commercialiser des produits réalisés à partir d'apport de ses membres (Hewlett Packard, IBM, Nixdorf, Digital, Bull, Apollo, Philips, Siemens). Les premières réalisations : l'interface graphique utilisateur Motif, l'architecture distribuée DCE.

PORTABILITE

Aptitude, pour des systèmes de fournisseurs différents, à offrir la même interface au niveau du système d'exploitation ou des applicatifs.

POSIX

Portable Operating System Interface for Computer Environnements

Norme de système d'exploitation développée par la commission IEEE de l'ANSI (American National Standard Institute) dans le but de garantir la portabilité des codes sources.

La norme Posix regroupe un ensemble de procédures, de commandes et d'appels standards que doit posséder un système d'exploitation pour assurer la portabilité des programmes d'une machine à une autre.

Des applications Unix à la norme Posix seront portables sur les machines supportant Posix.

RUN-TIME

Bibliothèque d'exécution d'un langage qui peut être liée lors de l'édition de liens de l'application ou chargée en mémoire sous la forme d'un utilitaire résident.

Un run-time est souvent payant, le fournisseur peut réclamer une redevance d'utilisation sur chaque machine faisant tourner l'application nécessitant ce run-time.

SQL

Standard Query Language

Langage, développé à l'origine par IBM, qui permet d'accéder aux données stockées dans une base de données relationnelle.

Ce langage est aujourd'hui normalisé et très largement répandu.

STANDARD

Ce sont des "normes de fait", ces spécifications n'émanent pas d'un organisme officiel de normalisation. C'est le concepteur du produit concerné qui fournit les détails de son implémentation.

UNIX est un standard, POSIX est une norme.

TCP/IP

Transmission Control Protocol/Internet Protocol

Protocole de communication couvrant les niveaux 3 et 4 du modèle OSI de réseaux. C'est un standard de fait.

BIBLIOGRAPHIE

Cette liste d'ouvrages et d'articles est non exhaustive et permet de poursuivre une recherche plus approfondie sur le sujet.

[1] ARTHUR ANDERSEN *Foundation cooperatif Version 1.01 - Présentation générale* - - Septembre 1991

[2] ARTHUR ANDERSEN *Foundation for cooperative processing- Application development Guide*- Version 1.0 - 1991

[3] BAL HENRI *Programming distributed systems* , Prentice HAll, Silicon Press, 1990

[4] BALTER R., J. BERNADAT, D. DECOUCHANT, A. DUDA, A. FREYSSINET, S. KRAKOVIAK, M. MEYSEMBOURG, P. LE DOT, H. NGUYEN VAN, E. PAIRE, M. RIVEILL, C. ROISIN, X. ROUSSET DE PINA, R. SCIOVILLE, G. VANDOME *Design and implementation of GUIDE, an object-oriented distributed system* Rapport technique 1-90, BULL-IMAG, 16 Novembre 1990

[5] BANATRE J.P., M. BANATRE, *Les systèmes distribués - Expérience du projet GOTHIC,* Paris, InterEditions, 1990.

[6] BANATRE J.P., S. KRAKOWIAK, R. BALTER *Construction des systèmes d'exploitation répartis*, Toulouse, INRIA Collection didactique.

[7] BOUY R., P. VIRIEUX *Modèle client-serveur pour le traitement coopératif* - Troisièmes journées internationales - Le génie logiciel & ses applications - Toulouse, 3-7 Décembre 1990

[8] BRUCE GLEN - *Cooperative Processing Strategy* - Guide 79, Anaheim, Session DB-4643- Guide Bulletin Issue n° 16 - Novembre 1991, pp22-37

[9] CGI *CGI et la production d'applications coopératives* - Position paper - Version 1 - Septembre 1991

[10] CREUSOT P. *La télémaintenance par réseau* Le Monde Informatique 25 Mai 1992, p I-III

[11] DAVY P., *Méthodes de conception - L'adaptation aux nouvelles architectures*, 01 Informatique, 20 Novembre 1992.

[12] DE WITT DAVID J., P. FUTTERSACK, D. MAIER, F. VELEZ *A study of three alternative workstation-server architectures for object oriented database systems.* Rapport technique Altaïr 42-90, 19 Janvier 1990.

[13] Dossier traitement coopératif - *Le Monde Informatique* - N° 499 - 1er Octobre 1990, pp31-46

[14] ERGIN B., J. KOULOUMDJIAN *COOPE : méthode de conception de systèmes d'information* - Congrès INFORSID 92 - 19-22 Mai 92 Clermond-Ferrand

[15] FITOUSSI B. - Les enjeux de l'architecture client/serveur - *Processeurs* - 5 Octobre 1991, pp57-58

[16] GARDARIN G., P. VALDURIEZ, *SGBD avancés : bases de données objets, déductives, réparties*, Paris, Eyrolles, 1990.

[17] GIRAUDIN J.P. *Méthodes d'Analyse et de Conception des Systèmes d'Information* Mémoire d'Habilitaion à Diriger des Recherches , Rapport de recherche Aristote, SUR004, LGI-IMAG, Grenoble, Octobre 1990.

[18] GUPTA *Database administrator's guide - SQLBASE Version 5.0,* 20-2121-1002, p31-38, 1992

[19] HEWLETT PACKARD *International Users Conference - Toward a client/serveur Concept* - P. Rocca - Hamburg 6 September 1991

[20] HEWLETT PACKARD *L'architecure client/serveur* - Séminaire Hewlett Packard - Paris 9 et] 10 Avril 1992

[21] HEWLETT PACKARD *Panorama des normes et standards dans les systèmes ouverts* - Pré Catelan - 2 Octobre 1991

[22] HEWLETT PACKARD *The HP 3000 Open System Environment* White paper - January 1992

[23] IBM *SAA Un guide d'évaluation d'application* - Réf : GF11.0600.00 - 1er Trimestre 1990

[24] IBM *Systems Application Architecture - Common User Access Advanced Interface Design Guide* -- 1989 -

[25] INFOSYS *Les outil, les réseaux et les serveurs bases de données-* Conférence client/serveur - Paris 29 et 30 Octobre 1991

[26] KRAKOWIAK S., M. MEYSEMBOURG, J. NGUYEN VAN, M. RIVEILL, C. ROISIN, X. ROUSSET DE PINA *Design and implementation of object-oriented, strongly typed language for distributed applications* Rapport technique 2-90, BULL-IMAG, 11 Décembre 1990

[27] LORIN HAROLD *Aspects of distributed computer systems* New York, Wiley-Interscience Publication, July 1980

[28] LUMETT K. *Cinq logiciels de télémaintenance sur PC* Le Monde Informatique 3 Février 1992, p 18.

[29] MARTIN D. *Etude de SGBD/L4G- ORACLE version 6* Laboratoire de Génie Logiciel - Feucherolles 16 Juin 1991

[30] MAUPAS P. LEPONT S. - *Les grands acteurs de la standardisation* - 01 Informatique - N° 1204 - 20 Mars 1992, p33

[31] MICHNOFF F. - *Meta Group : quel modèle client-serveur ?* - Processeurs - 5 Novembre 1991, pp40-42

[32] PIERRE SAMUEL - *Réseaux locaux - Fondements implantation et études de cas* - Québec - Eyrolles - 1991.

[33] O.R. - *Cinq étapes vers le modèle client-serveur* - Le Monde Informatique - 04 Mai 1992, p11

[34] TARDIEU H., A. ROCHEFELD, R. COLETTI, G. PANET, G. VAHE *La méthode Merise : démarche et pratiques* Editions d'Organisation, Paris, 1985.

[35] TEAMLOG *Etude SGBD : DB2, INGRES, INFORMIX, ORACLE, SYBASE* Mars 1991

[36] TRANSARC CORPORATION *Encina fron Transarc - Enterprise Computing in a New Age* - Product Overview - 1991 - Pittsburgh, PA