Système et langage portable pour le traitement des application réparties

Xuan Dang Nguyen

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
Génie Informatique

par

NGUYEN XUAN DANG

SYSTEME ET LANGAGE PORTABLE POUR LE TRAITEMENT DES APPLICATIONS REPARTIES

Thèse soutenue le 9 mars 1978 devant la Commission d'Examen :

Président : L. BOLLIET

Examinateurs : A. LAPLACE
G. SERGEANT
H. ZIMMERMANN

Rapporteur : J.P. VERJUS

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Monsieur Philippe TRAYNARD : Président

Monsieur Pierre-Jean LAURENT : Vice Président

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BLOCH Daniel	Physique du solide
	BONNETAIN Lucien	Chimie minérale
	BONNIER Etienne	Electrochimie et électrométallurgie
	BOUDOURIS Georges	Radioélectricité
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	DURAND Francis	Métallurgie
	FELICI Noël	Electrostatique
	FOULARD Claude	Automatique
	LESPINARD Georges	Mécanique
	MOREAU René	Mécanique
	PARIAUD Jean-Charles	Chimie-Physique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	POLOUJADOFF Michel	Electrotechnique
	SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M.	ROUXEL Roland	Automatique

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BOUVARD Maurice	Génie mécanique
	COHEN Joseph	Electrotechnique
	LACOUME Jean-Louis	Géophysique
	LANCIA Roland	Electronique
	ROBERT François	Analyse Numérique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	ANCEAU François	Mathématiques appliquées
	CHARTIER Germain	Electronique
	GUYOT Pierre	Chimie minérale
	IVANES Marcel	Electrotechnique
	JOUBERT Jean-Claude	Physique du solide
	MORET Roger	Electrotechnique nucléaire
	PIERRARD Jean-Marie	Mécanique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme.	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan	Automatique

CHERCHEURS DU C.N.R.S. (Directeur et Maîtres de Recherche)

MM.	FRUCHART Robert	Directeur de Recherche
	ANSARA Ibrahim	Maître de Recherche
	CARRE René	Maître de Recherche
	DRIOLE Jean	Maître de Recherche
	MATHIEU Jean-Claude	Maître de Recherche
	MUNIER Jacques	Maître de Recherche

Je tiens à rendre un hommage si leucieux à la mémoire de Mr J. du Masle qui m'a accueilli dans son équipe de recherche en 1975 et qui m'a initié aux problèmes réseau.

Je remercie Mr Bolliet, professeur à l'université de Grenoble qui m'a fait l'honneur de présider le jury de cette thèse et qui a manifesté beaucoup d'attention à l'équipe Réseaux d'ordinateurs et télé-traitement.

Le soutien critique et les encouragements de Mr J.P. Verjus, professeur à l'université de Rennes, m'ont beaucoup aidé dans la rédaction de cette thèse. La manière dont il a suivi et corrigé le manuscrit de cet ouvrage m'a montré comment doit être conçu le rôle d'un rapporteur. Pour cela, je tiens à lui exprimer ma gratitude.

Dès mes premières approches du réseau Cyclades j'ai entendu parler de Mr H. Zimmermann. Directeur de ce projet, entre autres activités scientifiques, il a été et est un précieux guide pour des équipes de recherche comme la nôtre. Je tiens à le remercier pour

l'intérêt qu'il porte à ce travail et les conseils critiques apportés lors de la rédaction de la thèse.

Mr A. Laplace, maître de conférences à l'université de Paris. 11, alors qu'il était encore à l'Imag comme responsable de l'équipe Langages Formels, a guidé mes premiers pas dans une équipe de recherche universitaire en informatique. Je lui suis redevable de beaucoup de choses et je ne suis pas prêt de l'oublier. Qu'il trouve ici un grand merci pour avoir bien voulu juger ce travail.

La recherche en réseau est par définition un travail d'équipe. J'ai la chance de collaborer avec Mr J. Sergeant, responsable de l'équipe Réseaux d'ordinateurs et télétraitement et Mr V. Quint, ingénieur de recherche IRIA affecté à Grenoble. Le rôle modérateur et clairvoyant de Mr Sergeant a permis à l'équipe Réseaux de l'ENSIMAG de travailler dans les meilleures conditions possibles après la disparition de Mr J. des Masle. Je le remercie d'avoir bien voulu faire partie du jury.

Le contenu de ce travail doit donc beaucoup à tous ceux avec qui j'ai pu travailler ou échanger des idées pendant ces dernières années ; je me dois de mentionner tous mes collègues du projet SIGOR : Mr J. Martins, Mr G. Sergeant et Mr P. Wilms, sans oublier Mr J.C. Chupin, directeur du Centre Scientifique CII-HB et Mr J. Seguin, du CICG.

Je remercie Mme C. Chaland de tout le mal qu'elle s'est donné dans la frappe intermédiaire et finale de cette thèse.

Je remercie enfin tout le personnel du service de reprographie pour la réalisation matérielle de cet ouvrage.

à ma Mère

à Martine Richard et C. K.

CHAPITRE A : INTRODUCTION ET GENERALITES

1. Les réseaux informatiques
2. Le réseau Cyclades
3. La Machine Réseau Logique SIGOR
4. Un système de multiprogrammation spécialisé en téléinformatique SYNCOP : rappels importants
5. Système et Langage Portable pour le traitement des applications réparties

CHAPITRE B : SYSTEME D'INTERPRETATION GENERALISEE ORIENTE RESEAU

1. Objectifs
2. Présentation du langage LI
   - 2.1. Introduction
   - 2.2. Langage de type procédural
   - 2.3. Vocabulaire du LI
   - 2.4. Facilités pour la programmation séquentielle
   - 2.5. Facilités pour la programmation parallèle
   - 2.6. Remarques
3. Architecture de l'interpréteur et du sous-système interpréteur
   - 3.1. Introduction
   - 3.2. Structure des données de l'interpréteur
     - 3.2.1. Description d'une procédure
     - 3.2.2. Représentation d'une procédure LI après la génération des données
   - 3.3. Noyau de l'interpréteur
     - 3.3.1. Introduction
     - 3.3.2. Etablissement du noyau
   - 3.4. Interface avec le réseau
     - 3.4.1.
     - 3.4.2. Le processus Concierge
     - 3.4.3. Le nom réseau d'un processus interpréteur
     - 3.4.4. Interface avec la station de transport

CHAPITRE A

INTRODUCTION ET GÉNÉRALITÉS

1. RÉSEAUX INFORMATIQUES

Les réseaux informatiques sont conçus actuellement pour offrir aux utilisateurs un service réparti des données et des traitements. Ils mettent en jeu deux domaines, hier encore nettement séparés, que sont l'informatique et la télécommunication.

Selon une définition proposée par L. POUZIN (POUZIN 77), les fonctions d'un réseau informatique appartiennent à trois grandes classes :

. traitement, i.e. toutes fonctions liées à l'exécution des programmes d'une application,

. transmission, i.e. l'infrastructure nécessaire au déplacement des informations d'un équipement informatique à un autre,

. distribution, i.e. les terminaux et les matériels ou les logiciels qui en assurent la gestion.

L'objectif du travail présenté ici est de proposer un modèle de Machine Réseau Logique (CHUPIN 77) qui se place dans la première classe : il définit un ensemble de fonctions et de mécanismes élémentaires qui aident à la répartition dans l'exécution des programmes d'une application réseau.

Ce travail est en cours de réalisation sur le réseau CYCLADES, mais logiquement sa définition n'est pas fonction des caractéristiques de CYCLADES. Dans cette optique, il est intéressant de constater quels sont les différents types de réseaux existants avant de faire une brève présentation de CYCLADES.

Dans l'évolution des réseaux d'ordinateurs, on a pu constater les principales étapes suivantes :

. réseau étoilé dont les exemples sont multiples (ils sont souvent orientés vers la connexion à un ordinateur central de terminaux à distance),

. réseau général d'ordinateurs qui permet l'interconnexion d'un vaste ensemble de systèmes informatiques avec différents services et possibilités; citons par exemple deux réseaux très connus : ARPANET et CYCLADES,

. réseau qui tend à présenter à l'utilisateur un système d'exploitation unique avec des ressources traitées automatiquement; ce type de réseau est encore au stade de la recherche (RSEXEC (FORSDICK 77), NSW (MILLSTEIN 77) en sont des exemples)

Dans les dernières années, se sont développés beaucoup de réseaux locaux (qui font partie des deux dernières classes de réseaux) qui réalisent l'interconnexion de gros, petits et micro-ordinateurs, des terminaux et d'autres périphériques en intégrant le matériel, le logiciel et les canaux de communication. Citons le réseau DCLN (LIU 77) et CMM* (SWAN 77).

2. Le Réseau Cyclades

Le réseau général CYCLADES (POUZIN 75) n'est pas conçu pour une application spécialisée. Il est conçu pour pouvoir supporter toute une variété d'applications, mettant en jeu l'ensemble des ordinateurs et des terminaux du réseau, grâce à la mise à la disposition des usagers d'un certain nombre de protocoles permettant à des systèmes spécifiques et hétérogènes de communiquer entre eux.

Le réseau CYCLADES se compose de trois niveaux :

* *niveau de communication de base* : ce niveau est réalisé par un sous-réseau de commutation de paquets appelé CIGALE (GRANGE 76). Il est composé de noeuds (MITRA 15) reliés par des lignes de télécommunication (4,8 à 48 kb/s) et assure le transport des informations. Comme tout système, CIGALE peut être victime de pannes et offre donc une probabilité élevée de livraison de paquets, mais exceptionnellement certains paquets peuvent être perdus ou dupliqués. CIGALE offre un service de transport de paquets indépendants (datagramme dans la terminologie du CCITT). Chaque paquet est transmis dans les meilleurs délais. Si un paquet est retardé (par exemple par une erreur de ligne nécessitant une retransmission), les paquets suivants ne sont pas retardés pour autant. Le séquencement des informations au départ n'est donc pas garanti à l'arrivée.

* *niveau de transport* : bien qu'il soit possible de travailler uniquement avec le sous-réseau CIGALE, les fonctions de communication du sous-réseau ne sont pas destinées à être directement offertes à un utilisateur système. Un protocole de bout en bout de transport d'informations a été défini (ZIMMERMANN 76); il assure principalement les fonctions suivantes :

. sous-adressage et multiplexage des conversations,

. contrôle d'erreurs,

. contrôle de flux.

Chaque site voulant se connecter au réseau CYCLADES doit réaliser la mise en oeuvre de ce protocole, la plupart du temps d'une manière logicielle. Cette pièce, logicielle dans le cas des deux grands ordinateurs de Grenoble qui offrent leurs services sur le réseau CYCLADES, s'appelle une station de transport (DANG 1-76) (ANDRE 77).

L'annexe 2 décrit très brièvement la mise en oeuvre d'une station de transport sur l'IBM 360/67 du CICG selon le protocole de bout en bout de transport actuellement utilisé sur le réseau (ELIE 75). Cette station de transport et les autres logiciels de même nom supportent le système de distribution de services du réseau : logiciels des concentrateurs, des serveurs temps partagé, des serveurs traitement par lots, .. L'existence d'une station de transport est aussi absolument nécessaire pour la mise en oeuvre d'une Machine Réseau Logique (CHUPIN 77), bien qu'elle soit masquée pour l'utilisateur de cette machine. Le protocole de bout en bout de transport constitue un premier niveau d'échange sur lequel peuvent être construits d'autres niveaux spécifiques, tels le protocole client-serveur, le protocole d'appareil virtuel (ZIMMERMANN 75), le protocole de transfert de fichier (GIEN 75). L'utilisation du réseau via ces protocoles est primitive et réservée aux spécialistes.

Cependant, il est souhaitable d'offrir à un utilisateur réseau quelconque un outil approprié pour une conception et une réalisation aisées de ses applications réparties ou pour une extension au domaine réseau de ses applications locales.

* Voilà pourquoi dans le réseau CYCLADES apparaît un troisième niveau dit d'*applications réparties* où l'on présente aux utilisateurs des facilités de programmation séquentielle, parallèle et répartie pour la définition algorithmique et ultérieurement la mise au point de leurs applications.

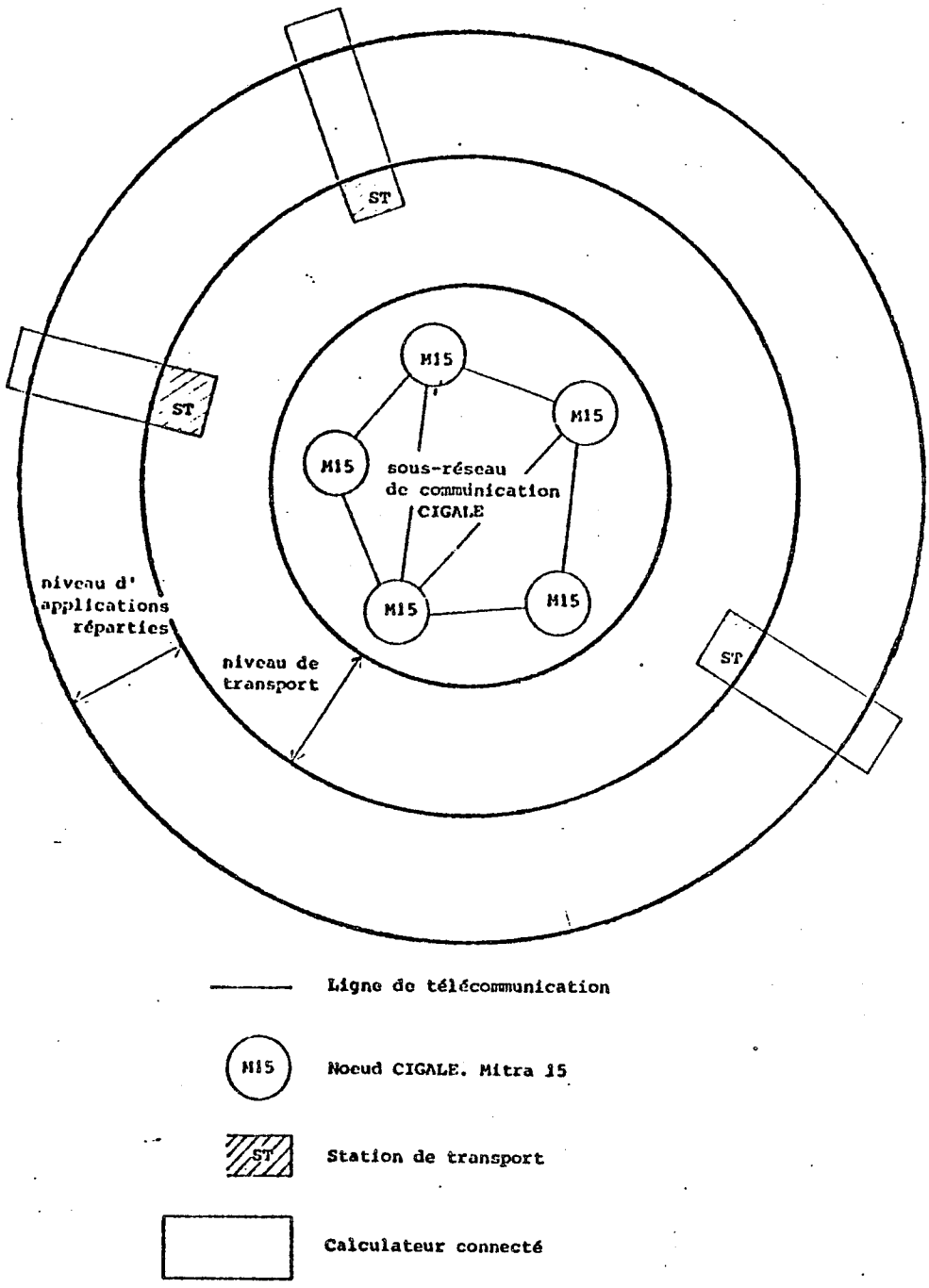


figure 1 - Les 3 niveaux de CYCLADES

3. Machine Réseau Logique

On distingue une application réseau d'une application locale dans la mesure où la première met en jeu deux sites ou plus, logiquement et/ou géographiquement disjoints.

Il y a deux types d'applications réseau :

. application centralisée,

. application répartie.

A l'inverse d'une application centralisée, une application répartie est une application réseau où les fonctions et les contrôles sont distribués géographiquement parmi les sites participants (CHUPIN 76).

Les problèmes posés par ce type d'applications se décomposent en deux classes :

. problèmes fonctionnels qui proviennent des objectifs fonctionnels de l'application,

. problèmes de contrôle qui sont relatifs à l'environnement réseau et qui sont pour la plupart indépendants de la nature de l'application.

En considérant d'abord le deuxième aspect, la démarche consiste à définir une Machine Réseau Logique qui matérialise les *objectifs d'environnement* (CHUPIN 74) : une Machine Réseau Logique et son langage d'opération jouent dans un réseau d'ordinateurs le même rôle que le logiciel de base dans un ordinateur classique.

Cette machine doit être dotée d'outils (mécanismes d'adressage et de contrôle) et d'objets spécifiquement réseau, pour permettre la définition et la mise en oeuvre des applications réparties (DANG 4-77).

Elle est elle-même répartie dans la mesure où chaque site participant possède une copie du logiciel de la Machine Réseau Logique.

Une telle machine possède sur chaque site un support de multiprogrammation. Ce support doit respecter les principes de base d'un (sous)système téléinformatique (DANG 1-77) et doit présenter, si possible, un interface transparent par rapport au site sur lequel il s'exécute.

4. Système (ou sous-système) de Multiprogrammation Spécialisé en Téléinformatique. Syncop : Principales Caractéristiques

Les principes de base d'un (sous)système téléinformatique sont les suivants :

. existence d'un grand nombre de processus,

. forte interaction entre les processus nécessitant une mise en oeuvre efficace du parallélisme,

. faible taux d'occupation de l'unité centrale par chaque processus activé,

. gestion des processus en temps réel nécessitant des actions déclenchées à l'aide de sonneries d'horloge,

. notion de l'attente de n événements sur une liste de p, mémorisation de l'attente de q processus sur un événement, synchronisation sur acquisition de ressources,

. méthode d'accès normalisée aux différentes ressources physiques.

Le Système Normalisé de COmmutation de Processus (SYNCOP) défini et réalisé sur plusieurs ordinateurs du réseau CYCLADES, se propose de répondre à ces principes. Les principaux objectifs sont :

. une adaptabilité maximale aux applications téléinformatiques,

. un interface utilisateur standardisé et propre,

. un ensemble de réalisation sur plusieurs machines.

Comme ces objectifs cadrent avec des besoins de la Machine Réseau Logique, SYNCOP est choisi comme support de multiprogrammation de référence pour la conception de la machine SIGOR décrite dans le chapitre B.

Les services de base proposés par SYNCOP sont les suivants :

. la gestion des processus,

. la mise en oeuvre du parallélisme entre ces processus (synchronisation, communication d'informations),

. la gestion de la mémoire,

. la gestion du temps,

. la gestion des ressources et

. la gestion des entrées/sorties.

Le lecteur intéressé se reportera aux références : (DANG 2-76), (DANG 1-77), (SEGUIN 76).

5. Système et Langage Portable pour le Traitement des Applications Réparties

La Machine Réseau Logique SIGOR (Système d'Interprétation Généralisée Orienté Réseau) propose un langage unique, transportable et interprétable, de type procédural, pour l'expression des applications réparties.

Elle se compose, sur chaque site participant, d'un sous-système interpréteur et de mécanismes d'aide à la répartition; l'ensemble étant géré par un (sous)système de multiprogrammation de type SYNCOP.

Le travail de définition du système SIGOR s'insère dans les travaux actuellement réalisés dans le domaine du *traitement distribué* dont voici une définition proposée par un groupe de spécialistes :

P. ENSLOW, D. FARBER, E. JENSEN, E. MANNING, A. VAN DAM, ".. distributed processing is defined as the ability to perform logical functions across geographically dispersed physical locations with complete freedom of movement of data and applications while sharing resource in an essentially systems integrated and systems controlled network .."
(VAN DAM 77).

La présentation du système SIGOR qui fait l'objet du chapitre B comprend :

. la description du langage d'opération du système,

. un survol de l'architecture du système avec les références aux travaux dont la conception de l'interpréteur de SIGOR s'est inspirée et

. la description des mécanismes d'aide à la répartition.

A propos de la mise en oeuvre de SIGOR, il faut remarquer les faits suivants :

. Comme SIGOR est lui-même réparti et qu'il entend faciliter le traitement des applications réparties, le logiciel de SIGOR sous-entend l'existence des logiciels de multiprogrammation (SYNCOP) et de mise en oeuvre de protocoles de transport (station de transport) sur le réseau,

. bien que non apparente dans cette présentation, une station de transport ou tout logiciel équivalent est le nécessaire et unique interface que possède et utilise une Machine Réseau Logique telle que SIGOR pour travailler avec le réseau. La mise en oeuvre d'une station de transport (DANG 1-76) est abordée brièvement dans l'annexe 2.

CHAPITRE B

SYSTÈME D'INTERPRÉTATION GÉNÉRALISÉE

ORIENTÉ RÉSEAU

1. OBJECTIFS

En tant que Machine Réseau Loqique, c'est-à-dire jouant le rôle d'un outil permettant l'écriture et la mise en oeuvre d'une application répartie, SIGOR vise les objectifs suivants :

* *transport de programme :*
  - . création dynamique de processus à distance,
  - . contrôle du transport d'un programme,
  - . contrôle de l'exécution distribuée;
* *communication d'informations entre processus :*
  - . communication d'informations typées entre processus,
    - communication implicite d'informations typées,
    - contrôle de la communication,
    - aspect sécurité,
  - . transfert explicite d'informations de processus à processus,
    - négociation de bout en bout pour l'ouverture d'une liaison,
    - communication explicite d'informations sur la liaison,
    - aspect sécurité;
* *expression du parallélisme entre processus :*
  - . aspect implicite géré par le système,
  - . aspect explicite géré par l'application.

Une fois que ces objectifs sont atteints, la mise en oeuvre de protocoles de haut niveau se trouvera facilitée. Par protocoles de haut niveau, il faut entendre par exemple par (RAYNER 77) :

. langage de commande réseau pour tout traitement par lots ou traitement interactif,

. méthode d'accès normalisée aux différentes bases de données,

. méthodes d'allocation de ressources réseau qui tend à banaliser ces ressources et présenter le réseau à l'utilisateur comme une simple machine générale donnée,

. exécution distribuée qui autorise le découpage d'un travail (job) en différentes parties; ces parties se déroulent sur de différentes machines et réalisent des transferts d'informations, tels le transfert de fichiers par exemple.

L'approche de la Machine Réseau Logique SIGOR s'effectue d'abord par la connaissance de son langage d'opération, le LI, qui propose des solutions aux objectifs désignés ci-dessus.

2. PRÉSENTATION DU LANGAGE LI

2.1. Introduction

Le langage LI peut être le résultat de la compilation d'un langage externe LE (langage de commande réseau, langage pour la mise en oeuvre des applications distribuées ou langage de requêtes des bases de données), ou celui de la macro-génération d'un langage de macro-instructions LM permettant l'écriture quasi-directe d'applications en LI * (FARZA 74).

Plusieurs raisons nous ont conduits à la définition d'un langage intermédiaire *à interpréter* :

. l'hétérogénéité du réseau nécessite un langage unique, transportable et capable d'être exécuté sur toutes les machines;

. la bonne compacité des programmes écrits en LI répond au souci de minimiser les transferts d'informations sur le réseau. Un programme écrit en langage interprétatif est généralement plus compact qu'un programme écrit en langage machine (rapport allant de 1 à 2,5 (CLEARY 69));

. l'écriture du compilateur se trouvera simplifiée : le code généré est adapté et proche du code source. La séparation du travail en deux composants (le compilateur et l'interpréteur) permet de réaliser d'abord l'interpréteur et la machine interprétative. Les applications réparties peuvent être écrites en macro-langage et générées en LI;

. la souplesse et la modularité de l'interprétation assurent l'extensibilité aux fonctions et aux mécanismes nouveaux. La machine interprétative est réalisée aussi dans cette optique.

---

* Ce macro-langage existe et s'appelle LM. Sa structure s'apparente à celle du macro-langage FANNY (GIEN 73). Il a été développé dans le cadre de la première version d'IGOR (FARZA 74) et est modifié pour générer du LI de l'actuelle version (SERGEANT 78) (GRIFFITHS 69).

Un des désavantages d'un langage intermédiaire interprétatif est sa relative lenteur d'exécution : le fait d'interpréter du LI au lieu de l'exécuter directement introduit une dégradation de performance élevée quant au temps unité centrale utilisé. Cependant, en l'état actuel des réseaux d'ordinateurs, ce fait ne handicape pas les services fournis qui se déroulent à des vitesses bien plus lentes :

. services réseau qui impliquent des télétransmissions et des entrées/sorties ,

. services locaux où il existe de nombreuses entrées/sorties.

Note :

Le langage LE n'est pas défini actuellement*. Faire des hypothèses sur le langage LI pour réaliser SIGOR revient directement à faire des hypothèses sur le futur langage LE ou sur l'ensemble des macro-instructions LM : par exemple, si le langage LI est un langage procédural, ce fait implique que le LE sera aussi procédural. Donc, la sémantique du LI imposera un certain nombre de contraintes à la définition de la syntaxe et la sémantique du LE.

Par contre, le LI est composé d'objets et d'instructions primitifs qui servent à traduire la sémantique du LE mais qui n'auront pas forcément d'équivalents dans le LE : par exemple, le fait que dans le LI se trouve l'instruction 'aller à' n'implique pas que la même instruction se retrouvera dans le LE.

---

* dans la suite du chapitre, quand on aura besoin d'écrire en LE, on utilisera une écriture en pseudo-PL1 (IBM c).

2.2. Langage de type procédural

Selon l'idée classique des procédures, définies dans les langages de la famille ALGOL ou PLi par exemple, on définit la structure d'un programme écrit en LI comme celle d'une procédure.

Dans la Machine Réseau Logique SIGOR, chaque procédure utilisateur répond à la définition globale suivante, avec des restrictions qui seront décrites ultérieurement :

. une *procédure* forme un *bloc* au sens ALGOL du terme. Des variables, d'autres procédures peuvent être définies comme locales à une procédure. Ceci implique que leurs existences ne sont connues qu'à l'intérieur de la procédure;

. une procédure est paramétrable : elle peut avoir globalement deux types de paramètres : *par valeur* et *par nom*;

. une procédure peut avoir une valeur comme résultat.

2.2.1. Procédure LI

* Toute définition de procédure comporte deux parties distinctes :

. *déclaration* : qui spécifie les paramètres formels, les déclarations de procédure et les variables locales,

. *instruction* : instructions LI qui forment le corps de la procédure.

* Toute procédure est déclarée à l'intérieur d'une autre procédure. La procédure la plus extérieure est considérée comme ayant été déclarée dans une procédure fictive dite d'initialisation.

* L'appel séquentiel d'une procédure depuis une autre procédure obéit à la règle non traditionnelle suivante : l'appel à une procédure englobante est interdit. En réalité, cette règle participe d'un principe plus général :

* Il est interdit d'utiliser les *globaux* à une exception près : dans l'appel aux procédures *standard* définies universellement.

Note :

Comme la procédure est considérée comme globale par rapport à elle-même, l'appel récursif d'une procédure est interdit en LI. En réalité, comme l'interprétation des procédures se réalise à l'aide des piles, on aurait pu autoriser l'appel récursif. Le fait d'interdire l'appel récursif provient

. du choix qui veut que la récursion dans un langage n'est pas un outil fondamental pour la programmation (DIJKSTRA 77), (BOUSSARD 77) et

. de la raison suivante : dans une procédure LI, il existe certaines variables dont l'identification est locale à la procédure mais qui doivent avoir une durée de vie globale si la procédure est appelée récursivement. Cette raison a conduit, semble-t-il, à interdire l'utilisation des variables de type fichier dans une procédure appelée récursivement en PASCAL (WIRTH 70). Pour simplifier le problème, dans le LI, on a interdit l'appel récursif en attendant une évolution de la structure du LI vers une structure de *module* ou de *segment* (VERJUS 75), (WIRTH 77) ; module à l'intérieur duquel on peut définir des variables *rémanentes* et des procédures appelables récursivement.

* L'appel *parallèle* d'une procédure obéit à la même règle que celle de l'appel séquentiel, mais a pour résultat de faire exécuter la procédure parallèlement à l'exécution de la procédure appelante.

2.2.2. Processus

Puisque SIGOR veut permettre une exécution en parallèle de programmes répartis sur plusieurs sites ou localisés sur un seul site, on définit chaque exécution en parallèle comme un *processus* (CROCUS 75).

Note :

SIGOR a donc besoin d'un support de multiprogrammation qui fait la commutation de processus (en l'occurence, c'est SYNCOP qui est choisi aussi bien pour la conception que pour la mise en oeuvre de SIGOR).

L'interpréteur du système SIGOR s'appelle IGOR; il se présente sous la forme d'un programme ré-entrant existant en un seul exemplaire par site participant. IGOR interprète donc toute procédure écrite en LI.

Communication entre processus

Les processus répartis sur un réseau ont nécessairement besoin de communiquer pour pouvoir coopérer, se synchroniser ou échanger des données. L'expression de cette communication pose les problèmes suivants :

a. d'ordre syntaxique et sémantique,

b. d'accès et de contrôle d'accès aux objets propres des processus,

c. de sécurité et de contrôle de bout en bout (ZIMMERMANN 76).

Une communication s'effectue sur un *objet de liaison* (ou par contraction une *liaison*) (FERRIE 76).

On peut proposer des solutions dynamiques (i.e. ayant lieu à l'exécution) à certains de ces problèmes (b. ou c. par exemple) à condition :

. d'avoir un graphe complet mémorisant les liaisons à un moment donné sur le réseau et

. d'offrir directement au processus utilisateur l'interface de la station de transport locale pour qu'il puisse gérer lui-même ses liaisons.

Ces conditions ont pour conséquences :

. de privilégier un site où se trouve le graphe complet des liaisons,

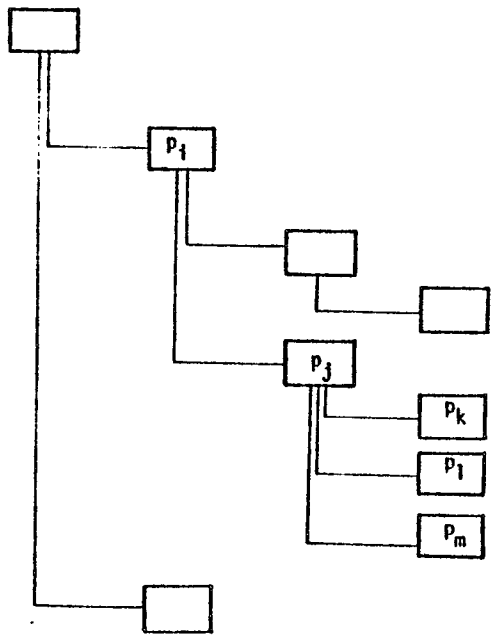
. de réduire l'expression de la communication sur les liaisons à l'utilisation d'un interface système, c'est-à-dire la rendre très élémentaire et

. de réduire les possibilités de contrôle statique.

Le système SIGOR, réparti sur tous les sites, n'élit pas de site central ou privilégié où un tel graphe pourrait se trouver.

Puisqu'il n'est pas possible de garder un graphe complet dans SIGOR, on peut conserver par site un graphe partiel représentant les liaisons vues localement par le site.

En poursuivant l'observation jusqu'au bout (pas de processus privilégié par site, etc..), on est amené à ne garder qu'un graphe partiel par processus; on définit une *hiérarchie de processus* avec une *relation de filiation père-fils* entre processus.



Chaque processus connaît son père et ses fils; ce qui se rapporte à cette connaissance de ses parents proches sera dit dans le reste de ce chapitre *implicite*.

Quand un processus veut établir une communication, sous forme de liaison directe, avec un autre processus, cette communication sera appelée *explicite*.

La *désignation de filiation* 'un tel processus est fils d'un tel processus' est faite à partir de la création d'un processus pour exécuter une procédure.

Définition :

A sa déclaration, chaque procédure se trouve statiquement déclarée dans une autre procédure. Prenons l'exemple suivant :

```
A : PROCEDURE ;
|
|   B : PROCEDURE ;
|   |
|   |
|   END B ;
|   C : PROCEDURE ;
|   |
|   |
|   END C ;
|
:
```

. La procédure B, déclarée dans A, est dite *affiliée* à A.

. Il n'y a aucune relation entre les procédures B et C affiliées à A.

. On ne peut appeler B que dans A.

La création d'un processus pour exécuter une procédure a lieu à l'appel parallèle d'une procédure (B par exemple) qui est soit *cataloguée* sur le site local, soit affiliée à une autre procédure (en l'occurence A) : le processus crée pour exécuter B est dit le *fils* du processus qui exécute A.

Durée de vie d'un processus :

Un processus p_j qui exécute une procédure B est vivant :

. quand il y a eu un appel parallèle de la procédure B dans la procédure A à laquelle B est affiliée,

. tant qu'il n'exécute pas l'instruction qui met fin à la procédure B (exemple, par association avec PL1 : END ou RETURN),

. tant que le processus père p_i (qui exécute A) n'exerce pas son droit de tuer p_j,

. tant que p_j n'exécute pas une instruction qui ne peut pas se réaliser (demande d'accès à un objet du père alors que la communication père-fils est coupée par le fait du réseau par exemple).

L'interdiction des globaux (y compris les procédures) implique qu'il n'y a pas d'environnement statique pour une procédure, donc facilite son transport sur un site distant.

La structure statique imbriquée du LI sert à exprimer :

. les relations de filiation entre processus et

. le choix de réaliser une certaine forme de communication (dite implicite) d'informations entre processus sur un réseau par partage de variables communes.

Le choix d'un langage de type procédural permet donc, dans le cadre du réseau, une définition simplifiée de la notion de transport de programme : *la procédure est l'unité de base servant à un transport.*

Note :

La (les) procédure(s) standard est la seule exception faite à la règle des globaux :

. la procédure standard est définie à l'extérieur de toute autre procédure,

. elle est appelable de n'importe où,

. elle est envoyée avec la procédure qui l'appelle pour être exécutée par le même processus que celle-ci.

Ce type de procédure est d'un emploi coûteux (à cause de son envoi systématique) et est prévu particulièrement pour résoudre le problème des reprises d'erreurs, le problème de procédures spécifiques de l'utilisateur (de codage, de compactage), pour définir de nouvelles fonctions et pour être utilisées à la place des procédures système présentes par défaut.

Dans une procédure standard, on ne peut appeler que les procédures de type standard.

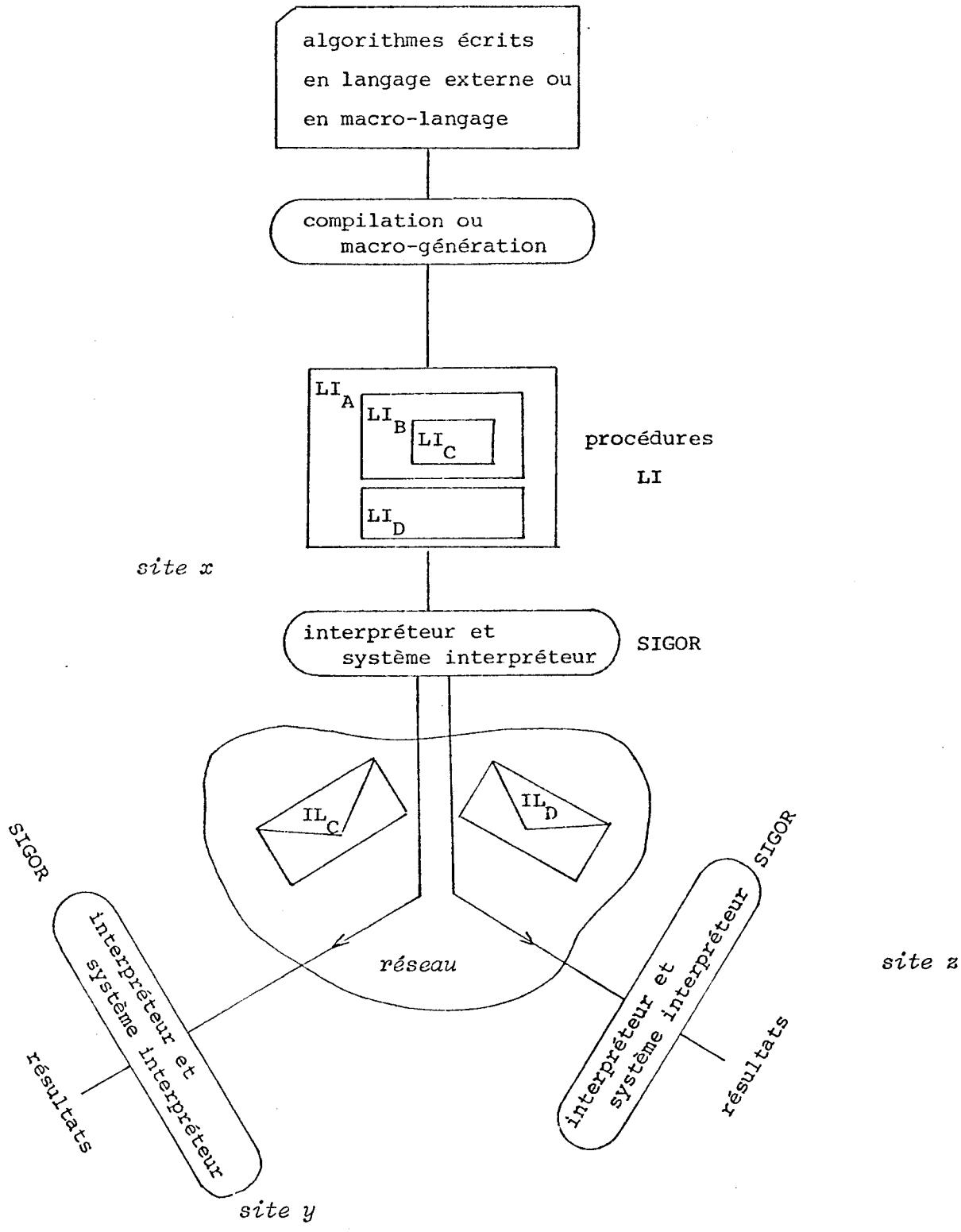


figure 2 - Applications Réparties

2.3. Vocabulaire du LI

Un programme (une procédure) en LI est composé de deux parties (voir figure 3) :

* la première partie est celle où l'on associe un mode à chaque nom d'objet; elle est dite de *déclaration*. C'est une suite finie d'octets qui décrit :

. les paramètres formels,

. les variables locales :

- de mode connu : logique, entier court, entier, adresse, chaîne de caractères, chaîne d'octets,
- de mode correspondant aux variables de SYNCOP : événement, bloc de réveil,
- de mode nouveau : fils, liaison, rendez-vous,

. les procédures affiliées.

Cette partie est décrite selon une structure codée et subit une pré-interprétation différente de l'interprétation des instructions LI.

* La deuxième partie est celle où l'algorithme est décrit; elle est dite d'*instruction*. C'est une suite finie d'octets qui représente les instructions LI.

On dispose des opérateurs suivants :

. arithmétiques et logiques,

. de traitement de chaînes de caractères et de chaînes d'octets,

. d'appel et de retour de procédure,

. de traitement du temps,

. de synchronisation,

. du lancement et de contrôle de processus,

. de communication explicite entre processus,

. de catalogage de procédure et

des opérateurs divers tels que les opérateurs de comparaison, de rupture de séquence, de chargement, ..

Leurs opérandes sont :

. soit des noms de variables : dans le LI, toute variable a pour nom un *repère* qui sert à accéder à la représentation de la variable,

. soit des adresses relatives dans le programme,

. soit des constantes qui ont les modes suivants : logique, entier court, entier, adresse, chaîne de caractères, chaîne d'octets, événement et bloc de réveil.

Notes :

. Le code de référence dans le travail sur SIGOR est le code EBCDIC; mais on aurait pu prendre aussi un autre ensemble de code, tel IA5 (ISO a).

. Les instructions LI sont commentées dans les sous-chapitres B.2.4. et B.2.5. Les détails techniques sur la représentation des variables se trouvent dans l'annexe 1.1.1. La liste détaillée des instructions se trouvent dans l'annexe 1.1.2.

Exemple :

La figure 3 représente schématiquement la génération d'un programme LI à partir d'un programme externe écrit en pseudo-PL1. La procédure LI de nom A est composée de sa partie dite de *déclaration* (du début jusqu'à la fin de la description de C) et de sa partie d'*instruction*.

La description de la procédure la plus externe d'un programme est dite *zone d'interprétation modulaire différée*; elle sera détaillée dans le sous-chapitre B.3.

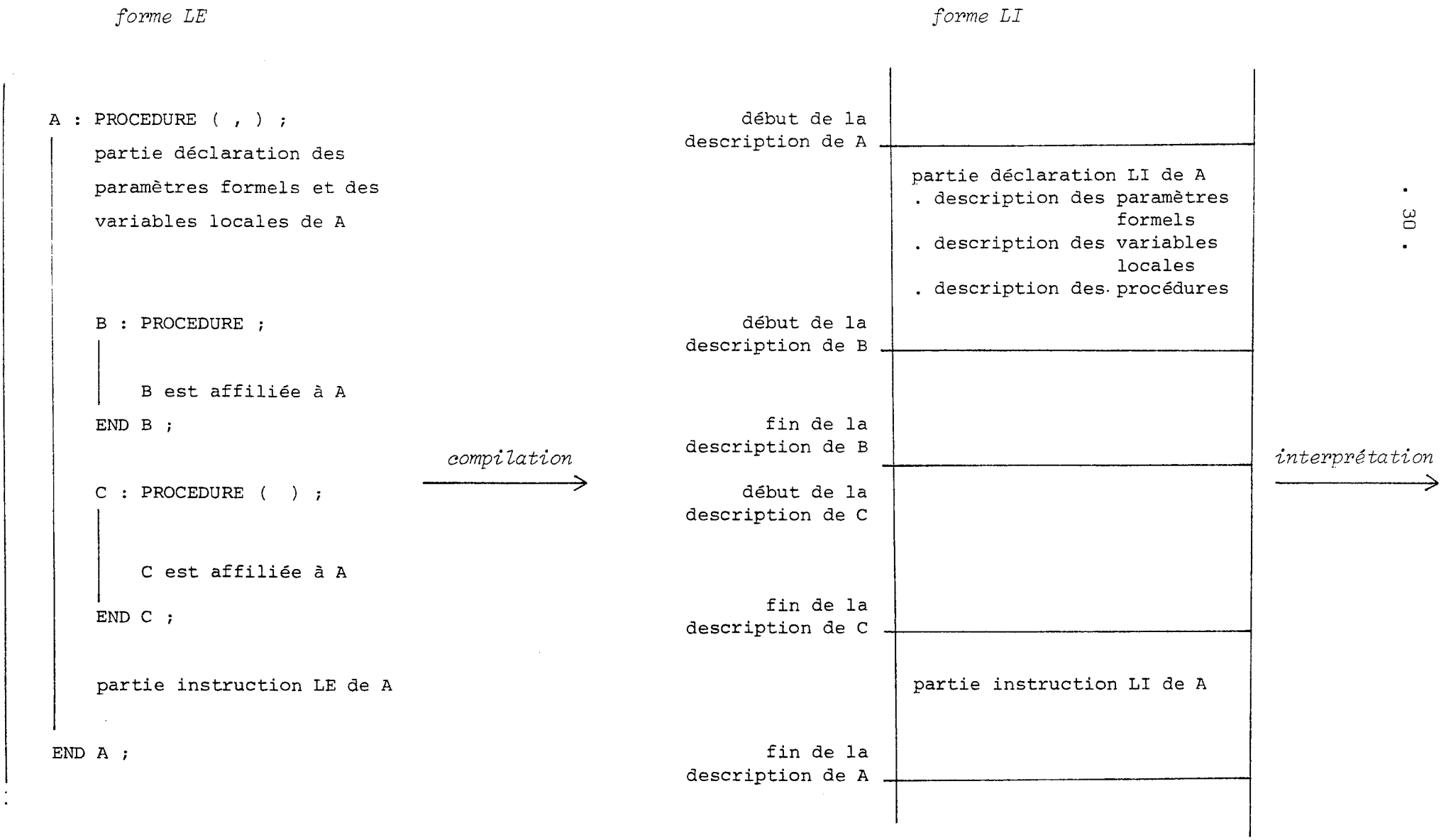


figure 3 - Description d'une Procédure LI

Paramétrage des procédures

Les procédures en LI peuvent avoir des paramètres qui sont de deux types :

. par valeur et

. par nom

* Appel par valeur

Si le paramètre effectif est une variable, l'appel par valeur est réalisé d'une manière différée. Le paramètre formel ne prend la valeur du paramètre effectif qu'au moment où il est utilisé pour la première fois dans la procédure. Si le paramètre effectif est une expression autre qu'une variable, le paramètre formel prend la valeur de l'expression à l'entrée de la procédure. On ne peut pas passer une fonction procédure comme paramètre effectif.

* Appel par nom

L'appel par nom est réalisé dans l'esprit de l'appel par nom en ALGOL 60. Dans le cas de l'appel parallèle de la procédure, l'interprétation de l'appel par nom nécessite un échange de demandes et de réponses mettant en jeu le réseau, des protocoles de communication d'informations, un automate par site, une file d'attente par variable ou par paramètre passé par nom (voir le sous-chapitre B.4.3).

Les paramètres ne peuvent être de mode :

. adresse, car toute variable de mode adresse représente une adresse relative à la procédure dans laquelle elle se trouve;

. chaîne d'octets, car le mode chaîne d'octets n'est pas prévu pour les informations échangées entre deux procédures par paramétrage; ce mode est prévu pour typer les informations échangées par deux processus qui se communiquent directement sans partager une variable commune chez un ancêtre commun (voir le sous-chapitre B.4.2.);

. fils, car la variable de mode fils a pour rôle de contrôler le déroulement du processus fils et de contrôler l'accès aux variables et paramètres locaux par le processus fils (voir le sous-chapitre B.4.1.);

. liaison, car la variable de mode liaison représente le contexte de la liaison physique qui réalise la communication explicite entre la procédure où elle est déclarée et une autre procédure exécutée en parallèle (voir le sous-chapitre B.4.2.).

Donc pour les raisons énumérées plus haut, les variables qui ont pour mode l'un de ces quatre modes doivent rester locales à la procédure où elles sont déclarées; ni leur nom, ni leur valeur ne doivent être connus à l'extérieur de cette procédure.

2.4. Facilités pour la programmation séquentielle

Comme on l'a dit dans la description du vocabulaire du LI, une procédure LI possède une première partie qui est la description des variables locales, des paramètres et des procédures.

2.4.1. Variables de mode connu

* Logique : la valeur est 0 ou 1 définie sur un octet.

* Entier : la valeur est comprise entre -2^{15} et $2^{15}-1$ pour un entier court et -2^{31} et $2^{31}-1$ pour un entier.

* Adresse : la valeur est un entier compris entre 0 et $2^{15}-1$.

* Bloc de réveil : sa valeur est composée de quatre informations différentes :

  . durée de réveil,

  . type de réveil,

  . événement ou adresse de la procédure (standard ou non) du traitement à exécuter quand l'horloge de garde sonnera,

  . informations de référence ou paramètres effectifs pour cette procédure.

Un parallèle doit être fait avec le bloc de réveil de SYNCOP, support de multiprogrammation de SIGOR.

* Chaîne de caractères : la valeur est une suite de caractères qui appartiennent au code EBCDIC (ISO b). Il s'agit uniquement des caractères affichables; les caractères de contrôle et les codes non définis n'y sont pas compris. Cette variable possède un descripteur.

* Chaîne d'octets : la valeur est une suite finie d'octets dont chaque octet peut prendre une valeur de 0 à 255. Cette variable possède un descripteur.

Note :

. Les valeurs données ci-dessus telles que 2^{15}, 2^{31},

. les codes tels que le code EBCDIC ou le code IA5,

. l'utilisation de SYNCOP comme système de multiprogrammation de support,

sont bien sûr typiques à la mise en oeuvre de SIGOR sur l'IRIS80. Dans un contexte plus général, si le LI est pris comme langage intermédiaire de référence, ces valeurs ou ces références s'adapteront aux conditions de mise en oeuvre sur une machine x donnée. Cette adaptation est transparente à l'utilisateur qui écrit son programme en langage externe.

Cependant, ce genre de conventions est indispensable entre des machines qui communiquent sur un réseau selon des protocoles donnés (ELIE 75), (ZIMMERMANN 76).

2.4.2. Instructions arithmétiques et logiques

Ces instructions utilisent des opérateurs élémentaires bien connus qui réalisent l'addition, la soustraction, la multiplication, la division et les opérations logiques : ou, et, non. Les opérandes des instructions d'addition et de soustraction peuvent être des adresses.

2.4.3. Instructions de traitement des chaînes

Les instructions de définition de sous-chaîne et de concaténation opèrent sur des chaînes de caractères. Les instructions de mouvement et de calcul de longueur opèrent sur les chaînes de caractères ou les chaînes d'octets.

2.4.4. Instructions d'appel et de retour de procédure

On dispose des instructions d'appel séquentiel de procédure et de retour de procédure. Si l'on fait un parallèle avec PL1, on dispose l'équivalent des instructions CALL, END, RETURN de ce langage (IBM c).

Dans le cas des fonctions procédures, la présence de l'instruction de retour de la valeur de la procédure est obligatoire avant la fin de la procédure.

2.4.5. Autres instructions

Les instructions de comparaison (plus grand que, plus grand ou égal, ..) opèrent sur des entiers, des logiques, des adresses ou des chaînes de caractères. Elles positionnent un code condition à vrai ou faux.

L'instruction de rupture de séquence pour aller à une adresse relative de programme s'effectue si le code condition est vrai.

On dispose d'instructions de chargement (de valeur ou d'adresse) entre une cible et une source.

2.5. Facilités pour la programmation parallèle

Les possibilités exprimées ci-après sont nécessaires pour exprimer et organiser l'exécution parallèle de différentes procédures. Elles concernent essentiellement le lancement de processus, le contrôle des processus, le traitement du temps, le traitement de la synchronisation et le traitement de la communication explicite entre processus.

2.5.1. Lancement de processus

Un processus est crée quand on effectue un appel parallèle d'une procédure. Une hypothèse est faite sur le temps d'occupation de l'unité centrale par un processus : il est supérieur à 0 et inférieur ou égal à un temps maximum défini comme donnée de l'interpréteur; et ce, à chaque fois qu'un processus a la main. La raison de cette auto-limitation vient du fait que l'interpréteur est une ressource partageable par les processus; comme dans le système de multiprogrammation SYNCOP, les processus fonctionnent sans interruption (vu du côté de l'utilisateur); tout processus doit laisser le contrôle de l'unité centrale au bout d'un temps défini pour se mettre en attente conformément aux principes d'un processus téléinformatiques (DANG 1-77).

L'instruction d'appel parallèle de procédure n'est pas bloquante : à partir du moment où la procédure est lancée, le programme appelant se poursuit en séquence.

2.5.2. Contrôle de processus

On dispose des instructions qui permettent :

. de connaître l'état du processsus père ou fils,

. de tuer, de suspendre ou de réveiller le processus fils,

. de se tuer.

La fin du processus est la conséquence directe de l'exécution de l'instruction de fin de procédure correspondante. Dans le cas des fonctions procédures, la présence de l'instruction de retour de la valeur de la procédure est obligatoire avant la fin de la procédure, comme dans le cas de l'appel séquentiel.

Note :

Il est intéressant de souligner qu'en cas de terminaison anormale d'un processus, l'interpréteur :

. exécute la procédure de reprise d'erreurs éventuellement mise en place par l'utilisateur (cette procédure étant une procédure standard, cataloguée ou normalement définie),

. et de toute façon, réalise la fin de la procédure en informant le processus père de la terminaison anormale du processus fils; cela permet de déclencher chez le processus père le déroulement d'une éventuelle procédure de reprise d'erreurs.

Le processus père est toujours informé de la terminaison d'un processus fils :

. soit en recevant la valeur de la fonction procédure fils,

. soit en recevant la notification de la fin du processus fils, fin normale ou anormale.

2.5.3. Traitement du temps

Le temps est une variable de base dans les applications réparties. On dispose dans le LI de la variable de mode de base bloc de réveil et des instructions pour réaliser les instructions suivantes :

. attente par le processus d'une durée déterminée avant de reprendre l'exécution,

. exécution d'une procédure *en parallèle* avec le processus au bout d'un temps déterminé, quelque soit l'état du processus (actif ou en attente).

Ces deux actions consistent à :

. dans les deux cas, affecter à la variable de mode bloc de réveil une valeur de mode entier correspondant à la durée du réveil et

. à associer dans le premier cas à cette même variable un événement interne permettant au processus de se mettre en attente pour une durée déterminée; lorsque la durée prévue s'est écoulée, l'événement est posté et le processus rendu activable (cet événement peut être placé dans une liste pour attente multiple),

. à associer dans le deuxième cas à cette variable une procédure avec sa liste de paramètres effectifs; procédure qui sera exécutée en parallèle avec le processus d'origine et sous le contrôle d'un processus spécial du système de multiprogrammation support (DANG 2-76).

On dispose d'une instruction pour désarmer un réveil précédemment armé qui n'est pas arrivé à son terme.

2.5.4. Traitement de la synchronisation

La variable de mode événement est introduite pour réaliser la synchronisation entre processus. L'événement est un objet connu des deux processus à synchroniser, objet par l'intermédiaire duquel la synchronisation s'effectue.

L'événement considéré ici a une définition plus étendue que dans la notion classique d'événement. En plus de la notion d'attente multiple offerte à l'utilisateur, chaque événement est à mémorisation multiple (le processus pouvant attendre la réalisation de plusieurs occurences du même événement avant son déblocage).

Ce qui différencie la variable événement des autres variables est le fait qu'elle ne peut servir de cible à une affectation. Il n'y a que les opérations suivantes qui puissent être appliquées à la variable événement :

. l'attente simple sur un événement,

. l'attente multiple de n événements sur p,

. la vérification pour savoir lequel des p événements attendus est arrivé,

. la réalisation de l'événement.

Une stricte conformité aux principes définis dans SYNCOP est ici maintenue (DANG 1-77).

2.5.5. Traitement de la communication explicite entre processus

Pour réaliser la communication explicite entre processus, il existe des variables de mode liaison et rendez-vous et les opérations suivantes : réservation, libération, ouverture, fermeture, écriture et lecture.

Le sous-chapitre B.4.2.4. est réservé à cette communication explicite.

2.5.6. Catalogage de procédures

Toute procédure peut être cataloguée sur n'importe quel site possédant SIGOR. Une procédure cataloguée est identifiée par son nom (identificateur de la procédure en toutes lettres) et le nom du site de catalogage. Donc, dans une procédure, l'utilisateur ne peut cataloguer que les procédures affiliées dont les noms lui sont connus.

Dans les appels séquentiel et parallèle de procédure, on peut s'adresser à une procédure cataloguée.

L'instruction de mise hors catalogue réalise l'action inverse de l'instruction de catalogage.

2.6. Remarques

La présentation du langage LI est très condensé pour ne pas importuner le lecteur avec des détails techniques tels la représentation des variables, la liste des instructions, les principes d'interprétation. Il les trouvera dans l'annexe 1.1.

Par contre, dans cette présentation, la partie procédure-processus est largement développée dans la mesure où elle conditionne l'exposé du reste du chapitre.

3. Architecture de l'Interpréteur et du Sous-Système Interpréteur

3.1. Introduction

Ce travail est le dernier en date d'une suite de travaux réalisés par J. DU MASLE et son équipe :

. le compilateur ALGOL sur la machine Phillips PR 8000 (DU MASLE 66),

. la machine interprétative IGOR destinée à un langage de commandes réseau (FARZA 74).

Ces travaux sont proches du compilateur de RANDELL-RUSSEL (RANDELL 66) en ce qui concerne la méthode d'interprétation du langage intermédiaire.

Citons quelques travaux importants réalisés à l'Université de Grenoble dont nous nous sommes inspirés :

. l'étude d'un langage intermédiaire pour la compilation d'ALGOL 60 (LE PALMEC 66),

. le compilateur ALGOL sur IBM 1130 de LECARME et BELLISSANT (BELLISSANT 67).

Tous ces travaux, y compris SIGOR, dérivent de la méthode de DIJKSTRA exposée en 1961 sur les principes de la technique interprétative (DIJKSTRA 61).

Ce chapitre suppose que le lecteur est déjà familiarisé avec les principes des interpréteurs : notion de descripteur, partie statique, partie dynamique, .. (LE PALMEC 66). Un bref rappel est fait pour définir les termes et pour insérer les renvois vers l'annexe technique correspondante.

3.2. Structure des données de l'interpréteur sur un site X

Comme il est souligné dans le sous-chapitre B.2, la procédure LI forme la donnée principale de l'interpréteur. Les autres données sont :

. les paramètres effectifs de la procédure,

. les procédures standard sur le site X,

. les procédures cataloguées sur le site X.

En interprétant les instructions LI, l'interpréteur utilise des procédures de gestion, dites système, qui lui sont propres. La principale caractéristique de ces procédures est le fait qu'elles sont exécutables et non interprétables. Pour des raisons de facilité et de clarté dans la mise au point du système SIGOR, la procédure système est traitée par l'interpréteur comme toute autre procédure : c'est-à-dire que son appel correspond à un changement de niveau de nomenclature, sa partie dynamique est repérée comme une partie dynamique d'une procédure LI, etc ..

Voici quelques exemples de procédures système :

. procédures de gestion de changement de niveau de nomenclature (activation, calcul des paramètres effectifs, liaison avec le père si appel parallèle, ...) (DIJKSTRA 61) ;

. procédures de recherche de la valeur d'un paramètre passé par nom dont la localisation se trouve sur un autre site ;

. procédures de conversion de types, ...

De ce fait, par abus de langage, on dit qu'il y a quatre types de procédures traitées dans SIGOR : LI, standard, cataloguée et système.

3.2.1. Description d'une procédure

Comme on l'a vu dans la présentation du langage LI, *la procédure LI est l'unité de base destinée à un transport*. Elle est transportée sous la forme d'un descripteur (partie statique) qui comporte entre autres informations :

. une partie déclaration :
  - description des paramètres formels,
  - description des variables locales,
  - descripteur des procédures affiliées,

. une partie instruction :
  - instructions LI formant le corps de la procédure.

Note

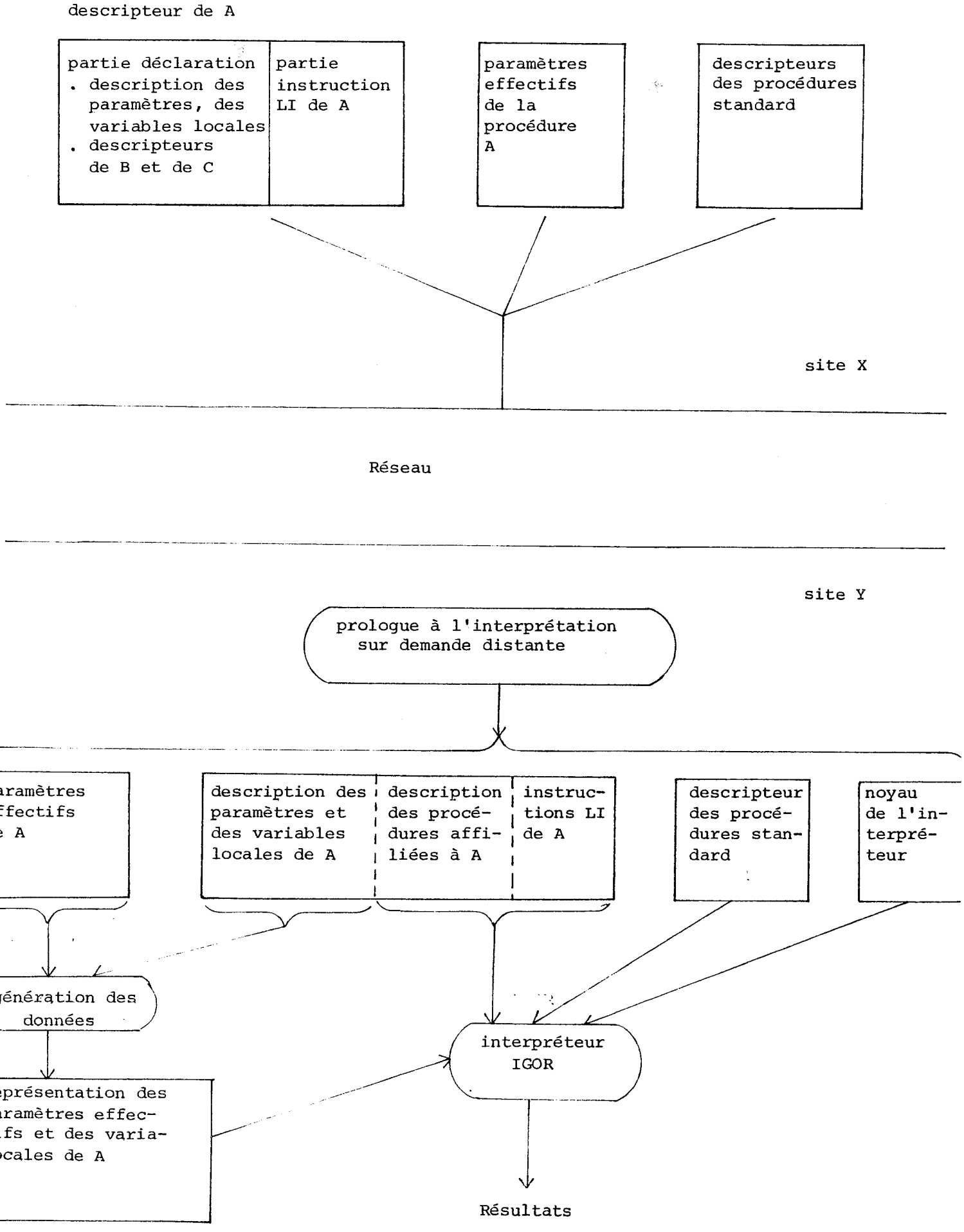
L'annexe 1.3 donne la définition détaillée de la description d'une procédure.

La description d'une variable locale par exemple, est différente de la représentation de la même variable locale : une description prend peu de place et nécessite une interprétation pour engendrer une représentation.

La figure suivante explique dans une forme plus détaillée que celle de la figure 2, le transport d'une procédure LI.

Soit un programme LI se déroulant sur un site X : dans ce programme, on veut envoyer une procédure A pour être exécutée sur un site Y. A possède B et C comme procédures affiliées. Il y a des procédures standard définies dans ce programme LI.

figure 4 - Transport d'une procédure



3.2.2. Représentation d'une procédure LI après la génération de données

Toute procédure LI, activée par un appel séquentiel ou par un appel parallèle, est représentée par deux parties :

. une partie statique décrite précédemment,

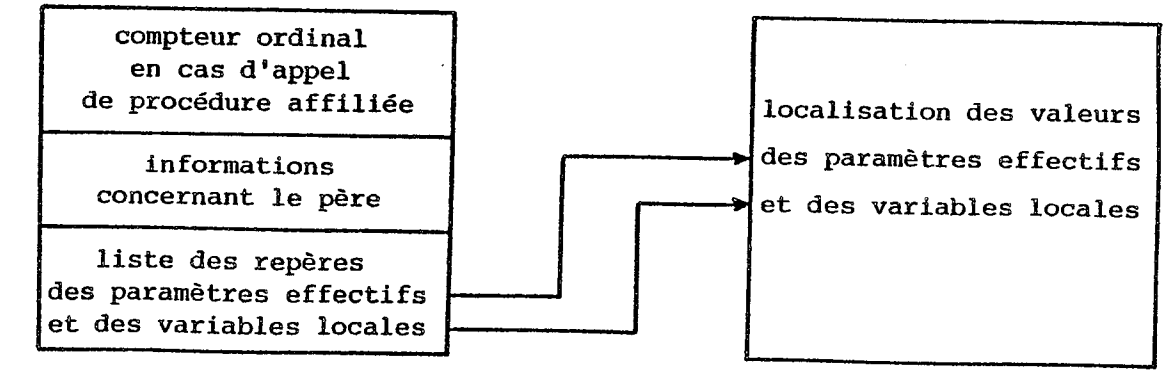
. et une partie dynamique, appelée dans SIGOR par la zone dynamique pour l'interprétation locale. Si on fait un parallèle avec PL/1, c'est l'équivalent de la DSA (Dynamic Storage Area) et des VDA (Variable Dynamic Area) (IBM b).

Cette partie dynamique contient entre autres les informations suivantes :

. vecteur qui sert de compteur ordinal quand il y a un appel à une procédure affiliée,

. liste des repères des paramètres effectifs et des variables locales ; les valeurs des paramètres effectifs, s'ils sont passés par valeur, et les valeurs des variables locales sont situées dans une zone dite zone de valeur pour l'interprétation locale ;

. informations concernant le père, si la procédure est appelée en parallèle. Ces informations concernent l'état du père et la liaison physique implicite entre père et fils (expliquée en détail au sous-chapitre B.4.).



Zone dynamique
pour l'interprétation locale

Zone de valeur
pour l'interprétation locale

Chaque variable locale ou chaque paramètre effectif passé par valeur, occupe un repère dans la zone dynamique pour l'interprétation locale. Ce repère comporte :

. un code qui donne le mode de la variable ou du paramètre,

. et une adresse qui pointe sur la valeur de la variable ou du paramètre, ou sur un descripteur dans le cas des chaînes de caractères et d'octets ; les valeurs et les descripteurs se trouvent dans la zone de valeur pour l'interprétation locale.

Le repère du paramètre passé par nom comporte, en plus du code donnant son mode, le repère du paramètre effectif chez l'appelant. Le traitement du paramétrage par nom est complexe et est abordé au sous-chapitre B.4.3.

Conséquence de la manière dont la partie statique d'une procédure LI est construite dans cette version de SIGOR, on trouve dans la partie dynamique les paramètres et les variables dans l'ordre suivant :

. les paramètres effectifs dans l'ordre de déclaration des paramètres formels,

. les variables selon l'ordre suivant : logique, entier, court, entier, événement, adresse, fils, rendez-vous, liaison, bloc de réveil, chaîne de caractères, chaîne d'octets.

Note :

Dans la partie statique de description de procédure, les adresses sont relatives ; dans la partie dynamique de représentation de procédure, les adresses sont absolues.

3.3. Noyau de l'interpréteur

3.3.1. Introduction

Pendant l'interprétation d'une procédure LI, la trace de l'instruction courante en cours d'interprétation, est gardée dans un compteur ordinal.

Le contexte d'un programme LI exécuté sur un site donné comporte trois parties importantes :

. le contexte du processus chargé d'exécuter ce programme ; les informations qui s'y trouvent sont conformes à ce qu'exige le support de multiprogrammation de SIGOR, c'est-à-dire SYNCOP (SEGUIN 76), (DANG 2-76);

. le vecteur d'état qui est constitué du compteur ordinal et d'une chaîne d'adresses dite chaîne statique (RANDELL 66) ; la structure de la chaîne d'adresses est fortement inspirée de celle de la machine Burroughs B6700 (BURROUGHS) ;

. les données nécessaires à l'interprétation, telles que :
- la pile d'interprétation,
- la mémoire de travail,
- la représentation du nom réseau du processus,
- la représentation des liaisons réseau gérées par le processus,
- ...

Un bref rappel est donné ci-après sur la façon dont le vecteur d'état d'un programme LI est construit (l'annexe 1.4 donne toutes les précisions voulues et des exemples), avant d'aborder la méthode d'accès réseau du processus.

3.3.2. Etablissement du noyau (cf. figure 4)

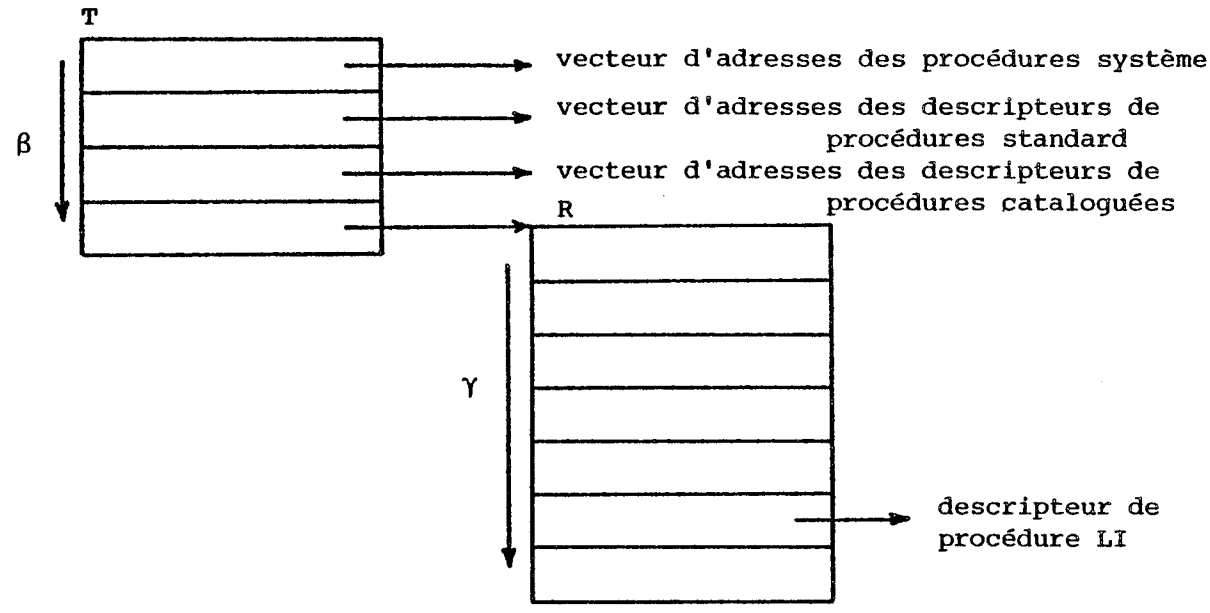
Quand une procédure LI de nom A est envoyée à partir d'un site Y sur un site X, un processus chargé de son exécution est créé et activé sur le site X. La première étape de l'interprétation de A n'est pas l'interprétation de la première instruction de A, mais correspond à l'établissement du vecteur d'état de A; cette étape est appelée, dans la figure 4, *prologue à l'interprétation sur demande distante* . Ce prologue n'est exécuté qu'une fois dans la vie du processus qui exécute A. Il a pour rôle :

a) d'établir le compteur ordinal ;

b) d'établir la chaîne statique qui comporte deux parties :

b.1) la première partie est un vecteur T d'adresses correspondant aux quatre types de procédures : système, standard, cataloguée et LI. Chaque adresse pointe vers un nouveau vecteur R d'adresses, dont la dimension est un paramètre de la génération du système SIGOR. Chaque nouvelle adresse pointe vers le descripteur d'une procédure d'un des quatre types indiqués ci-dessus.
Le repérage d'une procédure est défini par un couple d'entiers (β,γ), indices des deux vecteurs T et R (T pour type et R pour rang). Le schéma suivant bien connu permet de visualiser la première partie de la chaîne statique.

Note : Il faut remarquer que les procédures cataloguées sont propres au site et non au processus : de même les procédures système sont une ressource partageable par tous les processus présents sur le site Les procédures standard et cataloguées sont nommée en symbolique.



b.2) La deuxième partie est une pile S dont chaque élément pointe vers la zone dynamique pour l'interprétation locale d'une procédure ; l'empilement ou le dépilement se font à l'appel ou au retour d'une procédure. De ce fait, les zones dynamiques ne sont pas chaînées dynamiquement, car l'appelant se trouve empilé juste avant l'appelé.

Note :

Compte tenu de ce que nous venons de voir, le compteur ordinal est composé de :

. l'adresse relative de l'instruction dans la procédure,

. γ : indice de la procédure dans le vecteur d'adresse R,

. β : indice du vecteur R dans le vecteur T

. adresse de la zone dynamique de la procédure donnée par tête de pile de S.

Voir l'annexe 1.4.

3.4. Interface avec le réseau

3.4.1.

Si un site veut être apte à recevoir et interpréter du LI, il doit posséder les logiciels suivants :

. SYNCOP (DANG 1-77), qui sert de support de multiprogrammation pour SIGOR et qui permet de présenter SIGOR comme une tâche du système d'exploitation de l'ordinateur du site, dit système hôte (mis à part le cas de certains ordinateurs à vocation spécialisée, ou certaines machines virtuelles où les systèmes hôtes sont absents ; SYNCOP étant alors un système d'exploitation particulier *) ;

. une station de transport qui permet de communiquer sur le réseau (CYCLADES en l'occurence ; ce logiciel est réalisé à partir d'un protocole de transport (ELIE 75) ; de nombreuses mises en oeuvre sont actuellement en exploitation ; (voir l'annexe 2) ;

. le logiciel de SIGOR qui se compose de :
  - l'interpréteur IGOR
  - et le sous-système interpréteur qui gère les mécanismes d'aide à la répartition et la communication inter-processus ;

. le macro-générateur de langage LM si le site veut produire du LI et l'interpréter (SERGEANT 78).

Note :

On pourrait réaliser SIGOR sur des sites possédant un système (ou sous-système) de multiprogrammation et une station de transport (ou un logiciel équivalent) : le modèle reste le même avec une mise en oeuvre différente.

---

* Citons le cas du mini-ordinateur ORDOPROCESSEUR où SYNCOP est défini comme système d'exploitation (HANDLE 76) et le cas de CP/67-IBM 360 (DANG 2-76).

3.4.2. Le processus CONCIERGE

C'est un processus transparent à l'utilisateur et permanent sur tout site possédant SIGOR. Il est toujours à l'écoute du réseau et de tout processus créé sur ce site.

Son rôle est, d'une part, de recevoir d'un processus concierge distant une procédure LI, les paramètres effectifs et les procédures standard. Selon un protocole qui sera développé au § 4.1.2. du chapitre B, le processus concierge va :

. créer le processus qui va interpréter la procédure LI,

. créer une liaison entre lui-même et ce processus.

D'autre part, il est en attente multiple de consommation sur les liaisons ouvertes entre lui et les processus créés sur le même site. Si l'un de ceux-ci veut faire transporter une procédure vers un site distant, il s'adresse au concierge qui s'acquittera du transport de procédure préparé par le processus.

Implicitement, il est entendu qu'un processus concierge :

. possède un nom sur le réseau (identification d'une porte au sens du protocole de bout en bout de CYCLADES (ELIE 75)),

. possède un répertoire lui permettant de faire la correspondance entre un nom de site et le nom du concierge de ce site,

. possède une liaison avec chacun des autres concierges (flot au sens du protocole de bout en bout).

3.4.3. Nom réseau d'un processus utilisateur

Tout processus, créé sur un site pour interpréter une procédure LI, a un nom réseau : c'est l'identification d'une porte selon le protocole de bout en bout de transport.

Ce nom est acquis par le processus concierge à la création du processus. Il est connu par le processus lui-même et par le processus père (voir le protocole décrit au § 4.1.2. du chapitre B). Il sert essentiellement à :

. établir des liaisons filiales

. et à donner une référence unique au processus sur le réseau.

3.4.4. Interface avec la station de transport

L'accès à la station de transport s'effectue par le moyen d'un interface qui offre les services suivants :

. ouverture de porte,

. ouverture de flot,

. fermeture de porte,

. fermeture de flot,

. envoi de lettre,

. envoi de télégramme,

. réception de lettre,

. réception de télégramme.

Note :

Cet accès est transparent à l'utilisateur de SIGOR. Il est destiné aux procédures système qui réalisent les communications implicites entre processus père et fils et explicites entre deux processus quelconques.

Chaque porte identifiant un processus sur le réseau, représente un moyen d'accès système au processus ; la méthode d'accès est donnée par l'interface avec la station de transport : accès en lecture/écriture (réception et envoi de lettre et de télégramme), ou accès uniquement en lecture (réception de lettre ou de télégramme) sur le réseau.

L'ensemble des portes identifiant des processus SIGOR est défini ; son cardinal est fixé de manière statique et ses éléments sont prédéfinis à la génération de SIGOR sur le site.

Remarques :

La station de transport gère les liaisons (ou flots), les portes et les communications sur flot ou porte ; c'est un outil d'aide à l'exploitation qui fournit à l'utilisateur des renseignements sur son correspondant :

. vivant ou mort,

. actif ou inactif,

. la liaison est fermée et la raison (fermeture du correspondant, coupure réseau).

Du fait de la station de transport, on peut répondre simplement à la question suivante :

Q. Comment un père se rend-il compte de la disparition sans avertissement du fils ?

R. La station de transport lui aura signalé l'absence du fils à l'autre bout de la liaison.

4. Mécanismes d'Aide à la Répartition

4.1. Transport de programmes

Le premier objectif visé par SIGOR est le suivant : transport de programmes. Ce transport, qui a pour objet une procédure LI, nécessite les trois fonctions suivantes :

- création dynamique de processus à distance,
- contrôle du transport d'un algorithme,
- contrôle de l'exécution distribuée.

Chacune de ces fonctions met en jeu le processus origine du transport, dit *processus père*, le processus concierge du site origine et le processus concierge du site distant qui créera le processus interpréteur de la procédure transportée; ce processus s'appelle le *processus fils*.

La coopération entre ces processus est régie par un protocole, dit *protocole implicite de lancement de processus à distance*.

4.1.1. Appel_parallèle_d'une_procédure

L'instruction d'appel parallèle de procédure met en oeuvre les paramètres suivants :

activer (nom de la procédure, liste des paramètres effectifs)
sur le site (nom du site)
sous le contrôle de (nom d'une variable de mode fils)

1) la procédure appelée parallèlement doit être affiliée à la procédure où se trouve l'instruction activer.

2) la liste des paramètres effectifs se présente sous la forme d'une suite de couples (t,e) :

. t donne le type du paramètre :

- par nom,
- par valeur différée,
- par valeur ;

. r donne l'emplacement de paramètre effectif ou sa valeur (dans ce cas, le type du paramètre doit être par valeur).

3) le nom du site d'interprétation de la procédure appelée peut être donné par :

. soit une chaîne de caractères de longueur inférieure ou égale à 8,

. soit le contenu d'une variable de mode chaîne de caractères

. soit le résultat d'une fonction procédure appelée séquentiellement et à valeur chaîne de caractères.

Cette fonction procédure a pour but de ménager aux utilisateurs une possibilité de détermination dynamique du site sur lequel ils souhaitent voir interpréter la procédure appelée. Ce type de paramétrage du site interpréteur peut servir par exemple à rendre transparent le réseau : c'est une procédure système qui cherche le site disponible au moindre coût; l'appel d'une telle procédure système est le rôle de la génération de code LI comprise dans un compilateur de langage de commandes réseau (DU MASLE 74).

Exemple :

. chercher un site qui possède un compilateur FORTRAN-H,

. chercher un site qui possède un système d'exploitation acceptant l'exécution d'un programme PL1 avec 500K octets disponibles.

Par ailleurs, le site nommé ou trouvé peut être identique au site du processus qui interprète l'instruction d'activation. Dans ce cas, le processus fils fonctionnera en parallèle avec le processus père sur le même site ; le transport de programme a lieu localement, mais les principes de base du mécanisme de transport restent les mêmes à quelques optimisations près.

Donc, dans le traitement du lancement de processus, la notion de distant ou de local est rendue transparente.

4) Le contrôle de l'exécution en parallèle du processus fils est effectué par une variable de mode fils. Cette variable contrôle, d'autre part, l'accès à partir du processus fils aux variables du père passées par nom comme paramètres effectifs.

Cette variable ne doit pas être déjà utilisée par un autre processus fils vivant, au moment de l'interprétation de l'instruction activer.

Dans la représentation de la variable de mode fils, on trouve entre autres :

a) l'état du processus fils :
   - en attente d'être activé,
   - en activité,
   - mort pour cause de cessation d'activité,
     de coupure réseau,

b) le nom de la procédure transportée,

c) le site d'interprétation de la procédure fils,

d) l'adresse d'un bloc qui contient toutes les informations nécessaires à l'utilisation de la liaison implicite réseau père-fils ; liaison dont le protocole exposé ci-après va montrer l'établissement. Ce bloc possède un symétrique du côté du processus fils .

e) une liste de couples de valeurs indiquant la correspondance entre le repère de toute variable du père passée comme paramètre par nom et le rang de la même variable dans la liste des paramètres effectifs.

Dans le sens du processus fils vers le processus père, on peut contrôler la cohérence de la demande d'accès vers une variable du père. Dans le sens inverse, pour retourner une valeur au processus fils, cette correspondance permet d'adresser la valeur de retour par la déduction : repère -> rang. (rappel : les paramètres effectifs sont représentés en tête dans la zone dynamique de toute procédure ; tout objet, variable ou paramètre, est représenté par un repère qui pointe sur la valeur de l'objet)

La variable de mode fils permet, non seulement de contrôler l'accès aux variables du père et aux paramètres du fils, mais de contrôler aussi l'exécution en parallèle du processus fils ; et ce, par l'intermédiaire des instructions suivantes :

. demande pour connaître l'état du fils,

. tuer le fils,

. suspendre ou réveiller le fils.

Note :

* La fin de la procédure appelante n'implique pas forcément ni la mort du père ni la mort du fils :

. la fin de la procédure i où se trouve l'appel parallèle d'une procédure affiliée (qui est devenue le processus fils) ne signifie pas la mort du processus père dans la mesure où i a peut-être été appelée séquentiellement par une procédure j englobante ; seulement la fin de i implique la disparition de la variable de mode fils qui contrôle le processus fils en cours d'exécution ; la fin de l'environnement d'appel implique aussi la disparition des variables locales ou des paramètres de i ;

la libération de la variable de mode fils entraîne la coupure de la liaison implicite père-fils ; ainsi le processus fils sera-t-il averti de la fin de la procédure appelante ; supposons qu'à partir de ce moment, le processus fils a besoin d'accéder aux variables passées par nom de la procédure appelante, il constatera que la liaison implicite est fermée et par conséquent l'accès est impossible et il se suicidera; dans l'hypothèse inverse, le processus fils continue à s'exécuter normalement ;

. même si la fin de la procédure appelante signifie la mort du processus père, cela n'implique pas forcément pour les mêmes raisons la mort du processus fils.

* L'interprétation de l'instruction activer nécessite une négociation et des échanges d'informations régis par un protocole et de ce fait met le processus père en attente du succès du transport de la procédure à activer sur un site distant.

Dans l'annexe 1.4., on trouve un exemple de l'état du noyau de l'interpréteur du processus père quand il laisse ainsi le contrôle de l'unité centrale.

4.1.2. Création dynamique de processus à distance

Soit sur un site X un processus père qui veut appeler une procédure A en parallèle sur un site Y.

1) Hypothèses de départ

. Il existe sur le site X un processus concierge appelé CONCIERGE_X par exemple. Sur le site Y, soit CONCIERGE_Y le processus concierge.

. On suppose $X \neq Y$.

. Il existe une liaison réseau entre CONCIERGE_X et CONCIERGE_Y, dite LIAISON_XY. Cette liaison réseau qui existe à la génération du système SIGOR, se matérialise par un flot fonctionnant en contrôle de flux sur le réseau CYCLADES par exemple.

. Il existe une liaison locale entre le processus père et le processus CONCIERGE_X. Cette liaison locale existe depuis le moment où le processus père a été créé.

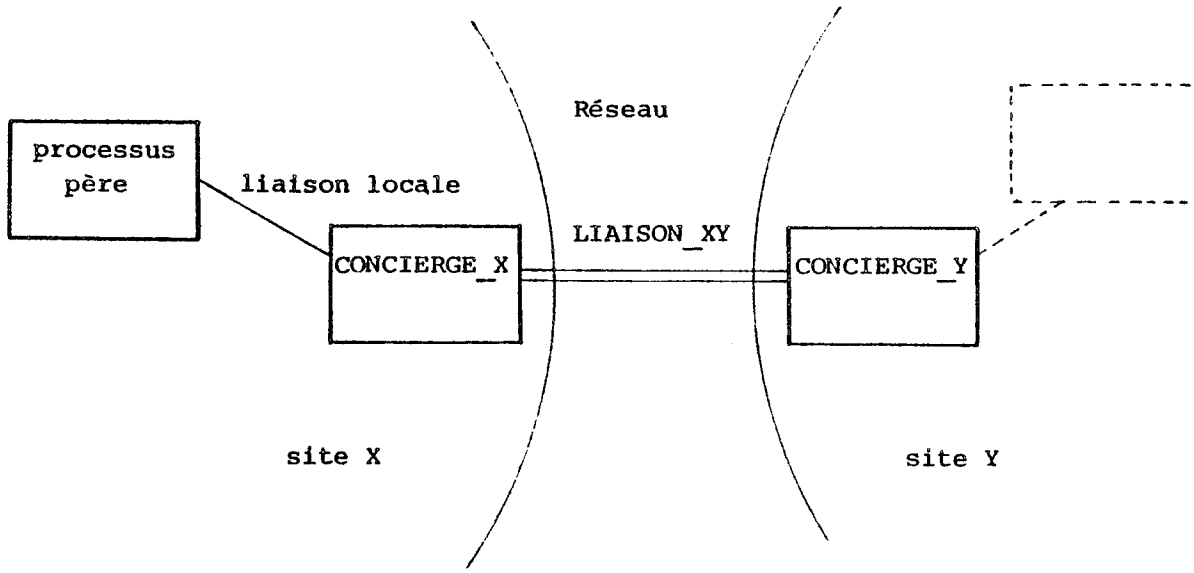
2) Protocole implicite de lancement de processus à distance

Le protocole décrit ici ne donne pas les détails techniques sur les commandes utilisées et sur la manière de les utiliser. Ils se trouvent dans l'annexe 1.5.

Note :

Entre deux processus, l'échange d'informations soumis à ce protocole est réalisé par des *commandes*. Les commandes sont exprimées dans ce sous-chapitre avec une notation fonctionnelle : fonction suivie de ses arguments.

La figure suivante permet de bien se fixer les idées sur les hypothèses de départ :



2.a) Le processus père produit une commande :

négociation (nom réseau du processus père,
site distant demandé,
longueur des informations à envoyer)

dans la liaison locale père-CONCIERGE_X. Le processus père se mettra en attente d'une demande d'ouverture de liaison réseau provenant du site distant demandé, en l'occurence Y. Cette attente est couverte par une horloge de garde.

La réception et le traitement de la commande de négociation par le processus CONCIERGE_Y entraîne l'envoi d'une commande :

création (nom réseau du processus père,
longueur des informations à envoyer)

sur la liaison réseau LIAISON_XY à destination du processus concierge distant CONCIERGE_Y.

Le nom du site du processus père est implicitement connu par le processus CONCIERGE_Y à cause de l'utilisation de la liaison LIAISON_XY pour l'envoi de la commande de création.

2.b) Le processus CONCIERGE_Y, à la réception de la commande de création va :

. créer un processus*, appelé processus fils,

. établir une liaison locale entre lui-même et le processus fils, de même type que la liaison père-CONCIERGE_X,

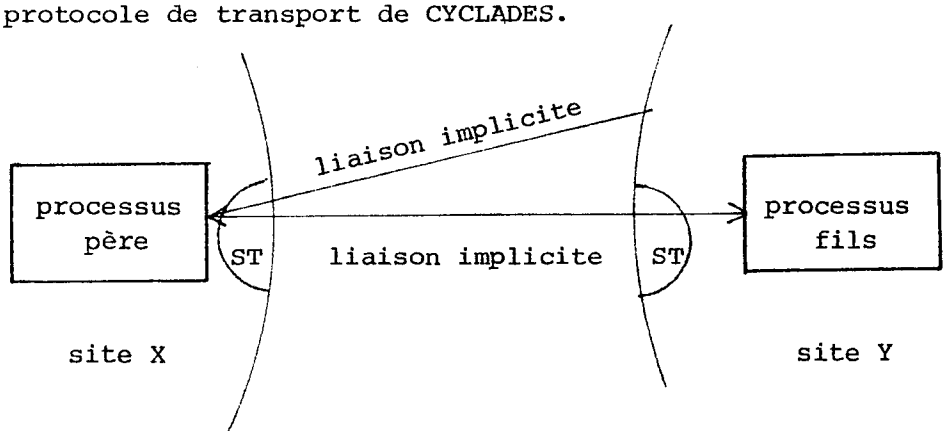
. donner un nom réseau au processus fils et

. communiquer au processus fils le nom réseau du processus père et la taille des informations que le père va envoyer.

Note :

Le nom réseau d'un processus utilisateur de SIGOR est expliqué en B.3.4.3. ; il correspond à l'identification d'une porte du réseau.

2.c) Le processus fils demande à la station de transport locale de son site l'ouverture d'une liaison réseau avec le processus père dont il connaît le nom réseau. Cette méthode dissymétrique d'ouverture de liaison (l'initiative est confiée aux fils) est celle suggérée par le protocole de transport de CYCLADES.



---

* ceci sous-entend que le support de multiprogrammation offre la possibilité de création dynamique de processus.

La figure ci-dessus représente la liaison implicite entre le père et le fils après la négociation nécessaire à son établissement.

Cette liaison s'appelle la *liaison implicite* entre un père et un fils. Un père peut avoir plusieurs liaisons implicites avec différents fils ; le fils n'en a qu'une avec son père.

Note :

Si le site Y est confondu avec le site X, la liaison implicite reste une liaison réseau.

2.d) Envoi de la procédure

L'envoi de la procédure A destinée à être interprétée par le processus fils s'effectue en trois étapes :

. envoi du descripteur de procédure de A :

Pour faciliter l'analyse chez le processus fils, les informations complémentaires sont envoyées avant le descripteur de la procédure A :

- (n+1), n étant le nombre de procédures affiliées à la procédure A, n pouvant être égal à 0,
- un vecteur de dimension n dont chaque élément de rang i donne l'adresse relative de la $i^{ème}$ procédure affiliée à A (si n vaut 0, ce vecteur n'est pas envoyé),

et le descripteur de la procédure A.

Note :

Le vecteur qui est envoyé avant le descripteur de la procédure n'est pas strictement indispensable dans la mesure où l'on peut le reconstituer à partir du descripteur de A. Mais il faut remarquer que ce vecteur s'obtient facilement à partir de la première partie de la chaîne statique (§ B.3.3.2) chez le père. De ce fait, il vaut mieux le constituer chez le processus père.

. envoi des paramètres effectifs de A :

Cet envoi est nécessaire si et seulement si la procédure envoyée A est déclarée avec des paramètres. Il se compose des informations suivantes :

- p, nombre de paramètres effectifs,
- un vecteur de dimension p dont l'élément de rang i donne :
  + le mode du $i^{ème}$ paramètre,
  + le type de ce paramètre : par valeur (différée ou immédiate) ou par nom,
  + une valeur indiquant soit le repère du paramètre effectif dans la procédure appelante s'il n'est pas passé par valeur immédiate, soit une adresse relative dans le champ défini ci-après,
- une zone des valeurs des paramètres effectifs passés par valeur immédiate.

. envoi des procédures standard :

Cet envoi, s'il y a des procédures standard à envoyer, est composé des informations suivantes :

- s, nombre des procédures standard,
- un vecteur de dimension (s-1) dont l'élément de rang i repère le descripteur de la $(i+1)^{ème}$ procédure standard dans la zone suivante,
- une zone où se trouvent les descripteurs des procédures standard.

2.e) Le processus fils produit une commande

<u>réponse à l'envoi de la procédure</u> (raison)

qui signale au processus père la fin du transport de procédure ou l'existence d'une anomalie dans le déroulement du protocole.

Note :

On a utilisé dans ce protocole le terme de liaison, de préférence au terme flot du protocole de bout en bout de CYCLADES. Ce choix provient du fait de la terminologie utilisée dans les protocoles de transport au niveau européen (SCHICKER 76) et à un niveau au-dessus (INWG 78). Mais ici, la liaison est un flot de CYCLADES avec un contrôle d'erreur et de flux (ELIE 75) : la station de transport du côté du processus père s'assure que les commandes sont envoyées et acquittées par la station de transport du fils.

Les commandes seront découpées en fragments et envoyées fragment par fragment. La taille exacte prise par chaque commande est une donnée que le récepteur, le processus fils, connaît à la réception de la commande. En ce qui concerne la taille des informations échangées sur la liaison implicite, donnée qui conditionne l'allocation de tampons de réception par le fils, elle a été négociée à l'ouverture par le père.

Le protocole implicite de lancement de processus à distance et les mécanismes mis en jeu pour le gérer sont transparents à l'utilisateur.

4.1.3. Contrôles

Il y a deux types de contrôles : le premier est lié au transport de procédure et le deuxième est lié à l'exécution de la procédure sur le site distant.

1) Contrôle du transport

Il est prévu des contrôles de la bonne marche du transport :

. Quand le processus père attend la demande d'ouverture de liaison de la part du fils, l'attente est couverte par une horloge de garde de délai assez long. Quand le réveil correspondant sonnera, le processus père considèrera que le fils ne peut assurer la demande d'ouverture et que le lancement de processus à distance a échoué.

. Le processus fils n'abandonnera la négociation de l'ouverture de la liaison qu'au bout de n tentatives infructueuses. Il se suicidera après avoir fermé la liaison locale avec le concierge.

. Une fois la liaison établie, s'il y a une erreur non récupérable dans le transport, les stations de transport le signaleront aux processus père et fils ; le processus père considèrera que le transport a échoué. L'action chez le fils est la même s'il y avait une anomalie constatée en analysant une commande : le fils doit fermer la liaison réseau père-fils après avoir informé le père de cette fermeture ; le fils se suicidera après avoir fermé la laison locale avec le concierge.

L'analyse d'une commande peut provoquer la constatation d'une anomalie, par exemple l'incohérence d'une information (désaccord sur un type de paramètre, ..).

Ce contrôle fait partie du protocole implicite de lancement de processus à distance ; il est placé ici dans le but de regrouper les problèmes de contrôle.

2) Contrôle de l'exécution distribuée

Une fois que le processus fils a reçu la procédure à interpréter et que le noyau de l'interpréteur est installé, le processus fils va commencer l'interprétation au cours de laquelle :

. il peut avoir des échanges de données avec le père, dits échanges typés et présentés ci-après ;

. par ailleurs, le père et le fils sont constamment tenus au courant de leur état respectif dans la mesure où les stations de transport savent surveiller la liaison réseau père-fils. Les instructions LI de demande d'état permettent au père comme au fils de connaître l'état respectif du parent. Les instructions LI telles que tuer, suspendre et réveiller permettent au père d'agir sur le fils.

Une extension possible des possibilités de contrôle consiste à prévoir dans le LI l'instruction *sur condition de* :

a) sur condition de (tel type)
appel séquentiel (nom d'une procédure standard,
liste des paramètres effectifs)

Les conditions qui déclenchent l'appel aux procédures standard de reprise peuvent être par exemple :

- mort du père pour telle raison ou
- mort du fils pour telle raison.

b) sur condition de (tel type)
activer (nom d'une procédure affiliée,
liste des paramètres effectifs)
sur le site (nom du site)
sous le contrôle de (nom d'une variable de mode fils)

4.2. Communication d'informations entre processus

4.2.1. Introduction

On peut regrouper en quatre classes les échanges d'informations entre processus :

a) La première concerne les échanges d'informations relatives à l'état d'un processus :

- . demande pour connaître l'état du processus père ou fils,
- . réponse à cette demande,
- . demande pour tuer un processus fils,
- . demande pour suspendre un processus fils,
- . demande pour réveiller un processus fils.

Ces échanges s'effectuent par la liaison implicite entre le père et le fils.

b) La deuxième classe comprend l'envoi du fils au père de la valeur de retour si la procédure du fils est une fonction et l'envoi au père de l'information concernant la mort naturelle du fils (sur fin de la procédure du fils).

c) La troisième classe contient les lectures et les modifications par le fils des variables locales du père via les paramètres. Dans la mesure où les commandes échangées concernent les variables de mode connu et constituent une méthode d'accès parfaitement définie à la représentation de ces variables et de ces paramètres, cette classe est appelée : *communication d'informations typées entre processus*

d) La quatrième classe contient les échanges d'informations réalisés entre deux processus qui partagent un même ancêtre (ou père). Pour des raisons de commodité, deux processus peuvent avoir besoin de réaliser entre eux un transfert explicite d'informations telles que lettre, paquet, tableau de données, fichier, .. Ces informations se trouvent sous forme de chaîne d'octets. Cette quatrième classe est appelée *communication explicite d'informations entre processus* (DANG 2 et 5-77).

Note :

La communication explicite d'informations peut bien sûr avoir lieu entre le processus père et le processus fils.

Dans la suite de l'exposé, on ne va aborder que les problèmes posés par les deux dernières classes d'échanges d'informations. Les Les deux premières classes se réalisent à l'aide des commandes dont la liste est donnée dans l'annexe 1.5.

4.2.2. Exemple sur les communications implicite et explicite entre processus

L'exemple présenté ci-après est informel dans la mesure où les langages et les objets, autres que le LI et la machine SIGOR, utilisés dans l'exemple sont supposés existants : le langage externe de requêtes proche de PLi utilisé pour décrire l'exemple, son compilateur, les bases de données, ..

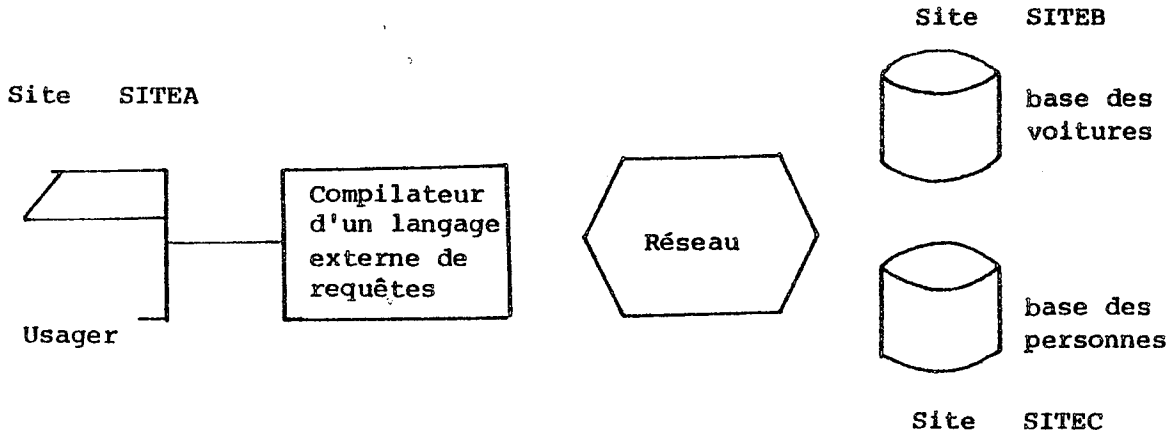
Cet exemple a pour but de :

. sensibiliser le lecteur aux problèmes de communication entre processus,

. servir d'introduction, de renvoi possible et d'aide à la lecture des paragraphes B.4.3. et B.4.4. et

. montrer les possibilités potentielles du système SIGOR et du langage transportable et interprétable LI.

Supposons la configuration réseau suivante :



Supposons qu'un usager effectue la requête suivante :

nom de toutes les personnes possédant une voiture rouge et habitant Grenoble ?

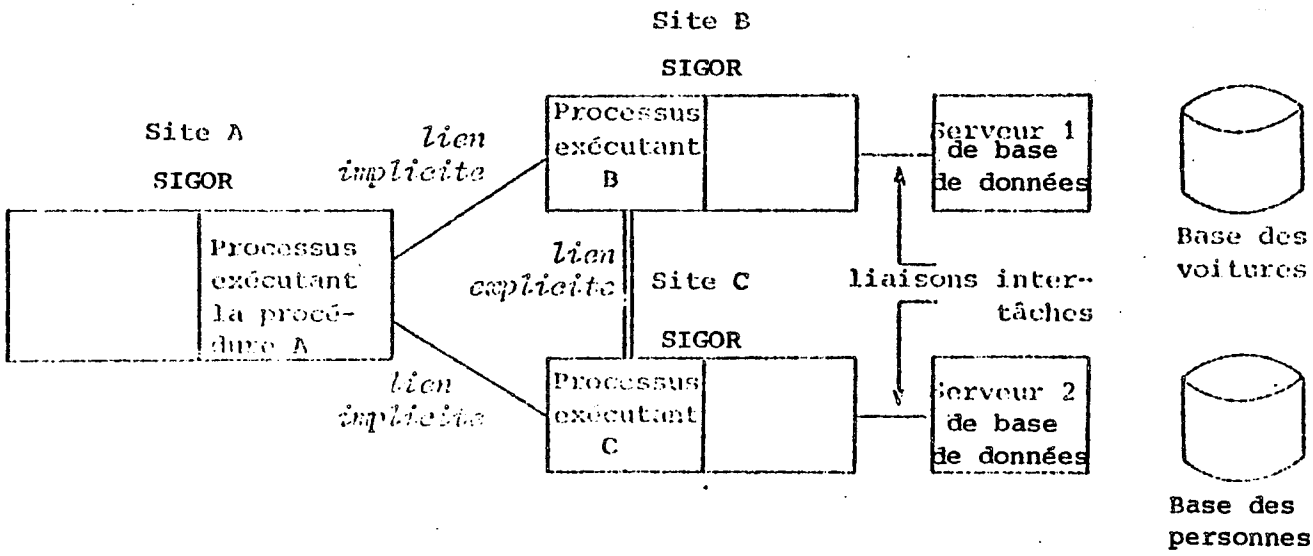
Cette requête définit deux critères :

. voiture de couleur rouge s'appliquant à la base des voitures,
. personne habitant Grenoble s'appliquant à la base des personnes.

L'intérêt de SIGOR est de pouvoir définir une procédure représentant chacun de ces critères et de demander l'exécution de ces deux procédures, B et C par exemple, sur les deux sites concernés. Une solution consiste à exécuter indépendamment ces deux procédures en récupérant les résultats sur le site interrogateur où s'effectuerait la comparaison. On transférerait de B vers A les noms de toutes les personnes possédant une voiture rouge et de C vers A les noms de toutes les personnes habitant Grenoble. L'établissement d'une liaison entre deux procédures B et C permet une optimisation importante des échanges. La procédure B recherche les personnes possédant une voiture rouge, transmet leur identification à la procédure C qui ne répercute vers A que les noms des personnes habitant Grenoble. Une optimisation peut être recherchée dans le choix de la procédure origine : procédure B cherchant les voitures rouges ou procédure C recherchant les personnes habitant Grenoble. Un critère de choix peut être la taille des bases et le nombre des accès nécessaires.

Pour cet exemple, on suppose qu'il y a moins de voitures rouges que de personnes habitant Grenoble.

Après le lancement des deux procédures B et C, on a la structure de fonctionnement suivante :



Le programme suivant réalise la requête.

Note :

Le programme est écrit en pseudo-PL1 (structure, procédure, paramètres, instructions, ..) sauf pour quelques notions empruntées à ALGOL (définition du type de passage de paramètres, mode entier).

Les instructions et les objets nouveaux, du fait du langage intermédiaire, sont commentés. Une relecture de cet exemple après la lecture de B.4.2.3. et B.4.2.4. permettra de préciser quelques points pouvant paraître obscurs.

```
A : PROCEDURE(N) ;
    DCL N INTEGER BY NAME ;              paramètre passé par nom
    DCL RV RENDEZ-VOUS ;                 variable de mode rendez-vous
    DCL (FILS1,FILS2) SON ;              variables de mode fils pour contrôler B et C
    DCL TAB(N) CHAR(20) ;                variable de mode chaîne de caractères
    DCL EVENTA EVENT ;                   variable de mode événement
    B : PROCEDURE (PARB) ;               B procédure affiliée à A
        DCL PARB RENDEZ-VOUS BY NAME ;   paramètre de mode rendez-vous passé par nom
        DCL I INTEGER ;
        DCL (COL,NAME) CHAR(20) ;
        DCL PONTB LIAISON ;              variable de mode liaison réseau
        RESERVE (PONTB) VIA PARB ;       négociation pour l'ouverture de la liaison
        OPEN (PONTB) ;                   explicite entre FILS1 et FILS2
        DO I = 1 TO N ;
            CALLEXT FINDCOLOUR (I,COL,NAME) ; appel de la fonction externe FINDCOLOUR
                                             du serveur base de données n° 1 qui consulte
                                             la base des voitures et détermine la couleur
                                             de la voiture (selon COL) et le nom de son
                                             propriétaire (mis dans NAME)
            IF COL = 'RED' THEN
                WRITE (PONTB) NAME ;         écrire sur la liaison réseau PONTB le contenu
                                             de NAME
        END DO ;
        WRITE (PONTB) 'STOP' ;               écrire sur PONTB la chaîne de caractères 'STOP'
    END B ;

    C : PROCEDURE (PARC,PARTAB,EVENTC) ; C procédure affiliée à A
        DCL PARC RENDEZ-VOUS BY NAME ;   paramètre de mode rendez-vous passé par nom
        DCL PARTAB (*) CHAR(20) BY NAME ;
        DCL EVENTC EVENT BY NAME ;
        DCL J INTEGER ; DCL (RESULT,AD) CHAR(20) ;
        DCL PONTC LIAISON ;              variable de mode liaison réseau
        RESERVE (PONTC) via PARC ;       négociation pour l'ouverture de la liaison
        OPEN (PONTC) ;                   explicite entre FILS2 et FILS1
```

```
        J = 1 ;
        READ (PONTC) RESULT ;          lire les informations qui arrivent de la
                                       liaison réseau PONTC et les mettre dans RESULT;
                                       PONTC est le nom de la liaison FILS2-FILS1
                                       vue du côté de FILS2
        WHILE (RESULT ≠ 'STOP') DO ;
            CALLEXT FINDADDRESS (RESULT,AD) appel de la fonction externe FINDADDRESS
                                       du serveur base de données n° 2 qui consulte
                                       la base des personnes et détermine l'adresse
                                       de la personne dont le nom est dans RESULT;
                                       l'adresse est rendue dans AD
            IF AD = 'GRENOBLE' THEN DO ;
                                    PARTAB(J) = RESULT ;
                                    J = J + 1 ;
                                    END DO ;
        END WHILE ;
        POST (EVENTC) ;                poster l'événement EVENTC
        CLOSE (PONTC) ;                fermer la liaison réseau FILS2-FILS1
    END C ;

    ACTIVE B(RV) SITE(SITEB) SON(FILS1) ; appel en parallèle de la procédure B sur
                                       le site SITEB avec comme variable de contrôle
                                       de processus fils : FILS1
    ACTIVE C(RV,TAB,EVENTA) SITE(SITEC) SON(FILS2) ; appel en parallèle de la procé-
                                       dure C sur le site SITEC avec comme variable
                                       de contrôle de processus fils : FILS2

    WAIT (EVENTA) ;                    attente de l'événement EVENTA posté
    .
    .
    .
    .

END A ;
```

4.2.3. Communication d'informations typées entre processus

Il est sous-entendu que les processus dont on parle ici sont liés par la relation de filiation.

1) Hypothèses et définitions

Pour faciliter la lecture du paragraphe 2, on va prendre l'exemple suivant :

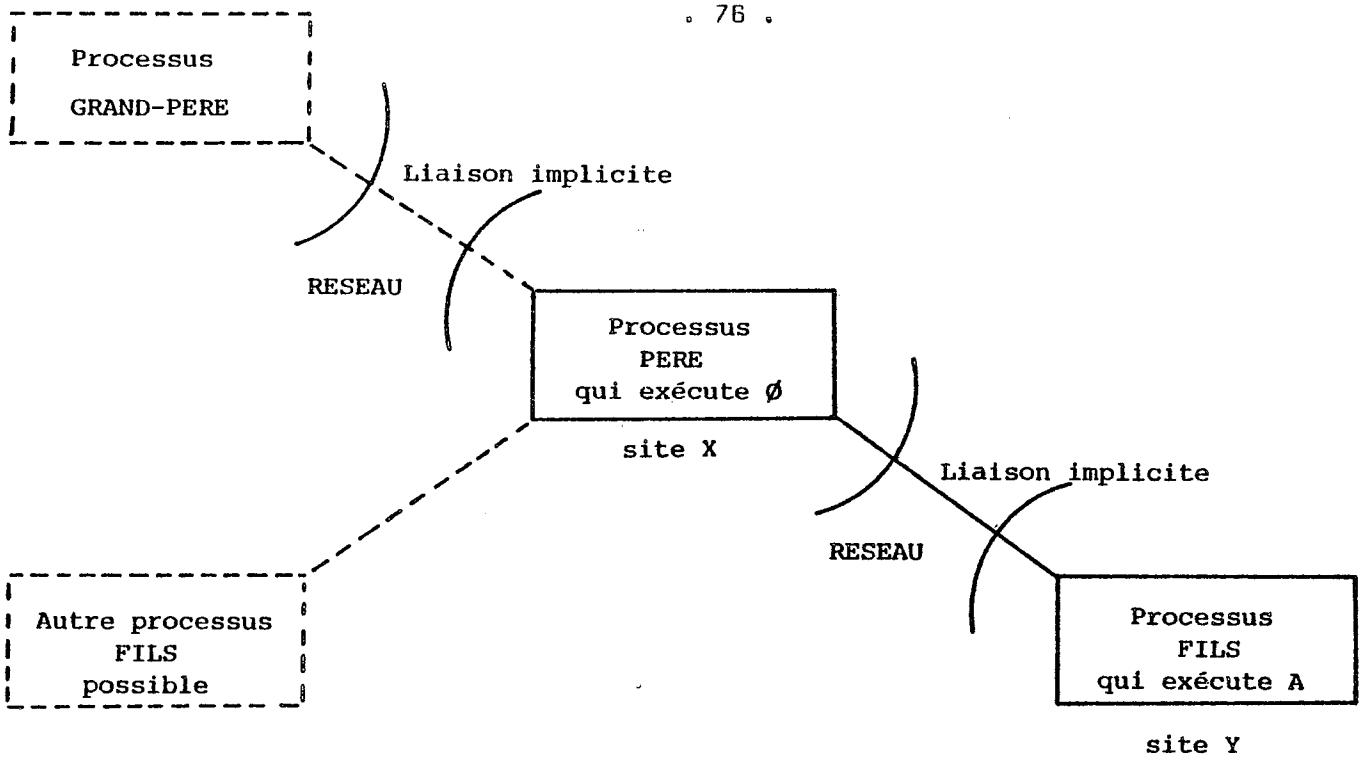
```
Ø : PROCEDURE ;
    /* déclaration de ARG1, ARG2, ARG3, FILS1 comme variables locales */
    A : PROCEDURE (PV1, PV2, PN) ;
        /* PV1, PV2 sont passés par valeur, PN est passé par nom */
        PN = O ;
        .
        .
        .
        ECRIRE (PN) ;
    END A ;
    ACTIVE A (ARG1, 5, ARG3) SITE (Y) SON (FILS1) ;
    CALL A (3, 4, ARG2) ;
END Ø ;
```

. La procédure A est appelée en parallèle avec la procédure Ø à laquelle A est affiliée ;

. A est définie avec trois paramètres formels, les deux premiers par valeur et le troisième par nom ;

. A est exécutée sur le site Y en parallèle avec l'exécution de Ø sur le site X supposé différent de Y ;

. après l'appel en parallèle de A, le processus père interprète en séquence l'appel séquentiel de la procédure A sur son site, en l'occurence X ; les deux appels de A sont indépendants l'un de l'autre.



Dans la suite de ce paragraphe, on entendra par :

. *environnement d'interprétation d'une procédure* : le noyau de l'interpréteur (compteur ordinal, chaîne statique, données de travail), les descripteurs des procédures (cataloguées, standard, LI), les zones dynamiques des procédures en cours d'interprétation ;

. *paramètre formel* : paramètre formel passé par nom d'une procédure et dont le paramètre effectif correspondant peut ne pas se trouver dans le même environnement d'interprétation qu'elle ; exemple : PN ;

. *paramètre effectif réseau* : variable locale ou paramètre formel par nom d'une procédure, ayant été passé comme paramètre effectif à une procédure qui ne s'interprète pas dans l'environnement d'interprétation de la procédure appelante ; exemple : ARG3. A la fin de la procédure appelée, tout paramètre effectif réseau qui n'est impliqué que par cet appel n'est plus visible du réseau mais uniquement dans la procédure où il est déclaré ;

. *paramètre effectif local* : variable locale d'une procédure ayant été passée comme paramètre effectif à une procédure appelée séquentiellement ; exemple : ARG2. Toute variable locale peut être à la fois paramètre effectif réseau et paramètre effectif local.

2) Exposé des problèmes à résoudre

Dans une procédure, un paramètre formel passé par nom peut être utilisé de deux manières :

. soit comme cible d'une affectation ; dans ce cas, l'interprétation doit affecter une expression à la cible, donc au paramètre effectif (réseau ou local) ;

exemple : PN = 0 ; dans la procédure A

. soit dans une expression ; dans ce cas, l'interprétation doit connaître la valeur du paramètre effectif (réseau ou local) ;

exemple : ECRIRE (PN) ; dans la procédure A.

Dans le premier cas, l'interpréteur local veut écrire dans le paramètre effectif qui peut être distant sur le réseau ; il réalise cette écriture avec une commande dite *commande E* d'écriture dans un paramètre effectif. Si le paramètre effectif se trouve sur un autre site, la commande E est envoyée sur la liaison implicite ; sinon elle est traitée localement. Ses arguments sont : E (repère du paramètre effectif, valeur à affecter).

Dans le deuxième cas, l'interpréteur local veut connaître la valeur du paramètre effectif (réseau ou local) ; il réalise cette lecture en adressant d'abord une commande, dite *commande L*, de lecture de la valeur d'un paramètre effectif à la procédure où le paramètre effectif se trouve :

L (repère du paramètre effectif)

et en attendant ensuite de celle-ci une réponse, dite *commande R* :

R (repère du paramètre formel, valeur retournée).

Si le paramètre effectif se trouve sur un autre site, les commandes L et R sont envoyées sur la liaison implicite de filiation.

Rappel :

Dans le cas d'une programmation séquentielle, tel qu'en ALGOL 60 par exemple, ces commandes sont en réalité réalisées par deux procédures, l'une allant chercher la valeur dans l'environnement dynamique de la procédure appelante et l'autre allant écrire dans cet environnement dynamique.

Dans le domaine réseau, l'environnement dynamique de la procédure appelante appartient au processus père. Le processus fils doit demander une coopération du processus père pour pouvoir y accéder soit en lecture, soit en écriture.

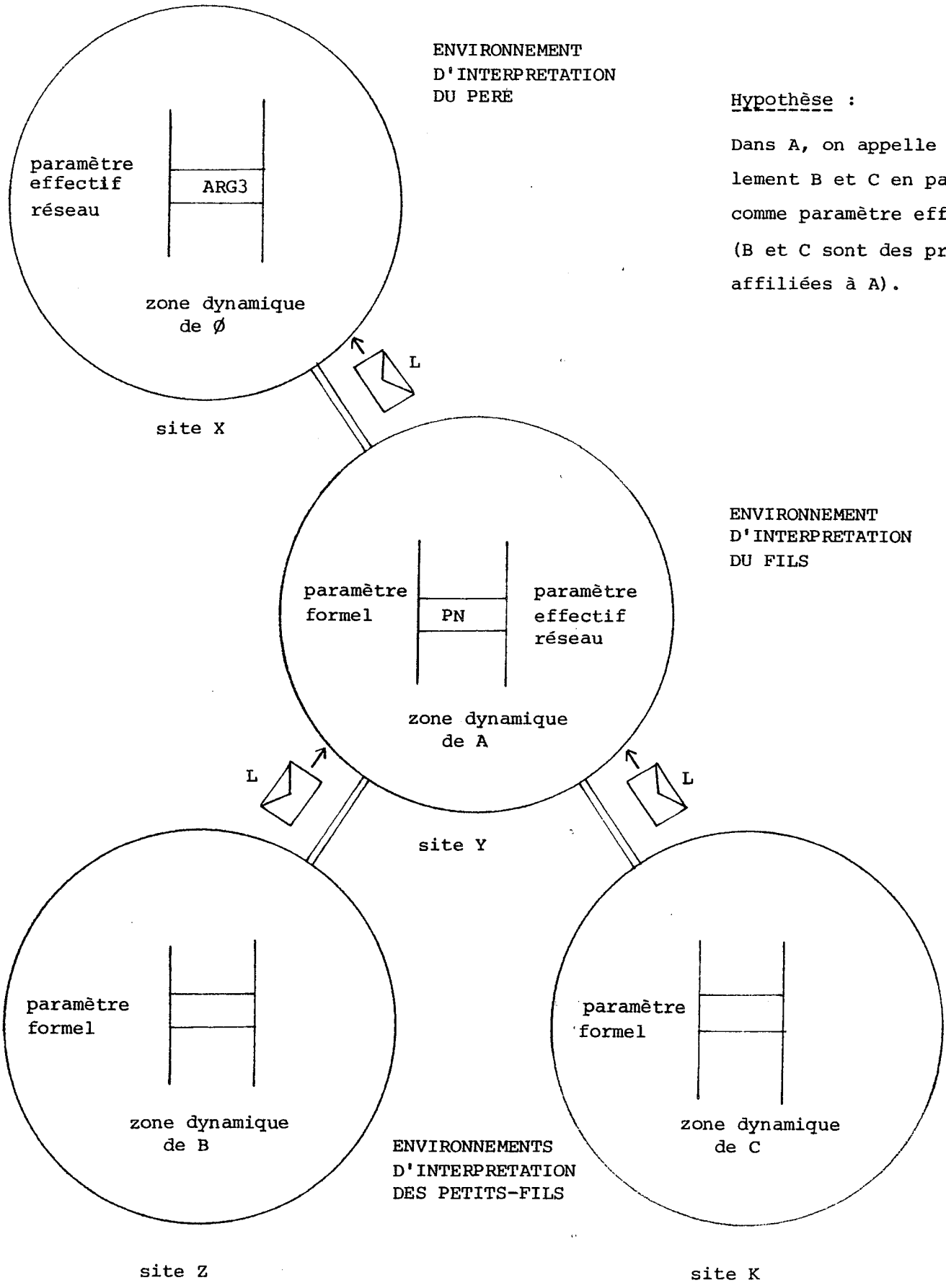
La gestion de cette coopération repose sur une entente appelée, dans SIGOR, *le protocole implicite d'échange d'informations entre père et fils.*

Un certain nombre de problèmes sont à résoudre :

. Une variable locale peut être paramètre effectif réseau simultanément pour plusieurs processus fils, tout en étant paramètre effectif local dans une procédure du père, sachant que le programme du père et ceux des fils sont interprétés en parallèle. Sachant que l'on ne peut établir une stricte hiérarchie dans le temps des commandes de lecture ou d'écriture, car le réseau peut introduire des retards dans la transmission, comment résoudre le problème des conflits entre les commandes et comment les satisfaire ?

. Tout paramètre effectif réseau peut être paramètre formel par nom (dans la mesure où l'on veut passer en paramètre effectif au petit-fils un paramètre que l'on tient du père) ; donc le problème ne se trouve plus circonscrit entre deux sites, ceux du père et du fils, mais se trouve généralisé sur le réseau entre n sites. Chaque site a émis une commande E, ou L, ou R, vers un autre site. Comment gérer cet ensemble de commandes à un instant donné, sachant que l'on ne possède pas de graphe complet décrivant les liaisons implicites existantes sur le réseau ?

figure 5



Hypothèse :

Dans A, on appelle parallèlement B et C en passant PN comme paramètre effectif (B et C sont des procédures affiliées à A).

3) Protocole implicite d'échange d'informations entre père et fils

Dans ce paragraphe, on utilise le terme *demande* pour exprimer l'échange entre les deux extrêmités qui sont :

. le processus fils, le *demandeur*, où le paramètre formel passé par nom est utilisé,

. le processus ancêtre, le *donneur*, où le paramètre effectif correspondant est représenté.

Ces deux processus ne sont pas forcément liés par une seule liaison implicite. Une demande est constituée de une ou plusieurs commandes échangées entre un ou des couples de processus père-fils.

Donc, toute demande de lecture ou d'écriture s'exprime sous la forme d'une suite non vide et finie de commandes E, L ou R.

On note : $d^s_{i,j}$ une demande de type s entre deux extrémités i et j

$c^s_{i,k}$ une commande de type s entre les processus i et k

on obtient :

$$d^s_{i,j} = (c^s_{i,i+1}, c^s_{i+1,i+2}, \ldots, c^s_{j-1,j}) \quad i \leq j - 1$$

$$s = E, L \text{ ou } R$$

Rappel :

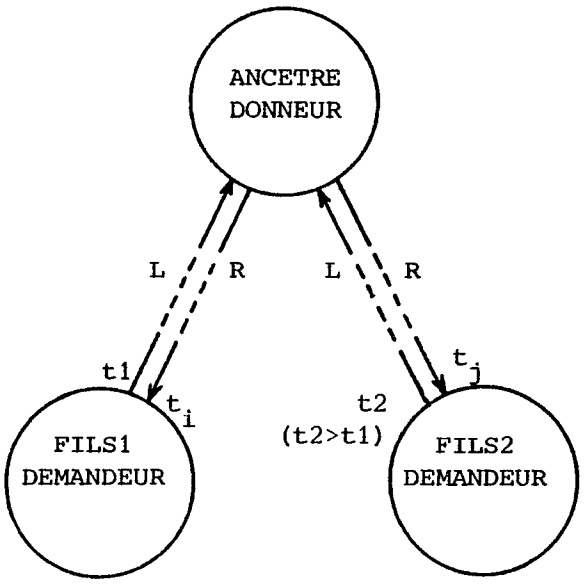
Entre le processus fils demandeur et le processus ancêtre donneur, il existe un et un seul chemin constitué par une ou des liaisons implicites et par n processus, avec $n \geq 0$ (ces processus se présentent comme des noeuds d'un arbre de filiation).

On fait l'hypothèse suivante pour faciliter l'exposé de ce paragraphe : tout appel de procédure envisagé ici est un appel parallèle, donc les actions envisagées sur le réseau sont asynchrones.

Comme il n'est pas possible d'établir une chronologie dans le temps des commandes de provenances différentes concourant à un même type d'actions, on admet les règles suivantes dans la lecture de ce protocole :

Règle 1 :

La première commande arrivée sera la première traitée par le processus receveur.



Soient deux processus fils voulant accéder en lecture à une variable locale de l'ancêtre donneur passée en paramètre par nom :

- à l'instant t1, le premier processus fils envoie une commande L à son père,
- à l'instant t2, le deuxième processus fils envoie une commande L à son père ;

on suppose t2 > t1 ;

- la deuxième commande peut arriver chez l'ancêtre avant la première du fait du réseau et c'est la deuxième qui sera d'abord traitée par l'ancêtre ;
- t_i et t_j sont les instants où les réponses R respectives arrivent chez les deux fils. Il n'y a pas de relation d'ordre possible entre t_i et t_j.

Règle 2 :

Le processus qui reçoit et traite une commande de lecture peut se trouver dans trois états :

a) incapable de satisfaire la commande (la valeur de l'objet désigné par la commande ne se trouve pas dans son environnement) ;

b) susceptible de satisfaire la commande (le processus vient d'envoyer une commande au processus père pour lire l'objet demandé) ;

c) capable de satisfaire la commande (la valeur de l'objet désigné par la commande se trouve dans son environnement d'interprétation).

Selon son état, le processus procède à l'action respective suivante :

. s'il est à l'état a, il envoie une commande de lecture vers son père, mémorise la commande et passe à l'état b ;

. s'il est à l'état b, il ne change pas d'état et mémorise la commande ;

. s'il est à l'état c, il envoie une commande de réponse vers le processus fils qui lui a envoyé la commande de lecture ; dans ce cas, le processus est appelé l'ancêtre donneur.

Règle 3 :

Les commandes de lecture du protocole sont échangées entre deux processus et non entre un couple de demandeur-donneur ; ces commandes ne mémorisent pas le chemin par où elles passent pour réaliser le routage des commandes de réponse au retour. Ce sont les processus sur le chemin qui mémorisent au fur et à mesure les informations (nom du fils, adresse de l'objet qui était à l'origine de la commande dans l'environnement du fils), nécessaires pour adresser les commandes de réponse au retour.

Les règles 2 et 3 ont pour conséquence l'obligation pour *tout paramètre réseau* d'avoir *une file d'attente* dont chaque élément :

. correspond à une commande de lecture,

. et contient les informations citées ci-dessus nécessaires à l'adressage de la commande de réponse correspondante.

Règle 4 :

Une demande d'écriture entre un processus fils demandeur et un processus ancêtre donneur (où se situe la variable locale destinée à être modifiée), provoque autant de commandes d'écriture qu'il y a de liaisons implicites existantes sur le chemin reliant le demandeur au donneur.

Règle 5 :

Si l'état d'un objet réseau est b, c'est-à-dire susceptible de satisfaire une commande de lecture et si sa file d'attente est non vide, l'arrivée d'une commande d'écriture pour modifier sa valeur, provoque en plus du départ d'une commande d'écriture, les actions suivantes :

. changement d'état, le nouvel état est a (incapable de satisfaire une commande de lecture),

. défilement de la file d'attente et envoi d'autant de commandes de réponse qu'il y a d'éléments dans cette file d'attente, avec comme valeur, la valeur apportée par la commande d'écriture.

Si l'état d'un objet réseau est a, c'est-à-dire incapable de satisfaire une commande de lecture, l'arrivée d'une commande d'écriture provoque le départ d'une commande d'écriture vers le père.

Si cet état est c, c'est-à-dire capable de satisfaire une commande de lecture, le processus propriétaire de l'objet est le processus ancêtre donneur ; l'arrivée d'une commande d'écriture provoque l'affectation de cet objet avec la valeur apportée par la commande.

Règle 6 :

L'arrivée d'une commande de réponse, s'adressant à un objet réseau donné, provoque les actions suivantes :

. défilement de la file d'attente de l'objet et envoi d'autant de commandes de réponse qu'il y a d'éléments dans cette file d'attente ;

. changement d'état ; l'objet auquel s'adresse une commande de réponse ne peut être que dans l'état b ; le nouvel état est a, c'est-à-dire incapable de satisfaire une commande de lecture.

Remarques :

1) Ces règles expliquent comment doit fonctionner un automate attaché à chaque objet réseau de SIGOR :

. les changements d'état selon l'arrivée des commandes,

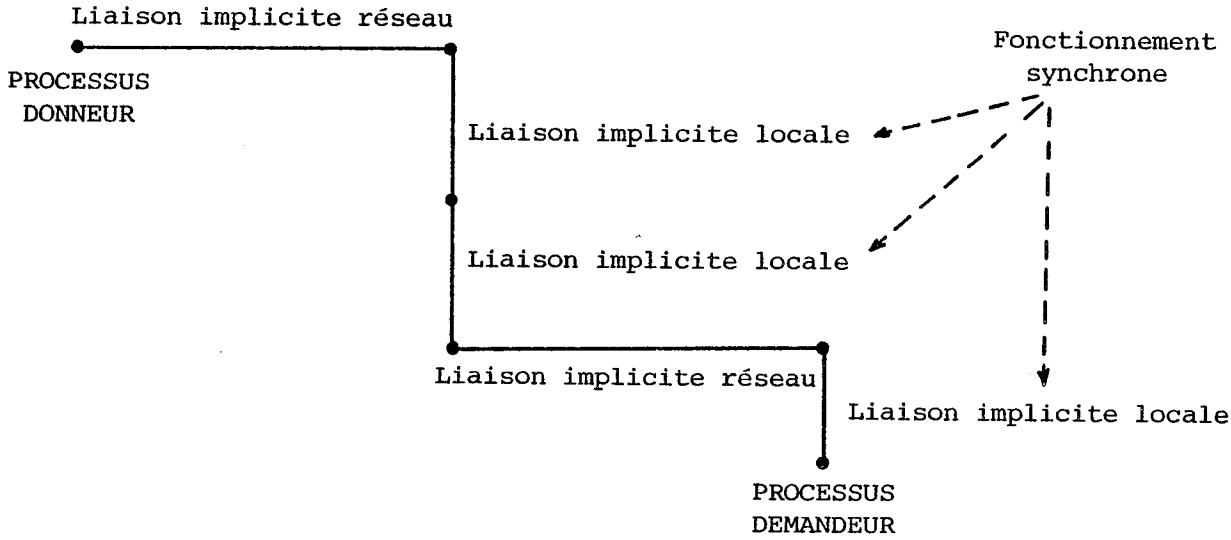
. et les actions à entreprendre correspondantes.

Un tel automate, compte tenu des problèmes de propagation de pannes sur le réseau, est étudié dans (MARTINS).

2) En réalité, dans une application répartie, les appels séquentiels et les appels parallèles de procédures peuvent se succéder dans un ordre quelconque ; le chemin défini entre un ancêtre donneur et un fils demandeur, est constitué par des liaisons implicites réseau et par des liaisons implicites locales (réalisées par logiciel) impliquées par l'ordre des appels.

La liaison implicite réseau est consacrée à des activités asynchrones, alors que la liaison implicite locale fonctionne d'une manière synchrone.

Illustration d'un chemin possible entre un donneur et un demandeur :



Pour résoudre ce problème, il existe des pseudo-commandes de lecture, d'écriture et de réponse : LS, ES et RS, qui sont sémantiquement équivalentes aux commandes L, E et R citées dans les règles lorsque l'appel de procédure est synchrone.

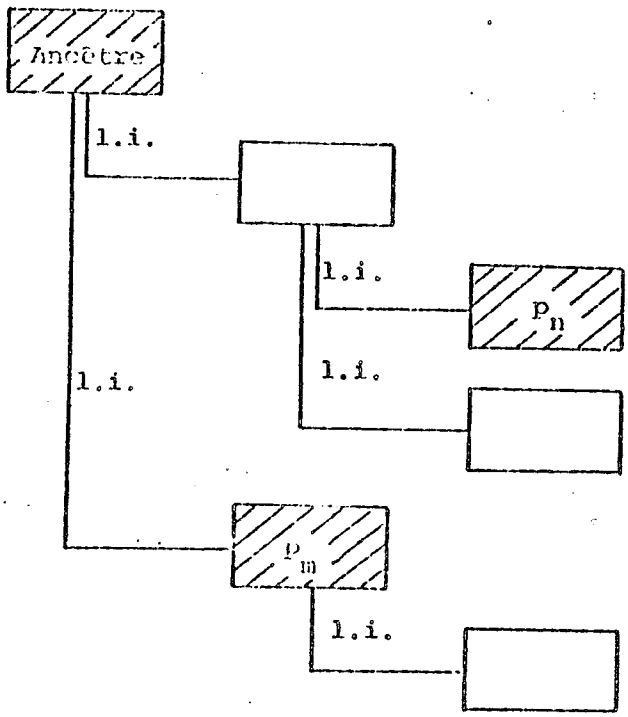
L'automate et les files d'attente prennent en compte les pseudo-commandes LS, ES et RS (voir l'annexe 1.1.5.).

3) Les détails techniques sur la structure des commandes du protocole sont développés dans l'annexe 1.1.5.

4.2.4. Communication explicite d'informations entre processus

Quand deux processus ont besoin de communiquer des informations directement (voir l'introduction § 4.2.1) sans passer par le partage d'une variable commune, ils utilisent, dans SIGOR, la communication explicite * (DANG 2 et 5-77).

L'approche faite par SIGOR du problème de communication inter-processus repose sur l'existence d'un père ou d'un ancêtre commun entre deux processus quelconques d'une application répartie.



l.i. = liaison implicite

Considérons deux processus p_n et p_m qui veulent communiquer. Le chemin qui relie p_n à p_m se compose de deux liaisons implicites fils-père et d'une liaison implicite père-fils ; donc ils peuvent communiquer via l'ancêtre par l'intermédiaire de paramètres.

---

* Dans le document de travail (FERRIE 77), l'utilisation des termes *implicite* et *explicite* pour qualifier une désignation d'objets est très proche de l'utilisation qui est faite ici de ces mêmes termes.

Ce mode de communication est lourd, d'autant que certaines informations de taille et de nature diverses ne peuvent s'exprimer facilement avec les types de paramètres de procédures. Aussi est-il nécessaire d'établir une liaison explicite entre p_n et p_m et de leur permettre de l'utiliser en lecture et écriture.

Etant donné que SIGOR possède un interface avec la station de transport de chaque site (ou un logiciel équivalent), une solution au problème posé plus haut consiste à offrir à l'utilisateur de SIGOR le même interface que celui du système avec la station de transport. Cette solution condamne l'utilisateur à :

. connaître les problèmes spécifiques et élémentaires du transport d'informations sur un réseau,

. utiliser un interface qui est fonction du logiciel de transport du site.

Dans le but d'offrir un outil de haut niveau d'aide à l'écriture des applications réparties, SIGOR propose l'approche suivante à ce problème :

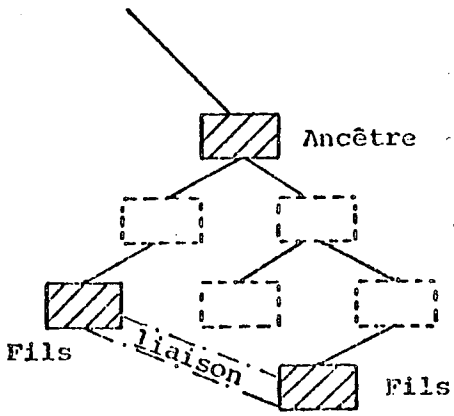
. établissement d'une liaison explicite entre les deux processus intéressés, grâce à une variable d'un nouveau mode, RENDEZVOUS, qui se trouve dans l'environnement du processus ancêtre commun ;

. utilisation de cette liaison avec les fonctions lire et écrire, comme si l'utilisateur opérait avec un fichier en accès direct. La liaison sera vue par chaque algorithme intéressé comme une variable d'un mode donné, LIAISON ;

. possibilités standard ou laissées à l'utilisateur pour définir des procédures de compactage et de cryptage de l'information transportée sur la liaison [RAZEK 76], [KAHN].

Le contrôle de la communication s'effectue à partir de l'ancêtre où se situe la variable de mode rendez-vous et à partir des deux processus concernés.

L'établissement et l'exploitation de la liaison nécessitent un protocole dit *protocole explicite d'échange d'informations*.



Conventions :

Dans le reste de ce paragraphe, les processus qui partagent un *ancêtre* commun et qui veulent communiquer, sont appelés *fils*.

1) Variable RENDEZ-VOUS

La variable de mode rendez-vous se trouve dans l'environnement de l'ancêtre. Elle est passée en paramètre par nom aux processus fils intéressés. Elle est donc partagée en lecture et écriture par les procédures fils.

Quand un fils veut communiquer avec un autre fils, c'est-à-dire négocier l'établissement d'une liaison réseau, il utilise la communication implicite entre lui et son ancêtre pour indiquer ses coordonnées et son nom à l'intention de son futur correspondant.

Dans la variable de mode rendez-vous, se trouvent les informations suivantes :

- nom du fils,
- état de la liaison,
- mot de passe qui sert à contrôler la négociation d'ouverture de la liaison.

2) Variable LIAISON

La liaison établie entre deux fils est vue par chacun sous la forme d'une variable de mode liaison. Elle se trouve dans l'environnement d'interprétation du fils. (Rappel : la variable de mode LIAISON n'est pas utilisable comme paramètre de procédure).

Dans la variable de mode liaison se trouvent les informations suivantes :

- l'état de la liaison,
- un événement utilisé pour surveiller l'activité de la liaison,
- l'adresse d'un bloc qui contient toutes les informations nécessaires à l'utilisation de la liaison réseau (établie sous forme d'un flot en contrôle de flux sur le réseau CYCLADES),
- le nom du paramètre de mode rendez-vous qui permet d'accéder en lecture et écriture à la variable de mode rendez-vous dans l'environnement de l'ancêtre,
- les noms des procédures de reprise d'erreur,
- les noms des procédures système ou utilisateur de compactage et décompactage, de cryptage ou de décryptage des informations transférées sur la liaison.

Note :

Bien que les procédures de compactage ou de cryptage ne soient pas mises en oeuvre dans un premier temps, ce point est très important dans un contexte hétérogène : si l'on peut se mettre d'accord sur une description commune unique de l'information sur le réseau, une procédure de traduction peut être associée à une liaison ; elle convertit automatiquement toute description locale en description réseau et réciproquement.

3) Instructions LI

Toute tentative d'ouverture d'une liaison avec un correspondant, se décompose en deux étapes successives :

. réservation de la variable de mode rendez-vous et désignation de la variable de mode liaison que l'on veut utiliser,

. ouverture de la liaison.

La raison de cette distinction tient du fait que l'on souhaite que l'utilisation d'une liaison soit précédée par un rendez-vous et que la séparation :

. de la réservation (qui marque l'intention de se servir d'un moyen de communication explicite et qui permet au processus de se nommer),

. et de l'ouverture de la liaison (qui indique le début du travail sur la communication),

permette de s'assurer de la disponibilité d'un objet avant son utilisation.

Les instructions correspondantes sont :

réserver (nom de la variable de mode liaison)
via (nom du paramètre de mode rendez-vous)

ouvrir (nom de la variable de mode liaison, mot de passe)

Les instructions symétriques sont :

fermer (nom de la variable de mode liaison)

libérer (nom de la variable de mode liaison)

Une fois que la liaison est établie, les processus fils peuvent travailler sur la liaison en lecture et écriture en utilisant une variable de mode chaîne d'octets :

lire (nom de la liaison, nom de la variable chaîne d'octets
qui reçoit l'information)

écrire (nom de la liaison, nom de la variable chaîne d'octets
qui contient l'information à écrire)

Note :

La réservation d'une variable de mode rendez-vous n'est abandonnée que quand la libération est demandée. Cela signifie qu'un fils peut :

- . réserver un rendez-vous avec un autre fils,
- . ouvrir une liaison avec lui avec un certain mot de passe,
- . fermer cette liaison,
- . ouvrir une autre liaison avec un autre fils, avec un autre mot de passe, pourvu que ce dernier ait réservé le même rendez-vous que lui.

4) Protocole explicite d'échange d'informations

4.a) La réservation est réalisée par une demande qui se compose d'autant de commandes réservation qu'il y a de liaisons implicites sur le chemin reliant le fils au père :

réservation (nom du paramètre effectif de mode rendez-vous,
nom réseau du processus fils)

Les paramètres formels et effectifs de mode rendez-vous sur le chemin fils-père conservent la mémoire du passage de la commande de réservation, n'en autorisent pas d'autres et savent adresser au retour la commande :

réponse à la réservation (raison)

Le processus fils obtient ainsi la raison du refus ou de l'acceptation de la réservation. Si le chemin fils-père est en état de marche, la seule raison d'un refus est que la variable de mode rendez-vous est déjà réservée pour un autre processus.

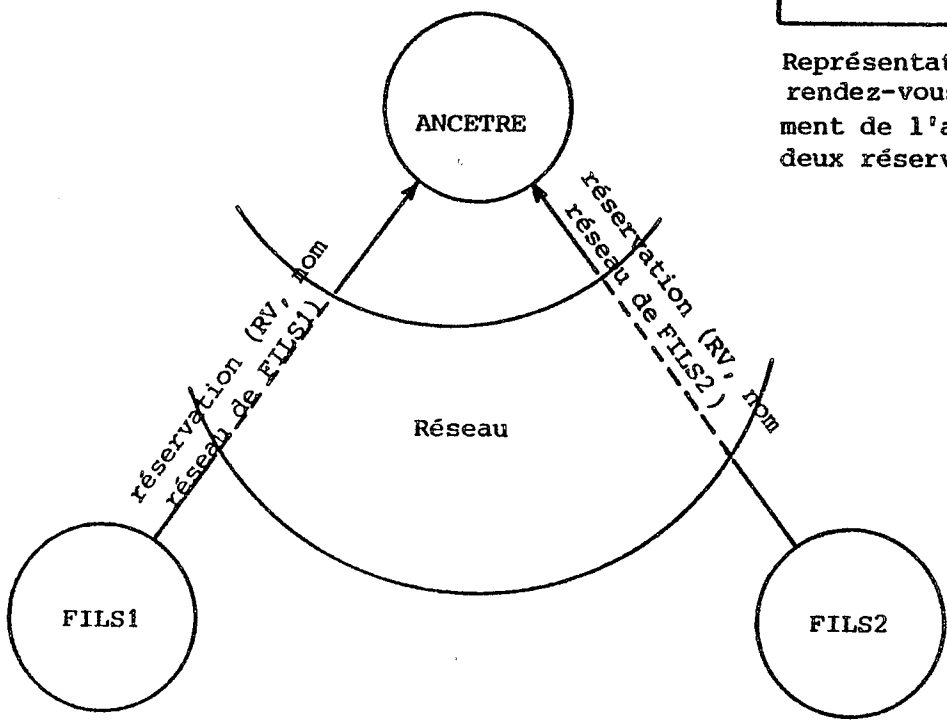
Note :

Les diagrammes présentés dans ce protocole permettent de mieux visualiser son fonctionnement. Pour faciliter les commentaires, on a donné des noms aux variables, tels que RV pour la variable de rendez-vous, FLOT pour la liaison explicite fils-fils, FILS1 et FILS2 pour les fils concernés.

RV

Nom réseau de FILS1
Nom réseau de FILS2

Représentation de la variable rendez-vous dans l'environnement de l'ancêtre après les deux réservations.



Les réponses à la réservation ne sont pas mises sur le diagramme.

4.b) Ouverture

La demande d'ouverture est composée de commandes :

ouverture (nom du paramètre effectif de mode rendez-vous, mot de passe).

La réponse est composée de commandes :

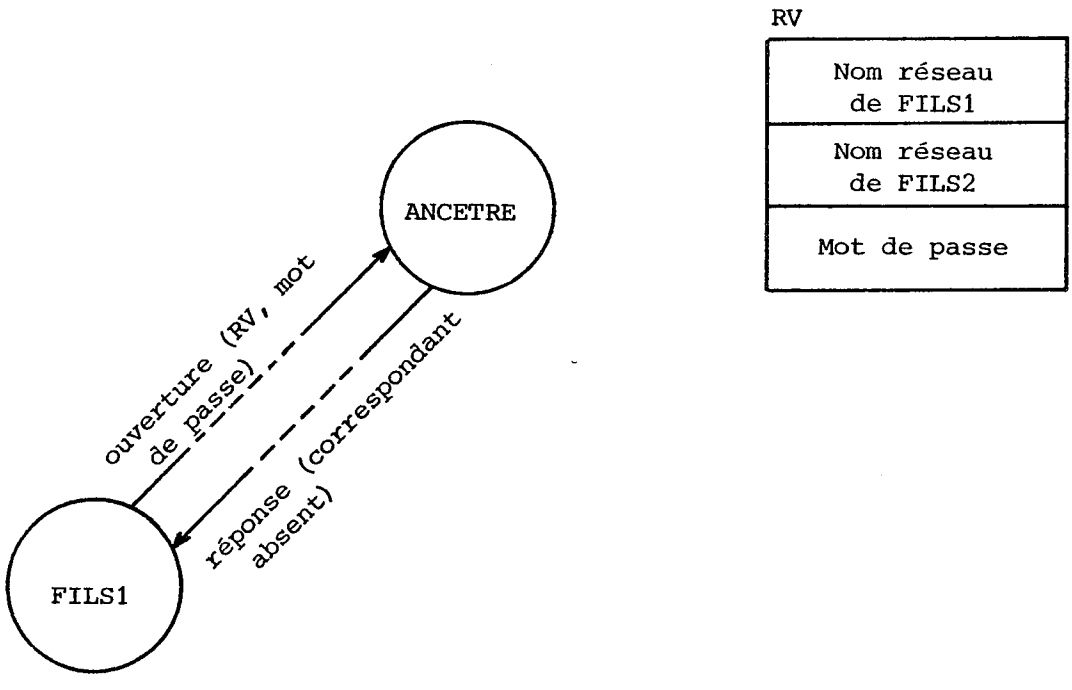
réponse à la demande d'ouverture (code de la réponse, coordonnées)

Trois situations sont à prévoir :

. la demande d'ouverture est faite avec un mauvais mot de passe,

. la demande d'ouverture du vis à vis n'est pas encore arrivée au rendez-vous (le processus fils qui reçoit cette réponse se met en attente d'une demande d'ouverture de la liaison réseau ; cette attente est couverte pas une horloge de garde) ;

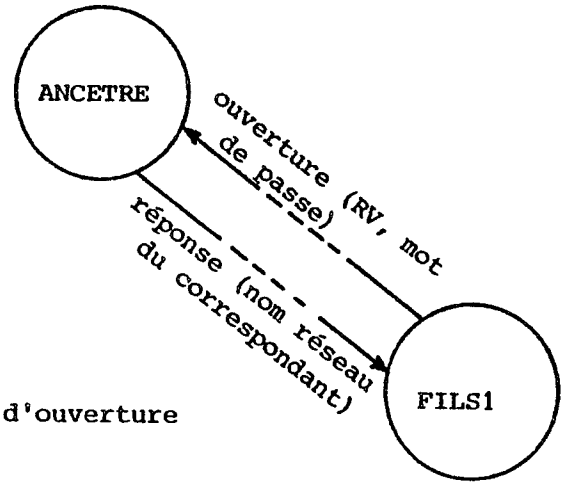
. l'ouverture distante a été demandée ; dans ce cas, les coordonnées donnent le nom réseau du correspondant (le processus fils qui reçoit cette réponse prend l'initiative de demande d'ouverture de liaison réseau avec son correspondant).



A la première demande d'ouverture

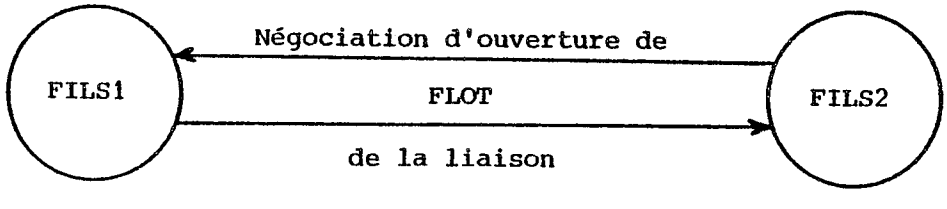
Note :

Bien que les noms réseau des deux fils se trouvent déjà dans la variable de rendez-vous, on ne livre le nom réseau du premier processus au second que quand les deux demandes d'ouverture sont arrivées sur le site de l'ancêtre.



A la deuxième demande d'ouverture

Selon le diagramme ci-dessus, l'initiative revient au processus FILS2, connaissant le nom réseau de FILS1, de procéder à la négociation d'ouverture de la liaison réseau explicite entre FILS1 et FILS2.



côté du processus qui a pris l'initiative

La liaison ainsi établie est dite *liaison explicite*.

Note :

Bien que la présence de l'ancêtre (c'est-à-dire l'accès à la variable de rendez-vous) soit nécessaire à l'établissement de la liaison explicite, le fonctionnement sur la liaison est indépendant du programme du père.

Les instructions de lecture et d'écriture sur la liaison sont traduites en des appels à l'interface réseau de SIGOR :

- demande de réception de lettre
- demande d'envoi de lettre

} la lettre contient en réalité la valeur de la variable chaîne d'octets

Comme la liaison fonctionne en contrôle de flux, le rédacteur est asservi au lecteur ; le premier ne peut écrire que si le second demande à lire. L'anticipation en lecture est cependant permise.

Note :

Chaque fils est immédiatement au courant de l'opérationnalité de la liaison explicite, dans la mesure où le logiciel de transport le prévient d'une coupure réseau, de la mort de son correspondant, ou d'une erreur irrécupérable.

S'il y a une quelconque anomalie dans le fonctionnement, il appartient au(x) processus vivant(s) de libérer la variable de rendez-vous.

4.3. Parallélisme entre processus

Il y a deux aspects dans l'expression du parallélisme entre processus :

. *l'aspect implicite* : SIGOR effectue une certaine synchronisation quand il réalise le lancement de processus à distance, l'échange implicite d'informations typées entre processus et l'échange explicite des informations entre processus. Cet aspect, qui est important, est géré par SIGOR et déjà exposé par ailleurs ;

. *l'aspect explicite* : l'utilisateur de la Machine Réseau Logique a besoin des outils et des objets pour exprimer le parallélisme propre à ses algorithmes.

SIGOR offre donc les possibilités suivantes de synchronisation :

- la déclaration de variable de mode événement,
- les instructions suivantes qui opèrent sur une variable de mode événement :
  - attendre un événement,
  - réaliser un événement,
- les instructions suivantes qui opèrent sur une file de p événements :
  - attendre n quelconques événements parmi les p ($1 \leq n \leq p$),
  - vérifier quels sont les n événements arrivés sur les p.

Cette méthode de synchronisation est effectivement imposée par le support de multiprogrammation de SIGOR : SYNCOP (DANG 1-77).

POSTFACE

Depuis ces dernières années, on constate une évolution rapide du domaine des applications réparties sur un réseau hétérogène d'ordinateurs et de celui des systèmes de traitement distribué. Dans la mesure où ces domaines sont très vastes, les besoins et les motivations sont divers. Citons cependant quelques spécialistes qui en soulignent les points critiques, lesquels servent de repères pour le travail présenté ici.

Une approche possible des applications réparties consiste à définir, selon L. POUZIN (POUZIN 77) : "... des noyaux autonomes, conçus pour servir directement les utilisateurs dans leur environnement. Selon les besoins, des couplages ont lieu avec d'autres noyaux, *par accès à des données et des traitements ... Des outils de communication* sûrs et performants sont indispensables pour intégrer cet ensemble de noyaux où chacun garde son autonomie de fonctionnement, tout en dépendant des autres pour une partie des services qu'il doit rendre".

Pour P. ENSLOW (ENSLOW 76), la définition d'un système de traitement distribué doit avoir les caractéristiques suivantes :

. deux ou plus de deux processeurs de n'importe quelle taille, du micro-processeur au grand ordinateur,

. un système d'exploitation de systèmes,

. l'emploi de communications type protocoles,

. les services doivent être demandés par nom,

. l'allocation de ressources est non déterministe.

Système de traitement distribué dont les points critiques semblent être (JENSEN 77) : le partitionnement des processus, la communication inter-processus et la connexion des processeurs.

On peut dire que l'approche du problème des outils pour l'expression des applications réparties et des systèmes de traitement distribué, s'est faite en deux étapes dans la conception de SIGOR :

. La première étape tient compte :

- d'une part de l'existence de la machine interprétative destinée aux langages de commande réseau (FARZA 74) ;

- et d'autre part des points critiques soulignés dans la page précédente et dont on a extrait les fonctions élémentaires à réaliser :

  . transport de programmes,

  . communication entre processus,

  . expression du parallélisme entre processus.

. La deuxième étape concerne les motivations dans le travail de définition de SIGOR : la motivation principale consiste à vouloir résoudre les problèmes de communication sur un réseau (dans une application répartie) avec des outils bien connus dans le domaine classique des langages de programmation et ce dans le but de rendre la vie plus facile à un utilisateur nouveau venu sur les réseaux qui a à écrire des applications réseau nouvelles ou qui veut transposer sur un réseau des applications locales existantes.

Ceci a donc amené à :

. définir la procédure comme unité de base de toute application répartie,

. réaliser une communication implicite entre processus par partage de variables communes exprimé par le paramétrage de procédure,

. réaliser une communication explicite entre processus avec l'emploi d'une variable de mode liaison et des fonctions ouvrir, fermer, lire et écrire, qui rappelle l'emploi de l'objet fichier dans les langages de haut niveau (WIRTH 70).

Il faut citer les travaux adjacents au travail de définition et de mise en oeuvre de SIGOR, mais qui concourent à étendre ses possibilités dans un avenir proche :

. sur la cohérence des objets à copies multiples sur le réseau (SERGEANT 77) ;

. sur l'étude exhaustive des problèmes posés par le partage de variables communes dans un réseau ;

. sur la comparaison de méthodes de synchronisation existantes (BEKKERS 77), (CAMPBELL 74), (CROCUS 75), (HANSEN 73), (HOARE 74), (LUCAS 75), (MOSSIERE 77), (ROBERT 77), ..., qui permettra de définir une expression du parallélisme plus évoluée que celle existante dans SIGOR.

Perspectives

L'énumération des prochains objectifs, conséquences logiques de ce travail, permet de mieux situer les domaines où la Machine Réseau Logique SIGOR peut jouer un rôle.

La perspective immédiate est la fin de la mise en oeuvre de SIGOR sur l'ordinateur IRIS 80 du Centre Interuniversitaire de Calcul de Grenoble.

Une perspective à moyen terme peut être de définir et de réaliser une maquette de système d'exploitation réseau qui permettra aux systèmes d'exploitation autonomes de faire partie intégrante d'une utilisation globale et avisée de leurs ressources et de leurs possibilités [BOCHMANN 77], [DUENKI 76], [RAYNER 77].

Dans la mesure où un système de gestion de base de données réparties a besoin d'un véhicule de transport de programmes ou d'un outil pour l'exécution à distance de requêtes, on peut chercher et trouver une utilisation de SIGOR dans le cadre d'un tel système [LE BIHAN 77].

Une perspective à long terme peut être de voir enfin se dégager un consensus pour la définition d'un langage de commandes réseau et une fois cela concrétisé, en réaliser le compilateur et sa mise en oeuvre site par site.

BIBLIOGRAPHIE

ANDRE E.,BOGO G., DECITRE P., GARDIEN R., PAYS P.

77 : Spécifications de réalisation de la station de transport ST2 sous SIRIS 8.
CII-HB Centre scientifique. Grenoble.

BEKKERS Y., BRIAT J., VERJUS J.P.

77 : Construction of a synchronization scheme by independent definition of parallelism.
Proc. IFIP TC2 Working conference on constructing quality software. Novosibirsk.

BELLISSANT C., LECARME O.

66 : Compilateur ALGOL 60 sur l'IBM 1130.
Document IMAG. Grenoble.

BOCHMANN G., PROBST W.G.

77 : Operating system design with computer network communication protocols.

BOUSSARD J.C., LECARME O.

77 : Didactique de la programmation
Ecole d'été de l'AFCET. Montréal.

BURROUGHS

Architectural design of the B6700 computer.
Burroughs Corp.

CAMPBELL R.H., HABERMANN A.N.

74 : The specification of process synchronisation by path expression.
Coll. sur les aspects théoriques et pratiques des systèmes d'exploitation. IRIA. Paris.

CHUPIN J.C.

74 : Control concepts of a logical network machine for data banks.
IFIP Congress. Stockhom.

CHUPIN J.C., SEGUIN J., SERGEANT G.

76 : Distributed applications on heterogeneous networks.
Prof. Conf. on computer networks and teleprocessing. Aachen.

CHUPIN J.C.

77 : Répartition d'applications et de bases de données sur un réseau général d'ordinateurs.
Thèse de doctorat d'état. USMG. INPG. Grenoble.

CLEARY J.C.

69 : *Proc. of the 4th Australian Computer Conference*. Adelaïde.

69 : Process Handling on Burroughs B6500.

CROCUS

75 : Systèmes d'exploitation des calculateurs.
Dunod.

DANG NG.X.

1-76 : Implémentation de la station de transport ST2 du réseau CYCLADES sous le système CP/67.
Rapport interne. (Jan. 76). USMG. INP3. Grenoble.

DANG NG.X., FOURNIER R., QUINT V.

2-76 : Système normalisé de commutation de processus.
+ Implémentation sous CP/67. (Jan. 76).
+ Macro-instructions SYNCOP sous CP/67. (Juin 76).
Rapports internes. USMG. INPG. Grenoble.

DANG NG.X., QUINT V., SEGUIN J., SERGEANT G.

1-77 : Présentation et définition de SYNCOP, un sous-système normalisé de commutation de processus pour la téléinformatique et les réseaux d'ordinateurs.
Rapport de recherche n° 64. Math. Appli. et Informatique.
USMG. INPG. (Jan. 77). Grenoble.

DANG NG.X., SERGEANT G.

2-77 : Process to process communication and synchronisation.
CCG Meeting. EIN project. (Fev. 77). NPL. Teddington.

DANG NG.X., QUINT V.

3-77 : The CICG approach of the IBM 360's (CP/67) connexion to the EIN network : problems and solutions.
CCG Meeting. EIN project. (Juin 77). Ispra.

DANG NG.X.

4-77 : System and portable language intended for distributed and heterogeneous network applications.
2^{nd} *Distributed processing workshop.* (Aout 77).
Brown University. Providence. Rhode Islands.

DANG NG.X., SERGEANT G.

5-77 : Expression of parallelism and communication in distributed network processing.
International conference on parallel processing.(Aout 77).
Shanty Creek Lodge. Bellaire. Michigan.

DIJKSTRA E.W.

61 : Making a translator for ALGOL 60.
ALGOL, bulletin supplémentaire n° 10.

77 : A discipline of programming.
Prentice Hall, series in automatic computation.

DUENKI A., SCHICKER P.

76 : Network job control and its supporting services.
ICCC'76. Toronto.

DU MASLE J.

66 : Compilateur ALGOL 60 sur Phillips PR8000.
Document IMAG. Grenoble.

DU MASLE J., GOYER P.

74 : Some basic notions for computer network command language.
IFIP Working conference on command language. Suède.

ELIE M., ZIMMERMANN H.

75 : Transport standard end-to-end protocol for heterogeneous computer network. *IFIP WG6.1. INWG 61.*

ENSLOW P.

76 : Distributed processing workshop transcripts.
ACM Computer architecture networks. Vol 5.

FARZA M., SERGEANT G.

74 : Machine interprétative pour la mise en oeuvre d'un langage de commande sur le réseau CYCLADES.
Thèse de troisième cycle. Université de Toulouse.

FERRIE J.

75 : Contrôle de l'accès aux objets dans les systèmes informatiques.
Thèse de doctorat d'état. Paris 7.

77 : Désignation dans les systèmes répartis et les réseaux.
Document de travail. Groupe Cornafion.

FORSDICK H., SCHANTZ R., THOMAS R.

77 : Operating systems for computer networks.
2^{nd} *Distributed processing workshop.*
Brown University. Providence. Rhode Islands.

GARCIA C,, GARCIEN R., MARCHAND A., MARTIN M., ZIMMERMANN H.

75 : Spécifications de réalisation de la station de transport ST2 portable.
SCH 536.2. IRIA. Paris.

GIEN M., SEGUIN J.

73 : FANNY.
Rapport sur CRIC. Centre Scientifique CII. Grenoble.

GIEN M.

75 : A file transfert protocol.
Document IRIA. IRIA. Paris.

GRANGE J.L.

76 : L'expérience d'un réseau de commutation de paquets : CIGALE, de la conception à l'exploitation.
Congrès de l'AFCET. Gif-sur-Yvette.

GRIFFITHS M, PELTIER

69 : A macro-generable language for the 360 computer.
Computer bulletin. Vol 3, n° 11.

HANDLE M.F.

76 : Implémentation sur Ordoprocesseur de SYNCOP.
Document IMAG. Grenoble.

HANSEN P.B.

73 : Concurrent programming concepts.
ACM Computing survey. Vol 5.

HOARE C.A.R.

74 : Monitors : an operating structuring concepts.
Communications ACM. N° 17.

IBM

a : CP/67.
b : PL1. Program Logic Manual.
c : PL1. Optimizing compiler. Language reference manual.
d : Supervisor and data management macro-instruction.

INWG

78 : Proposition d'un protocole de transport.
IFIP.

ISO

76 : ECMA Working paper on a possible information field structure for function-oriented and user levels.
ISO/TC97/SC6 (ECMA-87) 1277.
a : IA5.
b : EBCDIC.

JENSEN E.D.

75 : The influence of micro-processors on computer architecture : distributed processing.
ACM annual conference.

KAHN

The code breakers.
Sphere publications. USA.

LE BIHAN J.

77 : Presentation of the SIRIUS project.
Informal French-German seminar on distributed data bases.
Karlsruhe.

LE PALMEC J.

66 : Etude d'un langage intermédiaire pour la compilation d'ALGOL 60.
Thèse de docteur-ingénieur. Faculté des sciences. Grenoble.

LIU M.T.

77 : The distributed loop computer network.
2^{nd} *Distributed processing workshop.*
Brown University. Providence. Rhode Islands.

LUCAS M.

75 : Primitives de synchronisation pour langages de haut niveau.
Séminaire de programmation. USMG. INPG. Grenoble.

MARTINS J.M.D.G.

77 : Communication implicite entre des processus répartis sur un réseau hétérogène d'ordinateurs.
DEA Génie informatique. USMG. INPG. Grenoble.

MILLSTEIN R., SHAPIRO R.

77 : Reliability and fault recovery in distributed processing.
Technical report of Massachussets Computer Associates.
Wakefield. Massachussets.

MOSSIERE J.

77 : Méthodes pour l'écriture des systèmes d'exploitation.
Thèse de doctorat d'état. USMG. INPG. Grenoble.

POUZIN L.

75 : The CYCLADES network. Present state and development trends.
IEEE Symposium on computer networks : trends and applications.
Gaithersburg.

77 : Les réseaux informatiques.
Séminaire d'informatique. USMG. INPG. Grenoble.

QUINT V.

76 : Protection logicielle contre les erreurs dans un réseau d'ordinateurs hétérogène. Application au 360/67 du réseau CYCLADES.
Thèse de docteur-ingénieur. USMG. INPG. Grenoble.

RANDELL B., RUSSEL L.J.

64 : ALGOL 60 Implementation. The translation and use of ALGOL 60 Programs on a computer.
Academic press. London.

RAYNER D.

77 : An introduction to network job control language.
NPL report COM 88. NPL. Teddington.

RAZEK M.

76 : Contribution à l'étude des techniques de compression des données et de leurs domaines d'application.
Thèse de docteur-ingénieur. USMG. INPG. Grenoble.

ROBERT P., VERJUS J.P.

77 : Towards autonomous descriptions of synchronisation modules.
Proc. IFIP Congress. Toronto.

SCHICKER P., ZIMMERMANN H.

76 : End-to-end transport protocol.
EIN project.

SEGUIN J., SERGEANT G.

75 : Système normalisé de commutation de processus.
Centre Scientifique CII- ENSIMAG. Grenoble.

SERGEANT G, WILMS P.

77 : Automate de gestion d'objets dupliqués dans les systèmes distribués.
Note interne. ENSIMAG. Grenoble.

SERGEANT G.

78 : Le langage de macro-instructions d'utilisation de SIGOR.
Document IMAG. USMG. INPG. Grenoble.

SWAN, FULLER, SIEWIOREK

77 : CMM* A modular multi-microprocessor.
AFIPS NCC.

VAN DAM A.

77 : Transcripts of the 2nd Distributed processing workshop.
2nd Distributed processing workshop.
Brown University. Providence. Rhode Islands.

VERJUS J.P.

75 : Objets et fonctions primitifs d'un système d'exploitation.
Ecole polytechnique de Lausanne. Lausanne.

WIRTH N.

73 : The programming language PASCAL (revised report).
ETH Zurich. Zurich.

77 : Modula : a language for modular multiprogramming.
Software-Practice and Experience.

ZIMMERMANN H.

75 : Le protocole d'appareil virtuel dans CYCLADES.
Mini computers and data communications conference.
Liège.

76 : Transport sur un réseau.
Séminaire de téléinformatique.
Bonas.

CLASSEMENT DES RÉFÉRENCES BIBLIOGRAPHIQUES

Chapitre A

POUZIN 77, CHUPIN 77, FORSDICK 77, MILLSTEIN 77, LIU 77, SWAN 77, POUZIN 75, GRANGE 76, ZIMMERMANN 76, DANG 1-76, ANDRE 77, ELIE 75, GIEN 75, CHUPIN 76, CHUPIN 74, DANG 4-77, DANG 1-77, DANG 2-76, SEGUIN 76, VAN DAM 77

Chapitre B

B.1.

RAYNER 77

B.2.

FARZA 74, CLEARY 69, GIEN 73, SERGEANT 78, GRIFFITHS 69, IBM c, DIJKSTRA 77, BOUSSARD 77, WIRTH 70, VERJUS 75, WIRTH 77, CROCUS 75, ZIMMERMANN 76, FERRIE 76, ISO a, ISO b, ELIE 75, DANG 1-77, DANG 2-76

B.3.

DU MASLE 66, FARZA 74, RANDELL 66, LE PALMEC 66, BELLISSANT 67, DIJKSTRA 61, IBM b, SEGUIN 76, DANG 2-76, BURROUGHS, DANG 1-77, ELIE 75, SERGEANT 78, HANDLE 76

B.4.

DU MASLE 74, SCHICKER 76, INWG 78, ELIE 75, DANG 2-77, DANG 5-77, MARTINS 77, FERRIE 77, RAZEK 76, KAHN , DANG 1-77

Postface

POUZIN 77, ENSLOW 76, JENSEN 77, FARZA 74, WIRTH 70, SERGEANT 77, BEKKERS 77, CAMPBELL 74, CROCUS 75, HANSEN 73, HOARE 74, LUCAS 75, MOSSIERE 77, ROBERT 77, BOCHMANN 77, DUENKI 76, RAYNER 77, LE BIHAN 77

ANNEXES

1. SIGOR

1.1. Détails techniques sur le langage LI

1.1.1. Représentation des variables

Toute variable possède un repère dans la zone dynamique de la procédure dans laquelle elle est déclarée. Ce repère est représenté sur quatre octets :

. le premier indique le mode de la variable,

. les autres octets donnent l'adresse de la valeur de la variable dans la zone dynamique de valeurs de la procédure.

Note :

L'entier qui sert à indiquer le mode d'une variable dans son repère est donné par le numéro d'ordre de la variable dans la présentation suivante :

1) Variable de mode logique

Sa valeur est 0 ou 1 et elle est représentée sur un octet.

2) Variable de mode entier court

Sa valeur est comprise entre -2^{15} et $(2^{15}-1)$ et elle est représentée sur deux octets.

3) Variable de mode entier

Sa valeur est comprise entre -2^{31} et $(2^{31}-1)$ et elle est représentée sur un mot.

4) Variable de mode événement

Sa valeur est représentée sur un mot. Cette variable est strictement équivalente à la variable événement réalisée dans SYNCOP (SEGUIN 75) (DANG 2-76).

5) Variable de mode adresse

La partie adresse du repère de cette variable indique le repère d'une autre variable ou une adresse quelconque dans le code LI de la procédure dans laquelle cette variable est déclarée ou une adresse quelconque dans les zones dynamiques de cette même procédure.

6) Variable de mode fils

SSTATE	SINDX
SEXSITE	
SLIAISON	
a_1	b_1
	:
a_i	b_i

SSTATE indique l'état du processus fils :

- . en attente d'être activé
- . en activité
- . mort et cause de la mort (cessation d'activité, coupure réseau, ..)

SINDX donne le rang de la procédure affiliée (dont le processus fils est le processus interpréteur) dans la procédure appelante

SEXSITE indique le site d'exécution du fils sur 8 octets.

SLIAISON indique l'adresse du bloc de contexte abonné qui est attaché à la liaison implicite père-fils. Dans ce bloc de contexte, on trouve , entre autres, une information qui permet de retrouver la zone dynamique de la procédure appelante et le rang de la variable fils dans cette procédure.

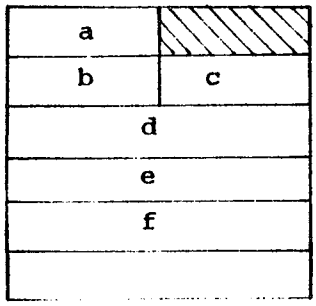
$((a_1,b_1), ..(a_i,b_i), ..)$ est une liste de couples de valeurs indiquant :

a_i : repère de la variable du père passée comme $i^{ème}$ paramètre par nom,

b_i : rang de la même variable dans la liste des paramètres effectifs donnée à l'activation de la procédure appelée.

Cette liste est vide si la procédure appelée est activée sans paramètre par nom. a_i et b_i sont représentés sur 2 octets chacun.

7) Variable de mode rendez-vous



a : état de la liaison :
- . en cours d'établissement,
- . en activité,
- . fermée et raison de la fermeture.

b : numéro de la porte du premier fils.

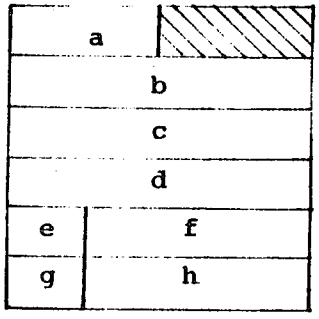
c : numéro de la porte du deuxième fils.

d : identification de la station de transport du premier fils.

e : identification de la station de transport du second fils.

f : mot de passe défini sur 8 octets.

8) Variable de mode liaison



a : état de la liaison.

b : événement associé à l'état de la liaison.

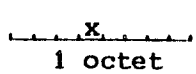
c : adresse du bloc de contexte abonné de la liaison.

d : rang du repère du paramètre de mode rendez-vous utilisé par la liaison dans la procédure du fils.

f : adresse de la liste des procédures de reprise d'erreurs (le nombre de ces procédures est $\leqslant 8$).

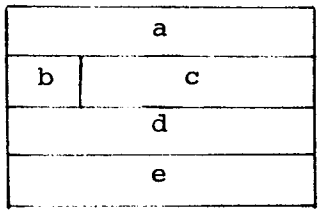
h : adresse de la liste des procédures de compactage et de cryptage (le nombre de ces procédures est $\leqslant 8$).

e et g : vecteurs de renseignement sur la présence de telle ou telle procédure d'erreur ou de service.



x { 0 non présente
  1 présente

9) Variable de mode bloc de réveil



a : délai de l'horloge de garde.

b : type de réveil.

c : adresse d'un événement ou d'une zone d'informations qui permet de retrouver la procédure (standard, affiliée ou cataloguée) destinée à traiter le réveil.

d : identification du processus propriétaire du réveil.

e : paramètres effectifs pour la procédure citée ci-dessus.

10) Variable de mode chaîne de caractères

Cette variable possède un descripteur qui comporte dans l'ordre :

- un code sur 1 octet signalant le descripteur de chaîne de caractères,
- l'adresse, sur 3 octets, de la valeur de la chaîne,
- la longueur, sur 2 octets, courante de la chaîne,
- la longueur, sur 2 octets, maximum de la chaîne.

11) Variable de mode chaîne d'octets

Cette variable possède un descripteur qui comporte dans l'ordre :

- un code, sur un octet, signalant le descripteur de chaîne d'octets,
- l'adresse, sur 3 octets, du début de la valeur de la chaîne,
- l'adresse, sur 4 octets cadrée à droite, de la fin de la chaîne d'octets.

Ce type de variable est destiné à recevoir des variables de mode tableau, de mode structure (au sens PL1), de mode enregistrement (au sens ALGOLW), ..

1.1.2. Les instructions du langage intermédiaire

Le LI dispose actuellement * :

1) des instructions arithmétiques :

. l'addition, la soustraction qui opèrent sur des entiers et des adresses,

. la multiplication et la division qui opèrent sur des entiers ;

2) des instructions logiques :

. et, ou, complément qui opèrent sur des logiques ;

3) des instructions de comparaison :

. plus grand que, plus grand ou égal, égal, plus petit que, plus petit ou égal

qui opèrent sur des entiers, des logiques, des adresses et des chaînes de caractères ; elles positionnent un code condition à vrai ou faux ;

4) des instructions de traitement de chaîne :

. définition de sous-chaîne,

. concaténation

qui opèrent sur des chaînes de caractères ;

. allocation et

. libération des chaînes d'octets ;

. calcul de la longueur et

. mouvement d'une cible vers une source d'une chaîne de longueur donnée

qui opèrent sur des chaînes de caractères ou des chaînes d'octets ;

---

* nous nous réservons la possibilité d'ajouter des instructions au langage intermédiaire pour nous adapter à une application donnée ou à un langage externe donné.

5) des instructions d'appel et de retour de procédure :

. appel séquentiel d'une procédure avec une liste de paramètres effectifs,

. appel parallèle d'une procédure avec une liste de paramètres effectifs, sous le contrôle d'un fils et sur un site donné,

. retour de procédure,

. retour de procédure avec la valeur de la fonction procédure,

. fin de procédure ;

6) des instructions de traitement du temps :

. armer un réveil,

. désarmer un réveil ;

7) des instructions de synchronisation :

. attente d'un événement,

. réalisation d'un événement,

. attente multiple de n événements sur p ($n \leqslant p$),

. vérification pour savoir quels sont les n événements arrivés sur les p attendus ;

8) des instructions de contrôle de processus :

. connaître l'état du père,

. connaître l'état du fils,

. suspendre un fils,

. réveiller un fils,

. tuer un fils,

. se tuer ;

9) des instructions de traitement de la communication explicite entre processus

. réserver une liaison via un rendez-vous,

. libérer une liaison,

. ouvrir une liaison avec un mot de passe,

. fermer une liaison,

. lire sur une liaison dans une chaîne d'octets,

. écrire sur une liaison le contenu d'une chaîne d'octets ;

10) des instructions de catalogage de procédure :

. cataloguer une procédure affiliée sur un site donné,

. mettre hors catalogue une procédure ;

11) des instructions de rupture de séquence :

. branchement conditionnel selon le code condition,

. branchement inconditionnel ;

12) des instructions de chargement (de valeur ou d'adresse) qui opèrent entre une cible et une source ;

13) des instructions de manipulation de pile :

. dépiler la pile de travail de l'interpréteur,

. empiler dans la pile de travail de l'interpréteur la valeur 0 ;

14) l'instruction de définition de constante.

1.1.3. Outils pour l'interprétation du LI

Pour interpréter une instruction LI, l'interpréteur IGOR dispose en plus des structures connues (compteur ordinal, chaîne statique, zones dynamiques de procédure, ..) :

. d'une pile (RANDELL 66),

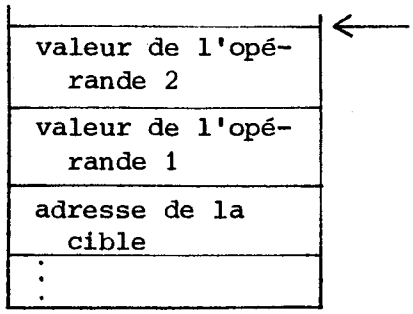
. d'une zone de travail sans structure particulière

. et du code condition.

Pour illustrer la manière dont l'interpréteur travaille, prenons l'exemple de l'interprétation de 3 instructions LI :

. l'addition de 2 entiers :

Quand l'interpréteur rencontre l'instruction d'addition, il s'attend à trouver la pile dans l'état suivant :



il réalise l'addition et charge la valeur trouvée dans la cible.*

Note :

La majorité des instructions LI s'interprètent avec l'aide de la pile qui contient soit les arguments de départ soit le code de retour de fin d'interprétation.

. la rupture de séquence conditionnelle :

L'opérateur aller_à est suivi d'une adresse et l'interpréteur réalise le branchement si le code condition est vrai.

---

* Ceci veut dire que l'instruction LE

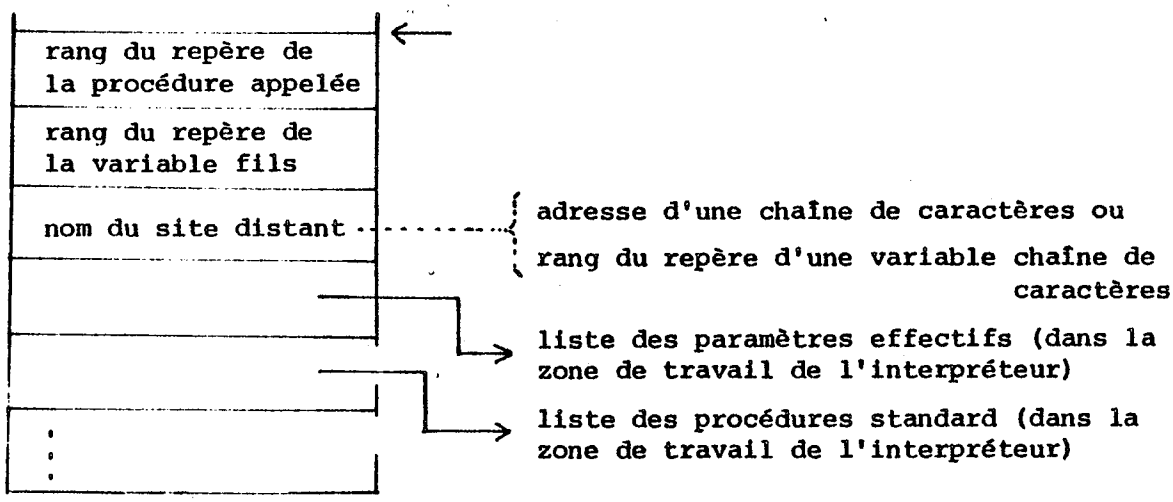
A := B + C ;

sera traduit en LI de la manière suivante :

Charger l'adresse de A dans la pile
Charger la valeur de B "
Charger la valeur de C "
Addition

. l'appel parallèle de procédure :

L'interpréteur s'attend à trouver la pile dans l'état suivant :



L'interprétation de cette instruction rend dans la pile un code retour qui indique le résultat du transport de programme.

Note :

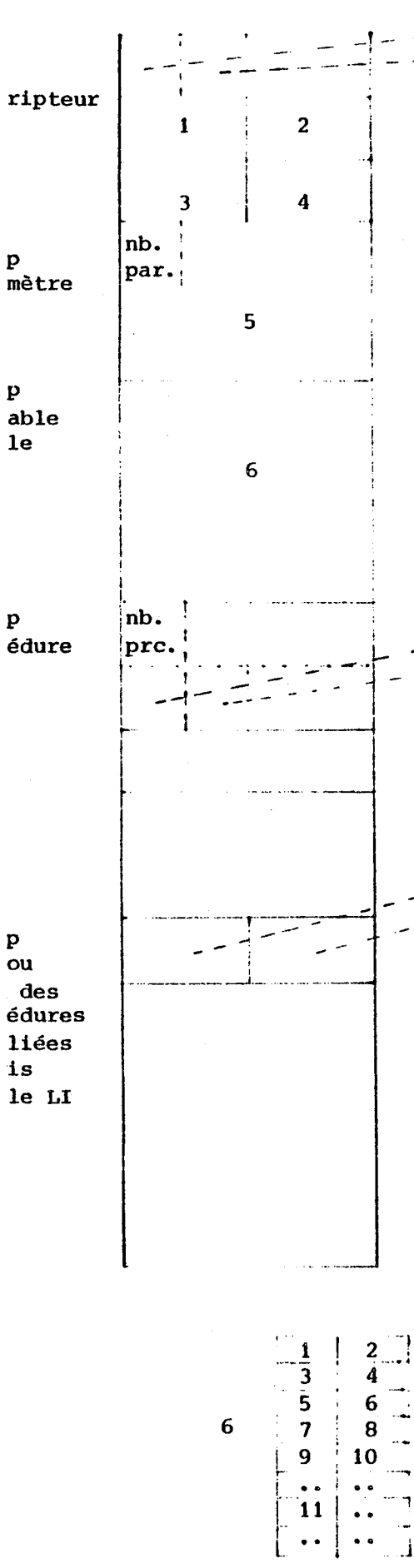
Ce n'est pas le propos de cet exposé de présenter complètement les détails de l'interprétation de toutes les instructions LI. Ils ont été mis au point pour la mise en oeuvre de SIGOR.

1.2. Exemple : Utilisation du passage de paramètre par valeur différée

```
A : PROCEDURE ;
    DCL (FILS1,FILS2) SON ;          variables de contrôle de processus fils
    DCL E EVENT ;
    DCL X INTEGER ;
    X = 0 ;

    B : PROCEDURE (PARB,EVENTB) ;
        DCL PARB INTEGER ;           paramètre passé par valeur
        DCL EVENTB EVENT BY NAME ;   paramètre passé par nom
        DCL Y INTEGER ;              variable locale
        WAIT (EVENTB) ;              attendre la création du second fils
        Y = PARB + 1 ;               première utilisation de PARB
        .
        .
    END B ;

    C : PROCEDURE (PARC,EVENTC) ;
        DCL PARC INTEGER BY NAME ;   paramètre passé par nom
        DCL EVENTC EVENT BY NAME ;
        .
        .
        PARC = 4 ;
        POST (EVENTC) ;              réaliser l'événement attendu par FILS1
        .
    END C ;

    ACTIVE B (X,E) SON(FILS1) SITE (K) ;  activer B sur le site K sous le con-
                                          trôle de FILS1
    ACTIVE C (X,E) SON (FILS2) SITE (H) ; activer C sur le site H sous le
                                          contrôle de FILS2
    .
    .
END A;
```

1.3. Description d'une procédure

DMIA



mode de la valeur de la procédure
si c'est une valeur chaîne, sa longueur

1 . déplacement par rapport au début du DMIA du début du champ paramètre
. valeur nulle si pas de paramètre

2 . déplacement par rapport au début du DMIA du début du champ variable locale
. valeur nulle si pas de variable locale

3 . déplacement par rapport au début du DMIA du début du champ procédure
. valeur nulle si pas de procédure

4 . déplacement par rapport au début du DMIA du début du champ LI

procédure ou fonction-procédure
non significatif si c'est une procédure
indique le mode de la valeur de la fonction-procédure
adresse relative du DMIA de la procédure affiliée par rapport au DMIA du père

taille exacte du code LI
taille arrondie du code LI (au multiple de 4 supérieur) + 4

5 chaque paramètre occupe un octet sauf celui de mode chaîne de caractères qui en occupe deux ; chaque paramètre peut être appelée :
. par nom
. par valeur
le demi-octet de droite sert à désigner le mode du paramètre formel. S'il est de mode chaîne de caractères, l'octet suivant indique la taille maximale.

6 6.1. nombre de variables logiques
6.2. " entiers courts
6.3. " entier
6.4. " événement
6.5. " adresse
6.6. " fils
6.7. " rendez-vous
6.8. " liaison
6.9. " bloc de réveil
6.10. variable chaîne de caractères suivi par sa longueur maximale sur 1 octet
6.11. variable chaîne d'octets suivi par sa maximale sur 3 octets

1.4. Vecteur d'état et noyau de l'interpréteur

Pour illustrer l'architecture du système SIGOR, prenons l'exemple suivant : soit un programme écrit en LE :

```
O : PROCEDURE ;
|   A : PROCEDURE ( , , ) ;
|   |   B : PROCEDURE ( , , ) ;        le transport de la
|   |   |   D : PROCEDURE ( , ) ;      procédure A sous forme
|   |   |   |                          compilée en LI vers le
|   |   |   END D ;                    site Y est demandé
|   |   END B ;
|   |   C : PROCEDURE ;
|   |   |
|   |   END C ;
|   END A ;
|   ACTIVE A (  ) SITE (Y) ... ;
END O ;
```

Les mnémoniques suivants sont pris pour désigner successivemen

. COMPTORD : le compteur ordinal,

. DISPLAY : la chaîne statique,

. DMIA (Delayed and Modular Interpretation Area) : le descripteur d'une procédure et

. DALI (Dynamic Area for the Local Interpretation) : la zone dynamique d'une procédure.

Les 2 figures suivantes illustrent deux états du noyau de l'interpréteur :

+ quand IGOR interprète l'instruction k de A et

+ quand IGOR interprète l'instruction s de IEBGENER, procédure cataloguée appelée dans D par exemple ; pour cette figure, on suppose que :

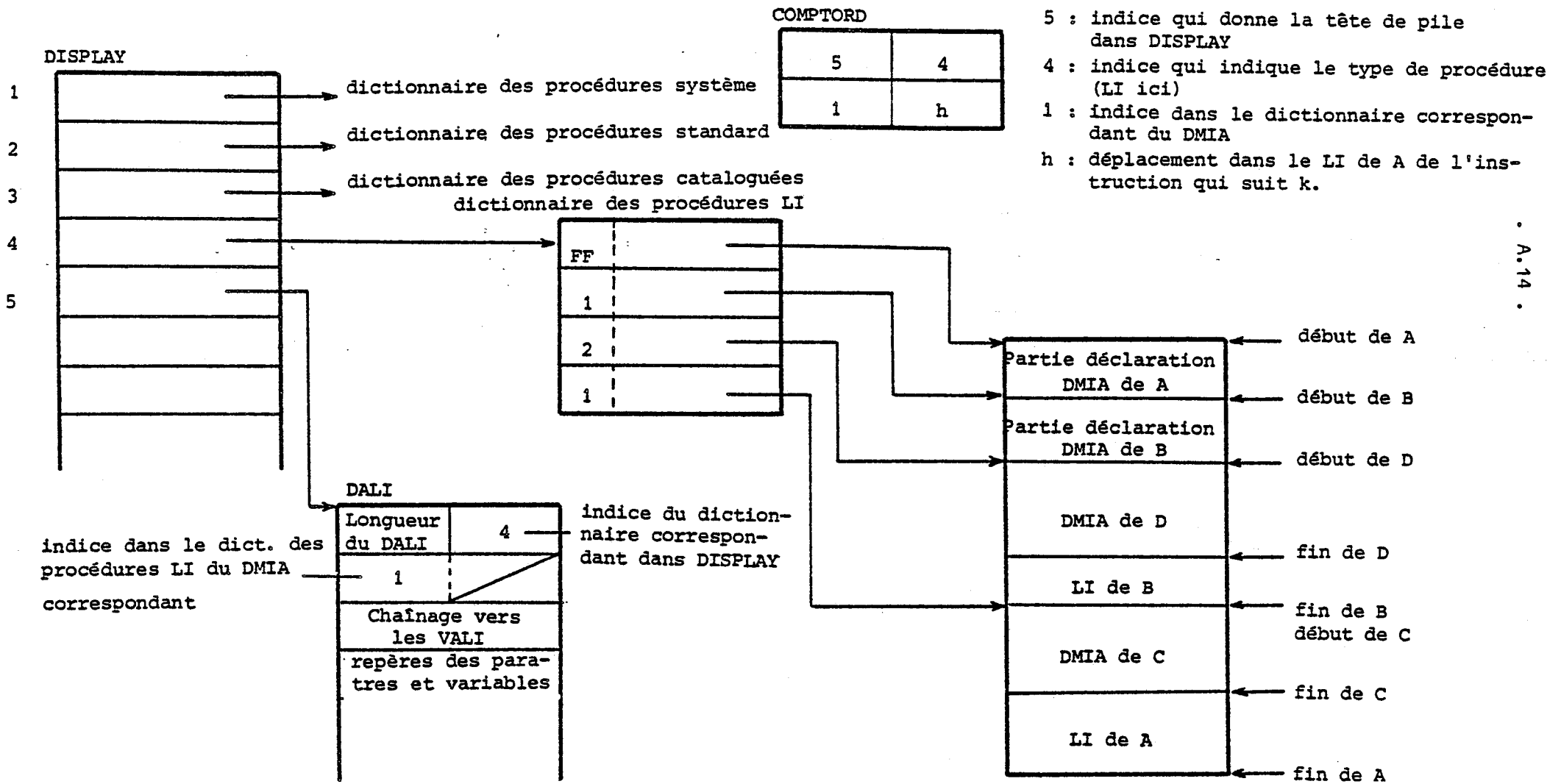
. dans A, l'adresse relative de l'instruction LI qui suit celle qui appelle B (CALL B (...)) est : u

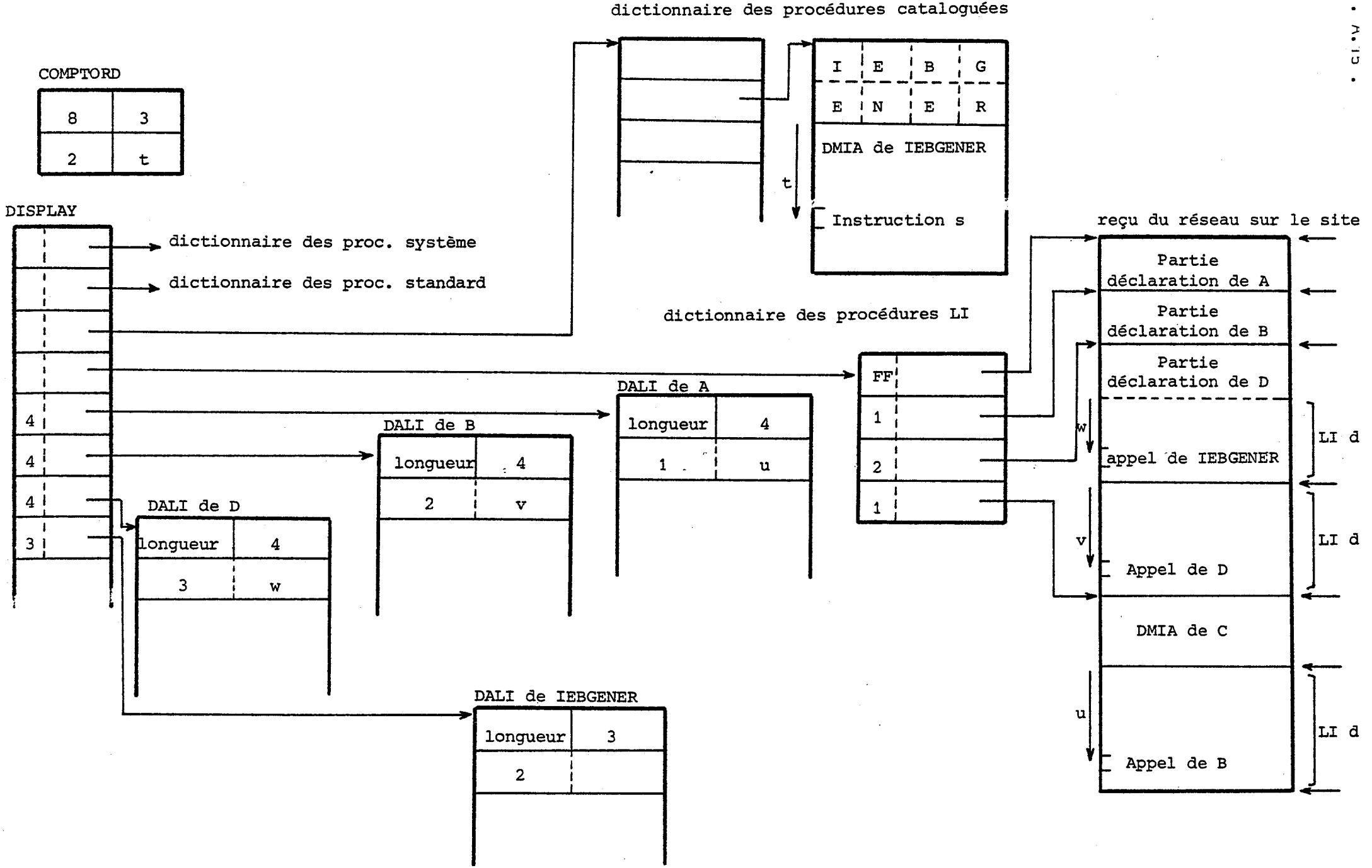
. dans B, l'adresse relative de l'instruction LI qui suit celle qui appelle D (CALL D (...)) est : v

. dans D, l'adresse relative de l'instruction LI qui suit celle qui appelle la procédure LI cataloguée, IEBGENER, est : w.

IGOR est en train d'interpréter l'instruction s de IEBGENER.

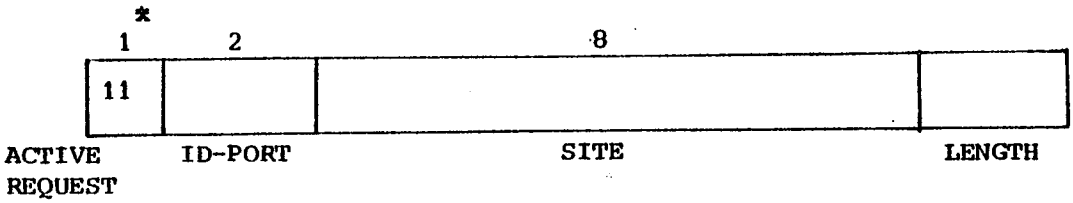
Illustration de l'état du noyau de l'interpréteur quand IGOR interprète l'instruction k de A :

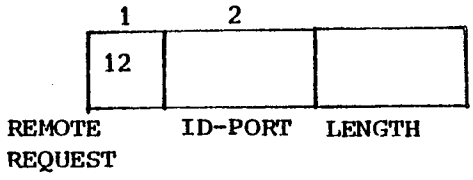


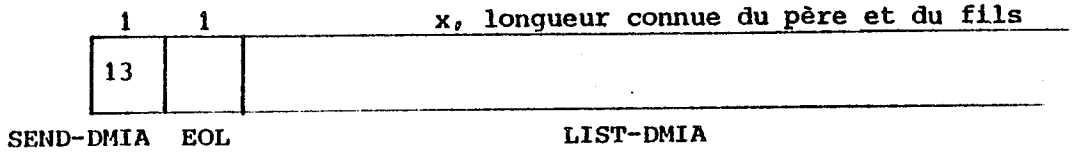

dictionnaire des procédures cataloguées
COMPTORD
8
3
2
t
I E B G
E N E R
DMIA de IEBGENER
t
Instruction s
DISPLAY
dictionnaire des proc. système
dictionnaire des proc. standard
reçu du réseau sur le site
dictionnaire des procédures LI
FF
1
2
1
Partie déclaration de A
Partie déclaration de B
Partie déclaration de D
appel de IEBGENER
LI d
Appel de D
LI d
DMIA de C
Appel de B
LI d
w
v
u
4
4
4
3
DALI de A
longueur 4
1 u
DALI de B
longueur 4
2 v
DALI de D
longueur 4
3 w
DALI de IEBGENER
longueur 3
2

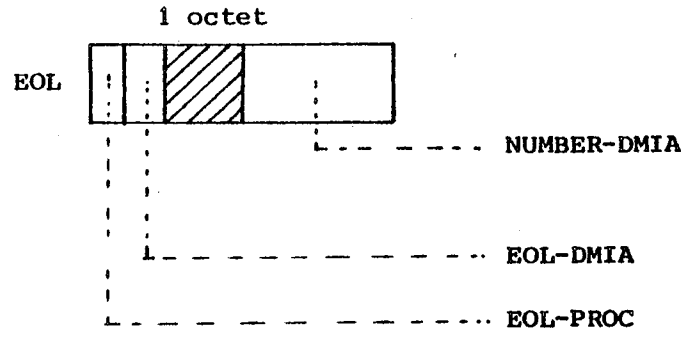
1.5. Détails techniques sur les protocoles utilisés par SIGOR

1.5.1. Commandes du protocole implicite de lancement de processus à distance







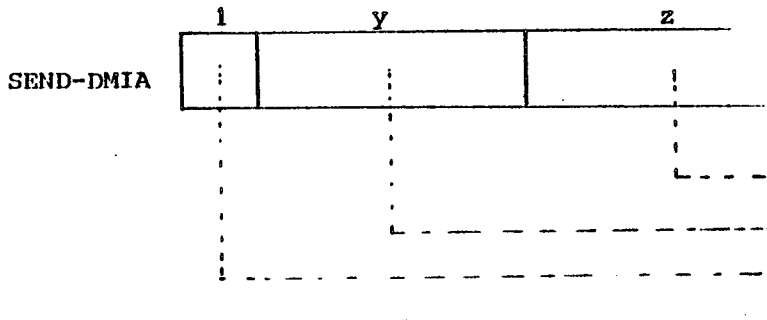


numérotation des SEND-DMIA d'un transport de programme commençant par 1

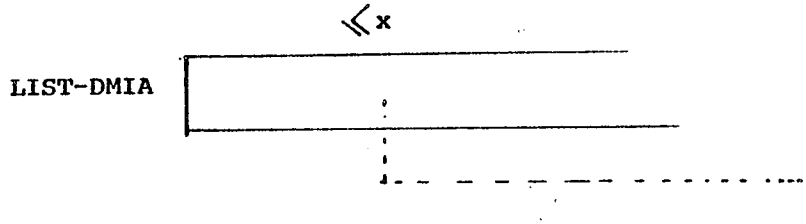
0 ce n'est pas la dernière commande
1 dernière commande SEND-DMIA

0 transport non fini
1 transport de programme fini

Si NUMBER-DMIA = 1 alors

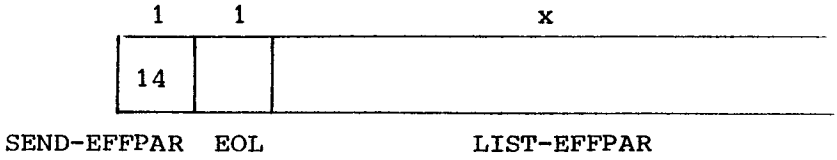


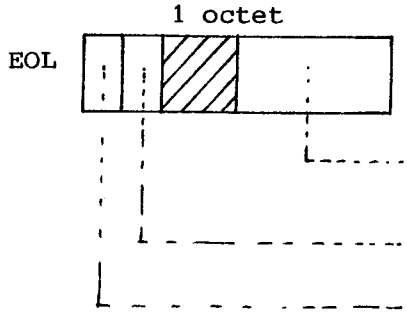
sinon



---

* unité de longueur utilisée : octet



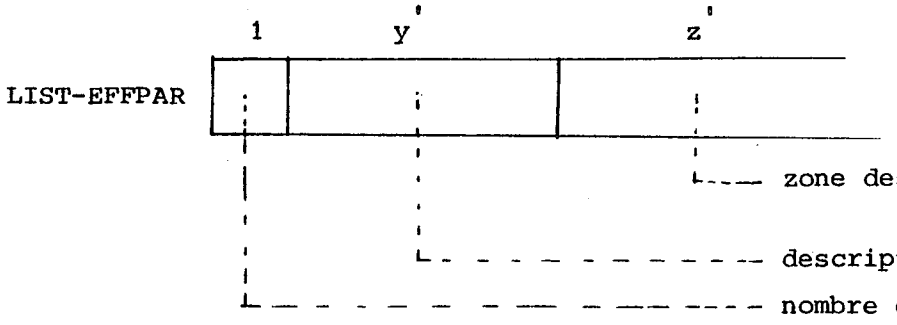


numérotation des SEND-EFFPAR commençant par 1

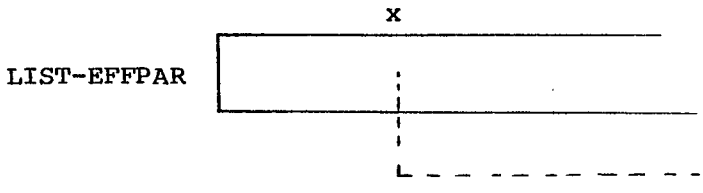
0 ce n'est pas la dernière comman
1 dernière commande SEND-EFFPAR

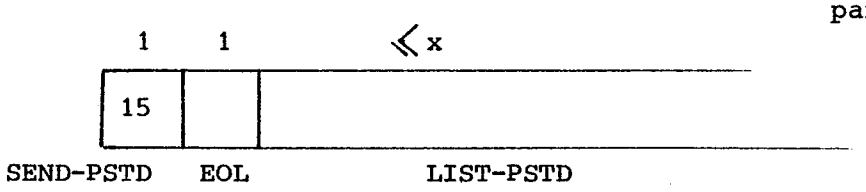
voir EOL-PROC dans SEND-DMIA

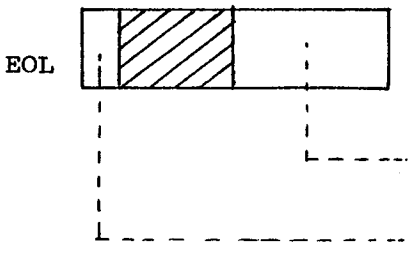
Si NUMBER-EFFPAR = 1 alors



sinon

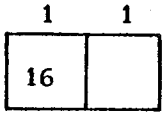






numéro de la commande SEND-PSTD commançant par 1

0 ce n'est pas la dernière command
1 c'est la dernière commande SEND-PSTD et fin du transport



REMOTE-ANSWER REASON

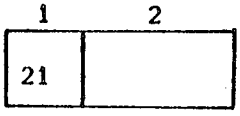
Les raisons sont :

0	OK
1	dépassement de capacité dans le dictionnaire des proc. LI
2	dépassement de capacité dans le dictionnaire des proc. std.
4	anomalie irrécupérable pendant l'installation ou pendant le transport

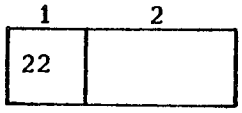
Note :

Cette commande peut être véhiculée par un télégramme si le service de livraison de télégramme est sûr.

1.5.2. Commande du protocole implicite d'échange d'informations entre père-fils

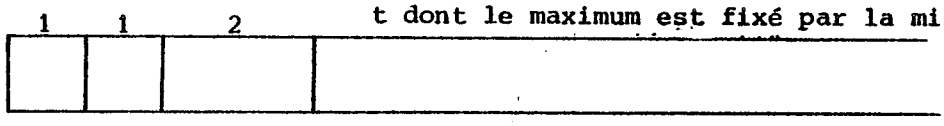


QUERY-VALUE EFFECTIVE-PARAMETER-ADDRESS

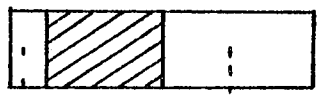


QUERY-VALUE EFFECTIVE-PARAMETER-ADDRESS
SEQUENTIAL

t dont le maximum est fixé par la mise en oeuvre



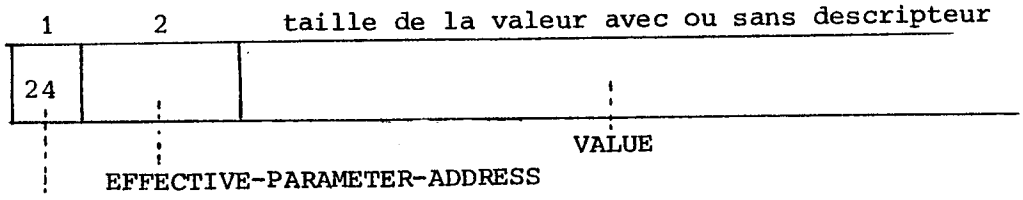
SEND-VALUE EOL EFFECTIVE-PARAMETER-ADDRESS VALUE



NUMBER-VALUE — numérotation des commandes SEND-VALUE pour une seule valeur envoyée (commençant par 1)

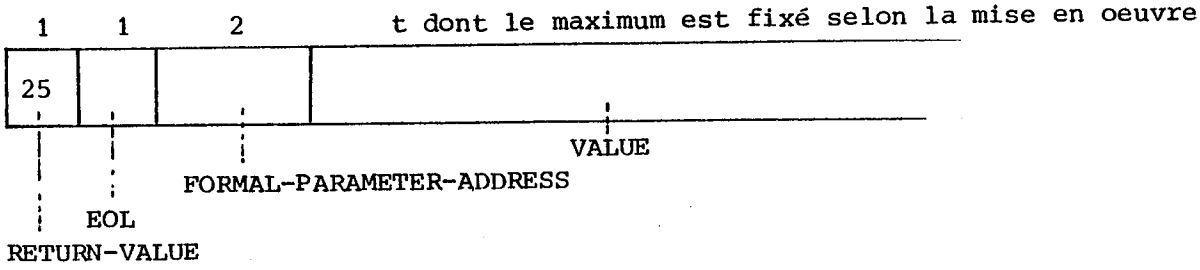
EOL — 0 ce n'est pas la dernière commande
1 fin de l'envoi d'une valeur

Si la valeur est une chaîne de caractères et si EOL = 0, alors elle est précédée par un descripteur qui occupe 4 octets.

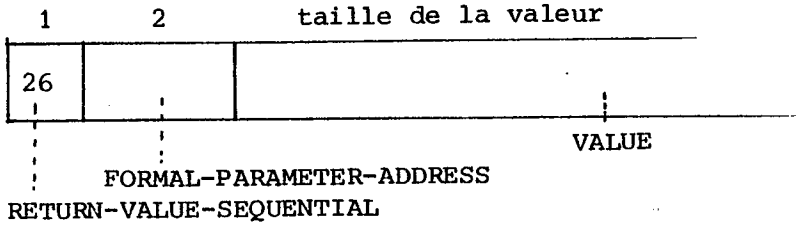


Si la valeur est une chaîne de caractères, alors elle sera précédée d'un descripteur qui occupe 4 octets :

MODE-VALUE	MAX. LGTH.	CUR. LGTH.

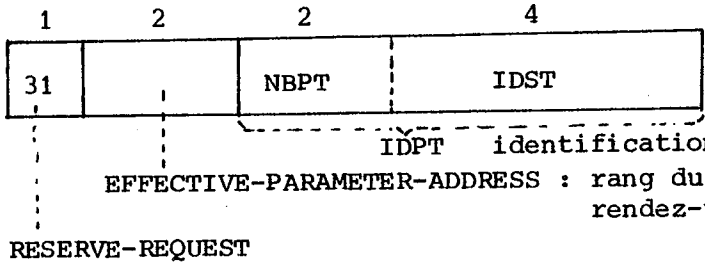


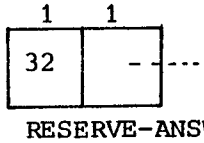
Les champs EOL et VALUE ont la même signification que dans la commande SEND-VALUE.



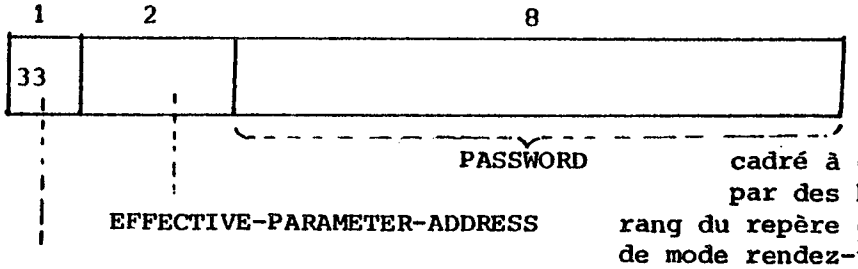
Le champ VALUE est identique à celui de SEND-VALUE-SEQUENTIAL.

1.5.3. Commandes du protocole explicite d'échange d'informations



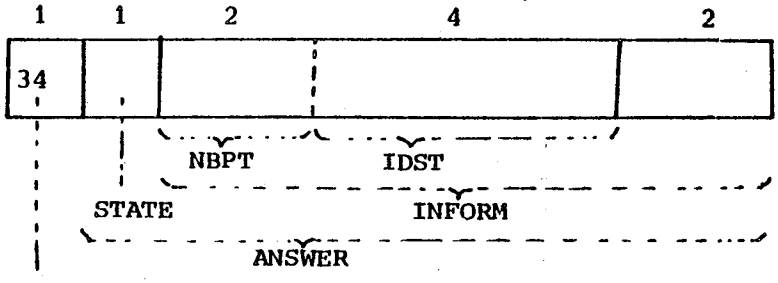


0 accepté
1 refusé (raison du refus sur le deuxième demi-octet)



PASSWORD cadré à gauche et rempli à droite par des blancs si sa longueur est < 8

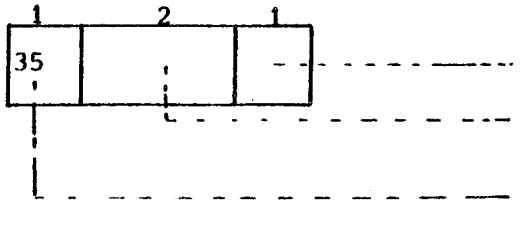
EFFECTIVE-PARAMETER-ADDRESS rang du repère du paramètre effectif de mode rendez-vous



si STATE = ACK (X'00') alors
NBPT + IDST donne l'identification de la porte du correspondant (nom de l'autre fils)

si STATE = NACK1 (X'01') alors
INFORM est non significatif

si STATE = NACK2 (X'02') alors
INFORM contient le mot de passe erronné.

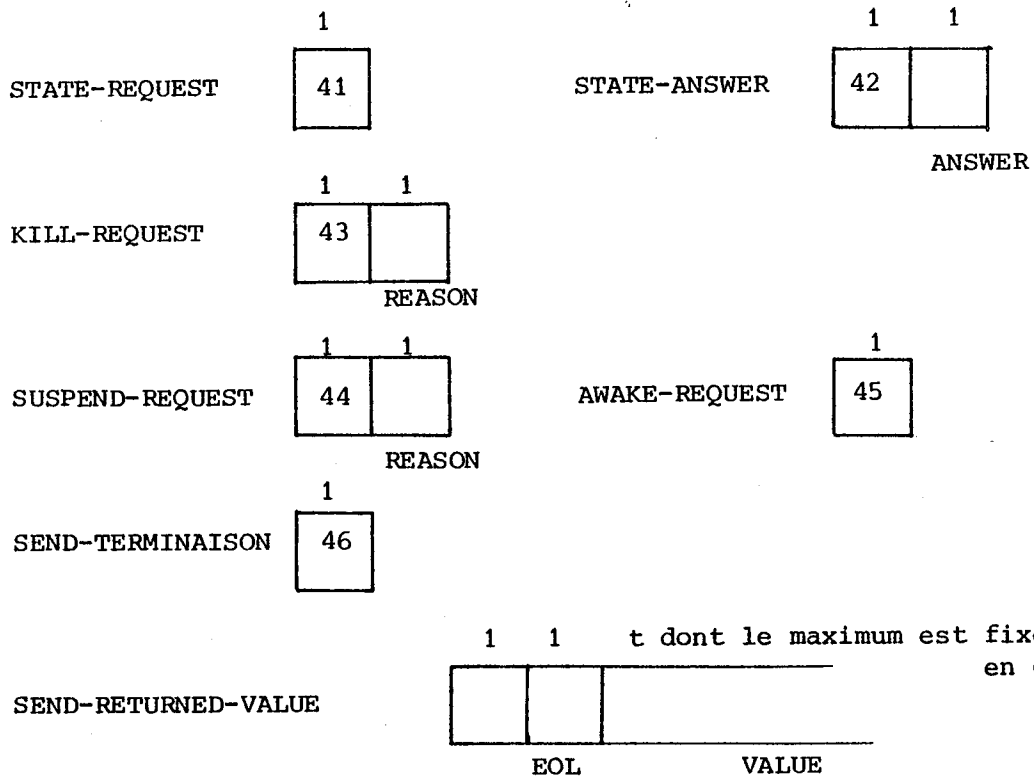


LSTATE état de la liaison

EFFECTIVE-PARAMETER-ADDRESS rang du paramètre effectif de mode rendez-vous

SEND-LIAISON-INFORMATION si la liaison n'est plus opérationnelle, le fils prévient l'ancêtre avec cette instruction.

1.5.4. Autres commandes



Les champs EOL et VALUE ont la même signification que dans SEND-VALUE du protocole implicite d'échange d'informations entre père et fils.

2. LA STATION DE TRANSPORT

Le logiciel qui réalise le protocole de transport de bout en bout (ELIE 75) défini sur le réseau CYCLADES s'appelle une station de transport.

Les objets manipulés par le protocole de transport peuvent être classés en trois groupes :

. le premier comprend les objets que l'on peut nommer par adresses :

+ la porte qui est soit une source pour l'envoi sans contrôle des informations, soit une cible pour la réception des informations ;

+ le flot qui est formé par un couple de portes et qui transporte des informations dans les deux sens avec ou sans fonctions de contrôle ;

. le deuxième groupe comprend les objets transportés :

+ la lettre et le télégramme qui servent comme champs pour le transport des informations ;

+ la commande d'acquittement de lettre et les commandes de négociation pour l'ouverture et la fermeture d'un flot avec des fonctions de contrôle ;

+ d'autres commandes ;

. le troisième groupe comprend les objets qui servent aux fonctions de contrôle et de négociation :

+ l'horloge de garde par exemple.

Sur des objets du premier groupe, sont définies des fonctions de contrôle et de reprises d'erreur qui assurent :

. que les lettres sont livrées dans l'ordre avec lequel elles ont été envoyées (contrôle de séquencement),

. qu'il n'y a pas de duplication de lettres à la livraison,

. qu'il n'y a pas de perte de lettres au cours du transport.

L'ensemble de ces trois fonctions constitue le contrôle d'erreur. L'utilisateur peut d'autre part demander un contrôle de flux qui comprend le contrôle d'erreur et qui assure l'asservissement de l'émetteur (source) au récepteur (cible).

Tout ordinateur connecté au réseau doit savoir gérer le protocole de transport. Dans le cas de l'IBM 360 fonctionnant en mode conversationnel (CP/67) et connecté au réseau CYCLADES, la station de transport est un logiciel (DANG 1-76) qui fonctionne avec SYNCOP comme système d'exploitation (QUINT 76).

L'interface avec l'abonné est réalisé par un ensemble de procédures :

. utilisables directement par l'abonné s'il se trouve dans la même machine virtuelle que la station de transport ou

. utilisables par l'abonné via une communication inter-machines virtuelles si l'abonné se trouve dans une autre machine virtuelle que la station de transport.*

La réalisation de cette station de transport est inspirée de la mise en oeuvre de la station portable (GARCIA 75) faite à l'IRIA.

D'autre part, l'IBM 360 (CP/67) est aussi connecté au réseau européen EIN. Selon les mêmes principes mais avec un autre protocole de transport (SCHICKER 76), une station de transport a été mis au point (DANG 3-77) pour offrir des services réseau EIN aux utilisateurs système de cet ordinateur.

---

* (ANSART & REY) : document de travail sur la communication logicielle entre deux machines virtuelles sous CP/67.

dernière page de la thèse

AUTORISATION DE SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 Avril 1974,

VU le rapport de présentation de :

- Monsieur J.P. VERJUS, Professeur à l'Université de RENNES

Monsieur Nguyen Xuan D A N G.

est autorisé à présenter une thèse en soutenance pour l'obtention du titre de DOCTEUR de TROISIEME CYCLE, spécialité "Génie Informatique".

Grenoble, le 23 Février 1978

Ph. TRAYNARD
Président
de l'Institut National Polytechnique

P.O. le Vice-Président,