THESE

Présentée à

L'UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIENE

Pour l'obtention du grade de

MAGISTER

Mention : **INFORMATIQUE**

UNE MACHINE VIRTUELLE ESTELLE POUR L'EXPERIMENTATION D'ALGORITHMES DISTRIBUES

Par :

Melle Semaoune Nabila

Soutenue le 09 Juillet 1992 Devant le jury Composé de

Mr A. Ainouche	*Président*
Mme . Abderrahim	*Examinateur*
Mr Z. Belmesk	*Examinateur*
Mr H. Khelalfa	*Examinateur*
Mr N. Badache	*Rapporteur*

REMERCIEMENTS

Je remercie le Professeur A. AINOUCHE, secretaire d'état aux postes et télécommunication, pour l'honneur qu'il me fait en présidant ce jury de these.

Je suis fort reconnaissante envers:

- Madame A. ABDERRAHIM, chargée de cours à l'U.S.T.H.B,
- Monsieur Z. BELMESK, maftre de conference à l'U.S.T.H.B,
- Monsieur H. KHELALFA, charge de recherche et chef du laboratoire de recherche et de developpement en informatique (L.R.D.I) au C.E.R.I.S.T,

pour l'honneur qu'ils me font en participant à la commission d'examen.

Mes remerciements et ma gratitude vont a Monsieur N. BADACHE pour l'interêt qu'il a porté à mon travaii, son aide et ses critiques constructives.

Je remercie tout particulièrement Monsieur M. Benhamadi, Directeur du CERIST. pour son acceuil au centre et la mise à ma disposition de son materiel informatique.

Enfin, je tiens à exprimer ma profonde gratitude a tous ceux qui, de près ou de loin, m'ont encouragée et soutenue.

S O M M A I R E

INTRODUCTION

INTRODUCTION

L'apparition des machines multiprocesseurs à haut degré de parallélisme et à mémoire distribuee et les réseaux locaux à grand debit a conduit a repenser la conception des alqorithmes. La conception de solutions aux problemes est vue en termes de plusieurs processus ou agents pouvant s'executer sur des processeurs distincts ayant chacun une connaissance partielle du probleme. Si l'alqorithmique séquentielle a été lonquement étudiée, il n'en est pas de même pour l'algorithmique dans un environnement parallele et distribue.. Cet environnement peut être aussi bien de type machines paralleles avec un rësseau d'interconnexion entre les processeurs ou simplement de type reseau local. Dans ce qui suit, nous definissons un système distribue comme étant un système unique dont les composants sont distribues sur un ensemble d'equipements informatiques, geometriquement repartis, interconnectés par un reseau de communication, et coop&-ant par échange de messages, pour realiser une tâche commune.

Un tel système est caractérisé essentiellement par:

- l'absence d'une mémoire commune,
- et, l'absence d'un référentiel de temps commun.

Il est évident que 1 es solutions adoptées dans les systèmes centralisés ne sont PIUS totalement applicables aux systèmes distribués.

Dans un tel environnement. un alsorithme distribué réalise généralement des fonctions de niveaux système d'exploitation telles sue I'exclusion mutuelle et I'interblocaue [MAH 88][KAD 89][RAY 851, et des fonctions propres à la répartition telles que la gestion des copies multiples ou tout simplement un calcul distribue.

Au vu de ces nouveaux problemes, la conception d'algorithmes corrects et fiables est difficile. Il est donc, nécessaire d'utiliser des logiciels d'aide à la conception. à l'évaluation et à la mise au point de proarammes distribues. C'est dans cette optique que se situe notre objectif. Il s'aqit de concevoir et de réaliser un outil d'experimentation et d'analyse de performances de programmes distribues et un outil de mise au point autour d'un même et unique environnement d'exécution distribue : une machine virtuelle Estelle répartie.

La conception et la réalisation de la machine virtuelle rdpartie constitue l'essentiel de cette thèse. L'outil de mesure et d'analyse de performances, et l'outil de mise au point se qreffent à celle ci et sont détaillés dans [BOU 921 [BEN 921.

Pour pouvoir étudier le comportemet des proqrammes distribués, nous proposons de les tester dans un environnement d'execution reel. 11 est donc nécessaire de disposer d'un outil de specification oermettant la description du comportement à analyser. Cette spécification est ensuite interprétée afin de générer un code distribué executable sur une architecture répartie.

L'execution de ce code nécessite un noyau d'execution qui offre les services necessaires à l'enchaînement des tâches, la synchronisation et la communication. La machine virtuelle Estelle dispose d'une couche appelee noyau d'exécution réparti charge d'offrir ces fonctionnalites dans un environnement distribue. Ce noyau comprend:

- Un module charge de la configuration, de l'initialisation du système et du lancement du scheduler qui assure l'enchafnement des instances locales,

- un environnement de travail Estelle offrant les primitives de manipulation des obiets Estelle n'existant pas dans Pascal et prenant en charge la communication distante.

une interface système disposant d'un ensemble de set-vines dont la qestion du temps, la transmission des interactions et la correspondance entre les instances et leur site d'execution.

Cette thkse se d&compose comme suit:

Le premier chapitre est consacré a l'introduction d'outils d'étude , d'analyse et de mise au point de proqrammes distribués de manière générale. Après avoir donné un aperçu sur les notions fondamentales de l'algorithmique distribuee, les points importants des systèmes de mise au point sont mis en evidence.

Le second chapitre est une presentation générale des bases semantiques et de l'utilisation de techniques de description formelles pour la conception et la specification de, prosrammes distribués.

Le troisième chapitre présente dans le detail la technique de description formelle Estelle choisie pour la specification des programmes distribues. Cette specification est. ensuite soumise a un interpréteur générant un code distribue executable. Ce générateur est décrit dans le chapitre IV. La mise en oeuvre et l'exécution de prosrammes distribues specifies en Estelle sont assurées par un noyau d'exécution distribué, qui est présenté dans le dernier chapitre.

CHAPITRE I

NOTIONS FONDAWENTALES DE L'ALGORITRMIQUE DISTRIBUEE ET OUTILS DE MISE AU POINT ET D'ANALYSE DE PERFORMANCE

I.1 NOTIONS FONDAMENTALES DE L'ALGORITRMIQUE DISTRIBUEE

Un algorithme distribue est un ensemble de processus coop&rant pour réaliser une tâche commune. Il existe deux classes d'algorithmes distribues:

- les algorithmes de calcul,
- les alaorithmes de contôle.

Contrairement aux algorithmes sequentiels, les algorithmes distribues doivent tenir compte de nouvelles contraintes telles que l'absence de mémoire commune. I'absence de référentiel de temps commun [DUD 871, et la non perception d'un état global instantane [LAT 871

Un certain nombre de caracteristiques propres aux algorithmes distribues et certains concepts de base sont énoncés dans les paragraphes qui suivent.

1. Qualités d'un algorithmes distribué

La conception d'un algorithme distribue peut être definie selon un certain nombre de caracteristiques qui permettent de les comparer et de les évaluer [RAY 851. Parmi ces caracteristiques nous citons:

- le degré de répartition: cette notion est interessante pour évaluer un algorithme distribue. Elle est liée a la symdtrie des rôles joués par les differents processus. On parle de non symetrie lorsque les textes des processus sont différents (client/serveur) et dans ce cas le degré de repartition est faible. Et on parle de symétrie totale dans le cas contraire. Dans le cas intermediaire. on parle de symétrie de texte et de symétrie forte [RAY 851.

- la resistance aux pannes: cette caractéristique est liée à la premiere. En cas de panne, la perte d'un processus jouant un role particulier risque d'être fatale pour une application distribuee. Cependant, si

tous les processus jouent exactement le même rôle, le problème est résolu en remplaçant le processus en panne.

- les hypothèses sur le réseau: En général, lors de l'écriture des algorithmes, un certains nombre d'hypothèses sur l'environnement et le réseau de communication sont supposés. Cependant, un algorithme distribué qui pose moins de contraintes est plus intéressant car on peut l'appliquer à une plus grande variété de systèmes. Généralement, les hypotheses sur le réseau de communication sont: le non déséquencement de messages, la non duplication. la non perte de messages, . . . etc.

le trafic engendré: celui-ci peut se mesurer par le nombre de messages, la charge induite par le réseau et le temps d'attente des processus. Plus le trafic est moindre, plus performant est l'alaorithme.

- l'état global et local: un alqorithme distribué est plus intéressant quand les processus qui le composent peuvent prendre des decisions sans avoir besoin de connaissance globale. Ceci permet d e minimiser le nombre de messages échangés nécessaires à la synchronisation et assure une meilleure resistance aux pannes.

2. Concepts de base

La conception de la plupart des algorithmes distribués de contrôle est basée sur les concepts suivants:

Le jeton c i rculant

Cette technique consiste à faire circuler un i-ton qui est un privi lége sur un anneau de processus. Le processus ayant le privilege est le seul à pouvoir réaliser la tâche en exclusion mutuelle. Ce privilége est matérialisé par un message ou une variable d'état.

La calcul diffusant

Le calcul diffusant [DIJ 801 consiste en une enquête pour récupérer un résultat qui oeut servir à la prise de decision. Cette enquête est réalisée en effectuant un parcours d'arbre dont les noeuds sont les sites du réseau et les branches sont les liens de voisinage.

Le transfert de connaissance

Cette technique consiste à acquerir des informations d'un ensemble de processus pour avoir une vue globale ou partielle *sur* l'état du système[HEL 851. Cette technique est réalisée en diffusant des messages qui s'enrichissent en passant par chaque noeud du réseau.

Estampillage

Dans le but de dater des Cvenements, ou tout simplement d'établir un ordre partiel ou total afin de resoudre certain conflits, Lamport proposa le principe d'estampillage [LAM 781. Il consiste à maintenir une horloge logique H qui est en fait un compteur, au niveau de chaque site. Afin de permettre l'évolution correcte de cette horloge, tous les événements spontanés, emission et réception sont estampillés. Une estampille est un couple (valeur horloqe. numero du site). La mise à jour d'une horloae H se fait comme suit:

- A lbccurence d'un événement: H := H + 1,
- A la reception d'un message m d'estampille (H_m,$site_m$) l'horloge est mise à jour comme suit:

$$H := \max (H_m, H) + 1 ;$$

De cette maniere, un événement A precede un événement B, si:

$$H_A < H_B$$
ou
$$H_A = H_B \text{ et } site_A < sites$$

I.2 GENERALITE SUR LES SYSTEMES DE MESURE ET MISE AU POINT

Effectuer une exclusion mutuelle ou encore qérer des données dupliquées sont des problèmes difficiles A mettre en *oeuvre* dans un environnement réparti. L'asynchronisme induit par le fonctionnement des processus et des lignes de communication rend difficile la possibilité de capter instantanement l'état global

d'un programme distribue [LAM 85][HEL 85][LAI 87]. Ce non déterminisme complique la maîtrise du comportement d'un système distribue. En effet, énumérer toutes les situations elementaires où encore suivre les ramifications du graphe des chemins d'exécution pour determiner le comportement d'un algorithme distribue n'est pas une tâche facile.

Pour aider à la conception et à la comprehension de ces alsorithmes, differents outils ont été conçus. A l'heure actuelle il en existe qui se distinguent par la compléxité du problème qu'ils peuvent traiter et par la nature des résultats fournis [BRO 87].

L'analyse de protocoles ou programmes distribues s'effectue generalement selon les trois étapes suivantes:

- expression informelle du service attendu.
- specification formelle.
- &valuation des performances ou vérification des propriétés.

La derniere étape s'appuie sur une description formelle et un modèle qui appartiennent à l'une des categories suivantes [GRO 87]:

Les automates:

Le.5 automates à états finis se prêtent bien à la description des systèmes distribues. mais ils deviennent trés vite lourds à manipuler à cause de l'explosion rapide du nombre d'états [MEN 82] [GRO 891.

Les réseaux de petri:

Les réseaux de petri sont appréciés pour leur puissance d'expression. Plusieurs extensions des réseaux de petri simples ont été vues (réseaux temporises. colorés, temporels, à prédicats...).[FAN 90] [MEN 821.

Arbres de synchronisation:

La representation d'un comportement ear l'arborescence de ses evolutions possibles est aussi *un* modèle fondamental du parallélisme [GRO 89][MIL 801.

La logique temporelle:

Introduite par Pnueli, elle se présente comme une extension de la losique propositionnelle. Elle permet de specifier des propriétés classiques (bouclage, terminaison,..) et des propriétés qui se réfèrent au deroulement futur des événements [AUD 881 [JEZ 91].

Les langages:

Très souvent la description des protocoles et algorithmes distribués est faite directement dans un langage plutôt que dans un modèle. On peut citer quelques langages utilises tels que: Estelle[DEM 873 [DIA 89], LOTOS [DIA 89] [FAN 88] [LAN 88], ADA, LDS [DIA 89].

Les langages et reseaux de petri restent les plus utilisés directement; les automates aparaissent comme support à beaucoup de méthodes de verification et validation.

Parmi les tentatives effectuees dans ce domaine, on peut citer:

VEDA:[JAR 88] est un simulateur de protocoles specifies en Estelle. Il est possible de simuler le deroulement de l'algorithme en utilisant des observateurs qui vérifient des propriétés sur l'algorithme et effectuent des traces au cours de la simulation.

XESAR:[DIA 89] un logiciel de validation. 11 permet de verifier la conformite de la description d'un système. La specification est un ensemble de formules de logique temporelle. La méthode d'évaluation est fondée sur le test des formules sur un graphe d'états engendré par le programme de description. Le graphe est de type graphe de marquages et comprend les états dans lesquels peut se trouver le programme. C'est un outil de preuve.

OGIVE/d: [MEN 82] un outil de validation fondé sur la construction et l'analyse de reseaux de Petri. Il permet de decrire et d'assembler automatiquement des sous-réseaux, et de les valider sur le graphe des marquages.

Remarquons que la simulation est une technique qui permet d'évaluer et de valider des programmes distribués. Cependant, on ne peut étudier qu'un nombre limité de comportements d'un programme tout en simulant certains paramètres qui peuvent être assez significatifs. Une seconde technique, la verification formelle n'est appropriée qu'aux programmes ayant un niveau d'abstraction très élevé.

Cependant, l'expérimentation d'algorithmes distribués dans un environnement reel reste la meilleure approche pour apprehender les difficultés de mise en oeuvre et évaluer leurs performances. Elle permet principalement:

de confirmer ou d'infirmer des résultats theoriques
- de mettre en évidence des comportements non prévus par la thèorie.

I.2.1 Introduction aux systèmes de mesure

Un système d'experimentation doit nous fournir des informations sur les performances des programmes par rapport à des criteres de qualité et d'efficacite prédéfinis ou construits à la demande (taux d'utilisation d'une ressource, charge du réseau, verification de propriétés, degré de parallélisme,...). Un système d'experimentation procède en deux phases: dans la premiere. il cherche à obtenir à moindre coût des informations de performances, alors que dans la seconde phase, il analyse les données collectées (figure I.1). Toutefois. toute méthode loaicielle de mesure induit une perturbation du système mesuré. Cette perturbation ne doit pas être négligée.

I.2.1.1 Phases du système de mesure

Une synthèse des outils de mesure et étude de performances proposées dans [BAD 91] [BOU 92] montre qu'en général, deux phases sont mises en evidence:

Phase 1:

1. Collecte des données brutes:

C'est la phase la plus délicate des systèmes loaiciels de mesure. Il s'agit généralement de trouver un compromis entre:

* la fidèlité et la précision afin de fournir des informations exactes.

* une moindre perturbation de l'activité du phénomène mesuré.

Les informations sur les événements sianificatifs sont récoltées selon un mécanisme préalablement choisi. Ceci peut se faire en plaçant des sondes ou "probes"

qui sont des proqrammes charges de capter des informations sur ces événements pour l'étude souhaitees [MILb 88] [WIB 88] [YAN 87]. La collecte d'informations est orientée selon le programme étudié et l'analyse visée [SNO 88].

2. Filtrase des donnees:

Les outils de meoure risquent d'extraire une qrande quantité d'informations. Il est donc, indispensable de fournir des moyens selectifs pour ne considdrer que le sous ensemble de données utiles. La selection ou le filtraqe, peut être mis en oeuvre pendant la phase de collecte au moyen de prédicats évalués dynamiquement pendant la mesure, ou par le positionnement de drapeaux grace à des paramètres d'entrees [HAD 88][SNO 88] [MIL 86] [MILb 88].

Phase 2:

3. Analyse des données:

L'analyse des résultats est beaucoup plus classique: la seule difficulté est de retrouver facilement la sémantique des donnees recueillies.

Deux apuroches sont possibles: les analyses peuvent se faire antomatiauement ou manuellement par l'usaqer. Dans le premier cas, le type d'analyse à opérer peut Ctre prédéfini dans le système, spécifié par l'utilisateur. ou une combinaison des deux possibilités [SNO 88].

4. Affichaqe des résultats:

Elle constitue l'interface entre l'utilisateur et le systbme de mesure. Certains systèmes permettent l'interaction avec l'usaqer pendant toute la session de mesure [WIB 881. Les résultats de mesure sont présentés sous différentes formes: tables, histogrammes,...etc [YAN 871.

5. Guide et predictions

Bien que l'on essaie toujours d'obtenir des informations compréhensibles afin de faciliter l'analyse. face à un qrand nombre d'informations, il n'est pas facile à l'analyste de localiser les problèmes. L'idée serait donc, de guider ce dernier à prédire les causes de dégradation de performances et ceci de manière automatique [YAN 88][MIL 851 [MIL 90].

Un graphe d'histoire d'exécution peut identifier les endroits oil **apparaissent les problèmes d'exécution. Ceci permet de focaliser** l'attention à un niveau donné.

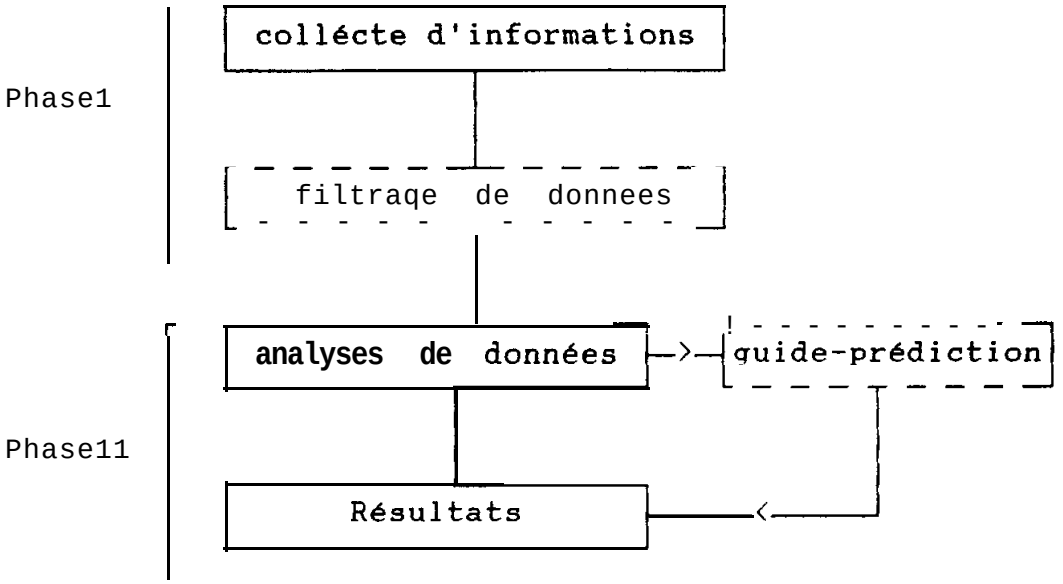


Figure I.1: Architecture d'un système de mesure

I.2.1.2 Niveaux d'intervention du système de mesure

Comment prendre ces mesures **à partir d'une spécification?**

La mesure de performance nécessite l'ausmentation des programmes à observer - adionction de code necessaire à l'observation et à la prise de mesure -. Celle-ci peut intervenir à différents niveaux. Elle consiste généralement, en la detection et le traçage des événements intéressants.

Il existe plusieurs approches qui sont décrites et discutees dans [HAD 881.

1. Niveau noyau:

Les modifications sont apportees dans le noyau, ce qui offre une transparence maximale mais dans ce cas, on dispose d'un outil non portable et lourd à implementer dans un environnement hétérogène [MIL 851 [YANG 881.

2. Modification des primitives:

Pour resoudre le problème de portabilité, on évite d'affecter le noyau en apportant les modifications au niveau des primitives tout en gardant des versions non modifiées. L'inconvenient de cette approche est que le programmeur doit connaitre les details d'implementation des primitives afin de pouvoir porter ses modifications: or l'existence même des primitives est dans le but de soulager l'utilisateur de ces details [HAD 883.

3. Modification du code source:

Les modifications du code source sont acceptables dans le cas où elles sont transparentes à l'utilisateur et où elles preservent la sémantique du programme.

I.2.2 Introduction à la mise au point

Ce domaine est encore peu formalisé dans l'environnement distribué. 11 y a deux approches possibles [LEU 891:

1) Une approche statique: totalement indépendante de l'execution du programme. Elle est basée essentiellement sur les informations issues de la compilation. Elle permet de détecter des blocaaes ou des conflits d'accés aux variables. Le programme est convert i en un graphe de flôts de contrôle qui permet essentiellement de détecter deux classes d'erreurs:

- erreurs de synchronisation (boucle, blocaqe)
- erreurs sur les données (non initialisées, accés concurrents).

Néanmoins ces outils ne sont pas suffisants pour la mise au point de programmes distribués, puisqu'ils ne peuvent détecter des problémes liés à la synchronisation ou la competitions des processus ou autres problémes engendrés à l'exécution.

2) l'approche dynamique qui utilise des informations résultant de l'execution du prouramme: Elle nécessite souvent la sauveaarde de l'état du programme à intervalles réguliers. Pour mettre en evidence les problèmes des programmes étudiés, deux types de techniques sont à distinguer:

a) les techniques basées sur une interaction directe avec le programme, permettant par exemple, à l'utilisateur de stopper l'execution en plaçant des points d'arrêt. La mise au point consiste à relancer plusieurs fois l'execution afin de cerner le problème.

b)les techniques basées sur une reexecution de programmes. Elles reposent sur la récolte d'une quantité d'informations bien spécifique lors de l'execution initiale. Ces informations sont utilisées lors de la reexecution pour reproduire fidèlement l'execution initiale. L'interaction se fera sur la reexecution et non pas sur l'execution initiale.

Remarquons que ces techniques sont applicables seulement à certains types d'applications (la reexecution, par exemple ne s'applique pas aux programmes qui font des entrees sorties). La méthode idéale serait de combiner les deux techniques.

CHAPITRE II

TECHNIQUES DE DESCRIPTION FORMELLE D'UN PROGRAMME DISTRIBUE

Les problèmes introduits par la repartition, induits par l'asynchronisme et le non determinisme, necessitent un formalisme approprid pour décrire de maniere claire et lisible des programmes distribués. L'étape premiere de mise en oeuvre d'une telle méthodologie est de specifier de façon formelle un comportement distribué à la fois détaché des contraintes propres aux machines ou réseaux, et en même temps, pratiquement utilisable sur des machines réelles. De nombreuses techniques de description formelle ont été développées [DEM 87][DIA 891. Les techniques de description formelle sont des outils importants pour la conception, l'analyse et la specification de systèmes; car elles fournissent des descriptions concises, completes, consistantes, précises et non ambiques [ISO$_1$ 89][ISO$_2$ 89]. Une bonne technique de description formelle doit satisfaire, en general, un ensemble de criteres [ISO2 891, parmi lesquels:

1) décrire le systbme de manière complete et precise en incluant les comportements communicants et la prise de decisions globales et locales,

2) offrir certaines abstractions qui reduisent la complexité de la description,

3) faciliter l'analyse formelle et l'évaluation de la description,

4) être totalement independante des méthodes d'implémentation et de mise en oeuvre.

5) et offrir la possibilité de générer automat isuement du code parallèle à partir d'une description.

Estelle (extended state transition language), et LOTOS (Language Of Temporal Orderina Specification) sont deux techniques développées par le qroupe F.D.T (Formal Description Technique) à l'ISO ("International Standard Organisation"), qui ont été toutes les deux normalisées.

Estelle repose sur un modèle de machine à états étendu oh la communication s'effectue à travers des files d'attente FIFO (First In First Out) et represente les donnees en utilisant le langage Pascal. LOTOS [FAN 88] repose sur le formalisme ccs [MIL 80] et represente les donnees sous la forme de types abstraits algébriques [DIA 89].

II.1 Presentation des techniques de description formelle

A - LOTOS:

Il est base essentiellement sur les quatre concepts suivants:

1- l'interaction qui est considérée comme une activité commune entre deux ou plusieurs processus. C'est aussi une forme de synchronisation basée sur la verification de valeurs de donnees et une generation non déterministe d'informations.

2- le principe d'ordonnancement temporel : Le comportement externe d'un processus est defini par l'ordonnancement temporel de ses interactions avec son environnement.
LOTOS spécifie un processus par une "expression de comportement" qui definit un ordre d'événements: un événement étant une instance d'interaction. Il fournit un ensemble d'operateurs qui permettent de structurer et combiner des expressions de comportements de différentes façons. La nature algébrique du langage est mise en apparence par la disponibilite d'un ensemble de règles de transformation qui permettent de transformer une expression de comportement en une autre avec le même comportement observationnel.

3- l'abstraction de processus: un processus correspond approximativement au concept de procedure dans les langages conventionnels. Un processus formel est defini avec des points d'intéraction formels et des paramètres formels. L'instanciation de processus permet la structuration de la specification.

4- les types de donnees abstraits: l'abstraction de type donne au programmeur la possibilité de créer de nouveaux types de

données. La **sémantique** d'un type abstrait est **définie** en **spécifiant le** sens de chacune des operations en termes d'un autre modele **appelé** modele de reference.

Exemple : **type** extended-natural number is
sorts nat
opns **0** : -> nat
succ : nat -> *nat*
_+_ : nat, nat -> nat

eqns for all **x,y** : nat
of sort nat
x + 0 =x
x+ succ(y) = succ(x+y) ;
endtype.

LOTOS permet d'exprimer un comportement à différents niveaux d'abstraction. Un système distribué est décrit en LOTOS par une hidrarchie ***de processus.*** Ces processus effectuent leurs communications au moyen d'interactions **échangées à** travers des points d'interactions ou ports de synchronisation.

LOTOS ***a*** pour objectif **d'être** abstrait afin de n'impliquer aucune implementation particuliere dans les **systèmes** reels, et d'offrir des **mécanismes précis** et puissants de description du **parallélisme.** L e parallélisme dans LOTOS s'exprime au moyen d'une extension du formalisme CCS.

B- **Estelle:**

Estelle est un modele d'automates communicants étendu par :

- une représentation des données grâce AU langage PASCAL.

la possibilité d e structuration ***du*** programme en composantes communicantes séquentielles, appelées modules. permettant d'exprimer toutes les hierarchies à composantes **parallèlles,**

- et par la manipulation dynamique de ses composantes , de leurs structuration et de leurs liens de communication.

Estelle présente un degré d'abstraction moins important que celui de LOTOS lui permettant ainsi, de satisfaire le cinquieme critere cite précédemment. Estelle étant la FDT adoptée pour specifier les programmes du systbme d'expérimentation, une description plus détaillée en est présentée plus loin.

Estelle permet de définir des comportements parallels synchrones ou totalement asynchrones.

Ces deux techniques (Estelle et LOTOS) ont été developpees simultanement et aucune n'est privilégiée par rapport à l'autre dans la communaute de l'ISO. Cependant, il existe des differences remarquables entre ces deux FDTs.

1. La communication est synchrone dans LOTOS ne nécessitant guére d'acauittements et asynchrone dans ESTELLE avec file non bornée. Cette caractéristique rend certains probl Pmes plus simples à exprimer dans l'un plutôt que dans l'autre. Le problème de "backpressure" en est un exemple [FAN 881.

2. Les liens de communication sont bipoints dans Estelle, par contre, LOTOS permet l e "multicasting" offrant ainsi la possibilité de synchroniser plusieurs processus. Dans Estelle , ceci est émulé mais, la synchronisation n'est pas respectée.

3. La notion de temps est inéxistante dans LOTOS. Elle est exprimée dans Estelle grace à la clause 'Delay'. Cette notion est importante dans, par exemple, les protocoles de communication pour exprimer un "time-out" ou encore les systèmes distribués temps reel.

IT.2 CHOTX D'UNE TECHNIQUE DE DESCRIPTION FORMELLE

Pour le choix d'une technique de description formelle un parambtre fondamental est le niveau d'abstraction voulu pour une specification à établir. De maniere générale, plus ce n i veau est élevé et la specification complete et concise, plus la difficult6 d'ensendrer de façon automatique des réalisations sera importante. Or, le but final d'une specification etant la réalisation, le passage d'une specification à son implementation doit être un critere important pour le choix du langage d'implementation.

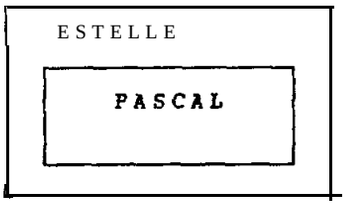
Ainsi, les possibilités de validation et d'implémentation automatique constituent les principales raisons pour lesquelles les approches imperatives sont actuellement largement developpees et utilisées. Les approches fonctionnelles actuellement en cours de développement semblent tout a fait prometteuses mais n'ont pas encore accede au même niveau de résultats [DIA 89].

Le langage Estelle satisfait à des degrés plus ou moins importants aux criteres d'une FDT satisfaisante tout en offrant un niveau d'abstraction permettant la generation automatique de code reparti.

CHAPITRE III

DESCRIPTION DU LANGAGE ESTELLE

Estelle est un langage base sur le modéle de transitions d'états étendu par le langage Pascal. Il inclut le langage PASCAL et l'encapsule dans des elements d'expressions de comportements paralleles.



Une specification Estelle est une hiérarchie de composants communicants sequentiels appelés modules et qui s'échangent des interactions à travers des points d'intéraction relies par un canal de communication (figure 111.1). Tous les objets Estelle sont fortement typés. La notion de typage de PASCAL est étendue aux objets ESTELLE. On parle alors d'un type de canal, de comportement. et d'un type de module.

Specification Estelle

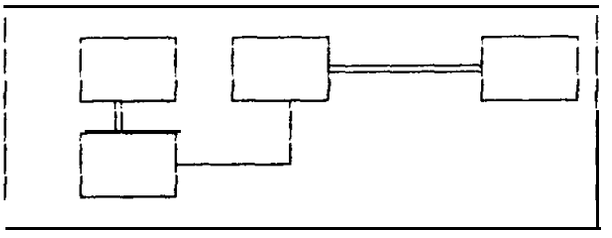


Figure III.1 : Structure générale d'une specification Estelle

III.1 Concepts de base

III.1.1 Structure d'un module

Un module est décrit par

- un interface de communication avec son environnement qu'on appelle en-tête du module,
- et par son comportement interne appelé corps du module.

Syntaxe:

```
MODULE typ_entête ( list. para)
 ip       < liste de ports >
 export   < liste variables exportées>
end:
```

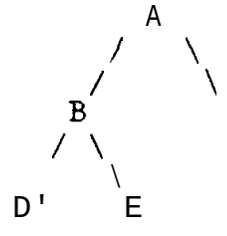
A une même entête de module, on peut associer plusieurs corps. Un module est donc un couple (entête, body).

```
BODY typ_b   FOR   typ_entete;
BODY typ_c   FOR   typ_entete;
```

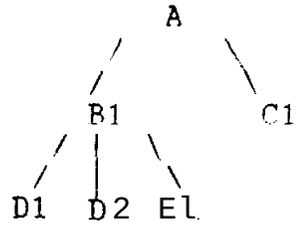
La definition d'un module peut comprendre des definitions d'autres modules (modules fils). On obtient ainsi un arbre hierarchique de definitions de modules dont la racine est un module particulier: la specification. Cet arbre des definitions sera un modèle pour l'arbre hierarchique d'execution (figure:III.2). Une instance d'un module est une execution du module. On peut oar exemple. avoir deux instances pour la definition d'un même module. L'arbre d'execution peut évoluer dvnamiquement dans le temps.

Les échanges d'interactions entre modules se font via des ports ou points d'interaction. Chaque point d'interaction est type par un canal.

a) arbre de definition



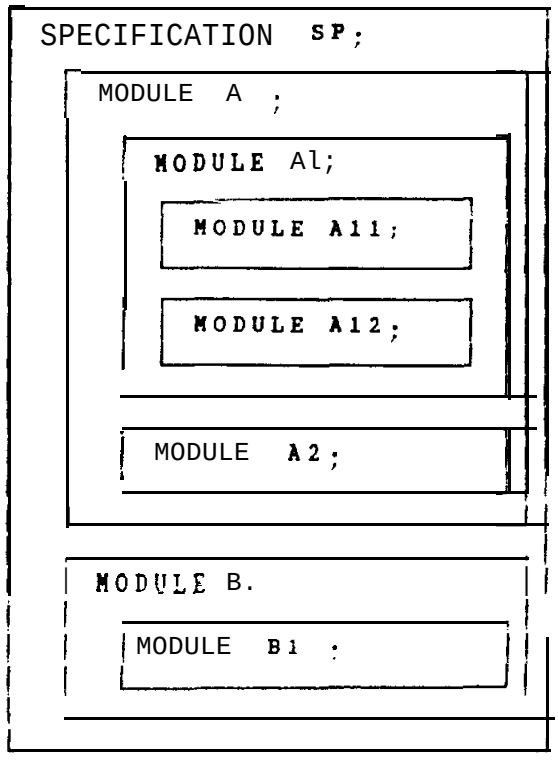
b) arbre d'exécution



où D1,D2 sont des instances de D et El une instance de E

Figure III.2 Arbre d'execution et arbre de definition

Exemple: Représentation de la structure de définitions de modules



Dans l'entête du module on trouve aussi la déclaration de variables export&es. Ces variables sont des variables partagées qui peuvent être accédées par le module père en lecture/écriture et dont la declaration est précédée par le mot clé export.

Le comportement interne d'un module est défini dans la partie "body". Il est donné par un automate à transition d'états défini par:

- un ensemble d'états de contrôle,
- et un ensemble de transitions.

En fait, le corps du module est compose de trois parties:

1) Partie declaration:

Cette partie comporte des declarations Pascal (de variables et de constantes) et des declarations d'objets Estelle (déclaration de canaux, déclaration de sous-modules, declaration d'états de contrôle,

declaration de points d'interaction et de variables module). Les variables modules servent de reference aux **instances des modules.**

2) Partie initialisation:

L'initialisation **spécifie les** valeurs initiales de certaines variables de l'instance **créée.** Elle comporte l'initialisation de

- **l'état** de contrble par la clause TO,
- certaines variables par l'affectation,
  la creation d'instances filles **par** l'initialisation des variables modules,
  et, **l'établissement** de liens de communication **par** des instructions de connexion entre points d'interaction.

Exemple:

```
initialize
   to     <etat_initial>
   begin
          <initialisation de variables
              et etablissement de liens
                   de communication >
   end:
```

3) Partie transition:

C'est un bloc qui comporte un ensemble de transitions. Chaque transition est **constituée** d'une liste de aardes qui **représente** la part.ie condition de la transition et d'une liste d'actions.

```
TRANS
     <condition>
     <action>
```

Un module dont le bloc transition est vide est dit **passif** autrement il est dit **actif.**

La condition de la transition est exprimde à l'aide d'une ou plusieurs des clauses suivantes

- * **from** Al, A2, . . . ,An **où les Ai** sont des états de contrôle. Cette clause exprime la condition sur l'état de contrôle courant.

* when P.m où P est un point d' interaction et m une interaction. Cette clause vérifie l'éxistance de l'intéraction m en tête de file du point d' interaction P.

* Provided B: cette clause permet d'exprimer une condition sur le contenu d'une interaction ou sur les variables locales de la tâche.

* **Priority** n : exprime la priorité de la transition par rapport aux autres transitions de la même tâche.

* Delay (e1, e2) oil e1 et e2 sont des expressions entières non negatives. Si la transition est satisfaite à l'instant t, elle sera retardée de el unites de temps au moins et e2 unités de temps au plus.

* Any : permettant le regroupement de plusieurs transitions en une seule: la métatransition, tel que l'exemple suivant:

```
trans  from S1 to S2
       any n:1..2 do
       when p[n].m
       begin  v:=n  end:
```

Cette métatransition est équivalente à

```
trans  from S1 to S2
       when p[1] .m
       begin  v:=l  end:

trans  from S1 to S2
       when p[2] .m
       begin  v:=2  end;
```

Certaines clauses peuvent être omises et au plus une de chaque catégorie peut apparaître dans la condition d'une simple transition. De plus, la presence d'une clause when exclut celle d'une clause delay et vice versa.

L'action d'une transition est composée:

* d'une clause TO A où A est un état de contrôle ou le mot clé same. Cette clause indiaue l'état prochain au'atteindra l'automate après execution de la transition. Le mot clé **same** indique ou'il n'v aura pas de changement d'état après tir de la transition.

* et d'un bloc PASCAL étendu par des instructions ESTELLE. Ce bloc peut être precede par des declarations d'objets locaux à la transition.

L'execution d'une transition par une instance de module est considérée comme une operation atomique qui ne peut être exécutée que si elle est satisfaite.

Dans la partie action d'une transition, on retrouve des instructions Pascal avec certaines restrictions et des instructions Estelle.

1) Instruction 'init'

Cette instruction est utilisee pour créer des instances de modules fils du module initiateur.

syntaxe:

init module-var with **body_idf**
init module var with body idf (<act mod para>)

module_var est une variable module. Cette variable permettra de référencer la nouvelle instance créée. Une variable module doit être declarée d'un certain type de module.

11 peut y avoir plusieurs corps de modules associés à la même entête de module. L'instruction 'init' les différencie en specifiant le corps du module (body idf).

l'instruction 'init' peut être utilisee dans la partie initialisation ou dans la partie transition permettant respectivement la creation statique et dynamique d'instances de modules.

Lorsqu'on assiqne la mêmevariable module a deux instances différentes. la premiere devient inaccessible.

Exemple

```
init X with B:
init X with B:
```

On peut initialiser une variable module par l'instruction d'affectation exemple:

```
init  X with B;
 Y:= x:
INIT X with B:
```

2) Instruction **'Connect'**

svntaxe

```
connect   internal ip          to child external ip
connect   child external ip    to child external ip
connect   internal ip          to internal ip
connect   child-external ip    to internal ip
```

Un point d'interaction est dit 'external' s'il est declare dans l'entête du module. 11 est dit **'internal'** s'il est declare dans le corps du module. Les deux points d'interaction indiqués par cette instruction doivent référencer le même canal et jouer des roles opposes. Cette connexion établit un lien de communication à travers lequel les deux points d'interaction references pourront échanger des interactions.

Deux contraintes a respecter:

a) un point d'interaction interne ne peut être connect.6 qu'à un point d'intéraction de même type de canal et jouant des roles opposes,

b) un point d'interaction est connecté à au plus un point d'interaction et jamais à lui-même.

Notons que la connexion n'a aucun effet sur les files d'attente associées.

31 Instruction **'disconnect'**

syntaxe:

```
disconnect internal_ip
disconnect child external ip
disconnect module_var
```

Dans les deux premieres formes il suffit d'indiquer un seul point d'interaction; le second est connu implicitement.

Dans sa dernière forme, l'instruction s'applique a tous les points d'interaction **"external"** de **son** module fils. Cette instruction n'a aucun effet sur les files d'attente.

4) instruction **'attach'**

svntaxe

```
attach external_ip to child_external_ip
```

Cette instruction attache les deux points d'interaction specifies. Le premier doit être un point d'interaction "external" du module appelant et le second doit être un point d'interaction "external" d'un de ses modules fils. Les deux points d'interaction désignés par l'instruction doivent référencer le même canal et jouer le mêmerôle. Et, elle retire *toutes les* interactions de la file associée au premier point d'interaction. Les interactions retirees de cette file sont associées :

a/ au deuxieme point d'interaction (child) s'il n'est pas attache à l'un de ses fils:

b/ sinon au point d'interaction du dernier fils: point terminal.

En résumé:

* Un point d'interaction "external" d'un module est attache à au plus un point d'intbraction "external" du module pére et au plus à un point d'interaction "external" du module fils.

* Un point d'interaction "external" d'un module attache a un point d'interaction "external" du module pére ne peut être connecté.

5) instruction detach

syntaxe

detach external ip
detach child external ip
detach module variable

Cette instruction détache les points d'interaction attaches.

Dans la troisiéme forme, elle detache tous les points d'interaction attaches du module fils. De plus, toutes les interactions sont remises dans la file d'attente du module appelant cette instruction.

6) instruction ALL

Elle sert à généraliser une operation sur un ensemble d'objets.

syntaxe

ALL domaine DO instructions

Lorsque cette instruction est utilisee pour des instances de modules, le domaine declare une variable module en specifiant son type comme suit:

```
var_mod: idf_entete_module
```

7) instruction **output**

syntaxe

```
output  Pi.idf_intéraction (<parametres>)
```

Elle permet l'envoi d'interactions idf_intéraction via le point d'interaction Pi.

III.2 **Communication et connexion**

Estelle, offre deux mecanismes de cooperation entre instances

- l'échange d'interactions,
- et le partaqe de variables.

Les variables introduites par le mot clé "export" dans l'entête d'un module sont. partagees avec l'instance parente. Le conflit d'accès a ces variables est impossible puisque l'instance du module père est prioritaire par rapport a celles de ses descendants selon le principe de priorité père/fils d'Estelle .

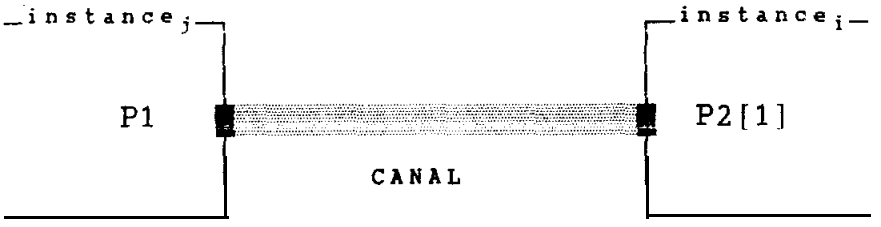
Le mecanisme d'echange d'interactions nécessite l'établissement de liens de communication au prealable. En Estelle les connexions sont bipoints. L'échange se fait via des points d'interaction relies par un canal de communication specifiant l'ensemble des interactions qui peuvent circuler sur celui-ci dans un sens ou dans l'autre comme suit:

```
channel canal-id ( role1,role2);
    by  role1          :  ml:  m2(i:integer);
    by  role2
    by  role1, role2 : kl:
```

Une interaction reçue par une instance est chaînée a la file d'attente FIFO du point d'intéraction destinataire. Cette file peut appartenir exclusivement 4 ce point d'interaction et elle est alors qualifiée de **individual-queue** où partagee par d'autres points d'interaction de l'instance et dans ce cas, il s'agit d'une **common-queue.**

La declaration des points d'intéraction est introduite par le mot clé IP. Elle precise le type du point d'interaction - qui est un type de canal- et le role qu'il doit jouer. On spécifie aussi la discipline de sa file d'attente.

```
IP p1: canal-id (role-id); queue-discipline;
   p2: array[1..5] of canal_id(role-id):
```



III.2 Parallèlisme et non determinisme

Le comportement d'un module relativement aux autres modules (execution parallèle et non déterminisme) depend strictement de l'imbrication du texte de definition du module et de la manière avec laquelle l'entête est qualifiée ou attribuée. Il existe quatre classes de modules:

- "systemprocess",
- "systemactivity",
- "process",
- et, "activity".

Dans une hiérarchie de processus, les principes d'attribution suivants doivent être observes:

1) Chaque module actif doit être attribue.

2) Un module système (c'est à dire "systemprocess" ou "systemactivity") ne peut être inclu dans un module attribue.

3) Les modules attribues "process" ou "activity" doivent être dans un module système.

4) Les modules attribues "process" ou "systemprocess" peuvent être structures en modules attribues "activity" ou "process".

5) Les modules attribues "systemactivity" ou "activity" ne peuvent être structures qu'en modules "activity".

Seuls les instances de modules **actifs** peuvent dynamiquement **créer**, detruire et changer la configuration des connexions d'instances de modules fils. Une instance active agit donc comme un superviseur pour ses instances descendant-es. Ceci est l'une des raisons de la priorite père/fils. Cette priorite signifie que le **père synchronise** l'execution de ses descendants. Ce qui fait *que* lors de l'execution d'une transition d'un module, toute execution de transition d'un module fils est snspendue. Cette priorite est transitive et elle exclut tout parallelisme entre instances **liées** par une relation **ancêtre/descendants.**

Les modules **systèmes** (c'est a dire "systemprocess" ou "systemactivity") ne sont jamais **inclus** dans un module **actif. Donc,** ils n'obeissent pas **à** la priorite d'un module **père.** Ainsi, les modules **systèmes** identifient des **systèmes** dont les comportements sont totalement asynchrones et qui communiquent par interactions.

Les modules "systemprocess" et "systemactivity" se differencient **par** leur **comportement** interne. Le parallelisme dans un systeme; c'est a dire entre instances de modules descendants d'un **même** module systeme, est determine par

- l'attribut **assigné** "process" ou "activity",
- et. par la priorité père/fils.

L'attrihut "process" spécifie une execution paralléle svnchrone alors que l'attribut "activitv" spécifie une execution séquentielle non **déterministe.**

III.4 **Comportement** d'une specification Estelle

111.4.1. **Module_spécification**

Le module principal englobant tous les autres modules **est appelé** module specification et il existe toujours une instance unique de ce module particulier.

```
specification spes_name [system-class]
[default option] [time option]
body_def
end.
```

oh **system_class** peut Ctre **"system activity"** ou "system process.

default option: type de files d'attente a assigner par défaut aux points d'interaction.

time option : definit l'unité de temps à utiliser dans la specification (sec. msec, ..etc).

En pratique, le corps de la specification fournit un contexte global necessaire à la definition et l'initialisation d'un système.

Exemple de spécification:

```
spécification exemple;

 channel canal (entree, sortie):
  by sortie: data-indication;

 channel chan (entree,sortie);
  by entree: data-indication:
  by sortie: ack (x:integer);

 Module user systemactivity;
  ip U:canal(entree);

 end:

 Body user-body For user;external;

 Module Receiver systemactivity;
  ip U: canal(sortie);
  ip N: chan (sortie);
 end;

Body receiver-body For receiver;external;

Modvar X:user; Y: receiver:

initialize
begin
        init X with user-body:
        init Y with receiver body:
        connect X.U to Y.U;
end:
 end.
```

III.4.2. Comportement global

Le comportement global est defini par l'ensemble des sequences possibles de chanqement d'états qui correspondent à l'execution de transitions:

Une specification Estelle décrit un ensemble de systemes dont les modules peuvent executer leurs transitions de maniere non deterministe. svnchrone ou asvnchrone.

Les systemes s'executent en une succession d'étapes de calcul. Chacune commence par la selection d'une seule transition (systemactivity) ou plusieurs (systemprocess) parmi les transitions satisfaites. Les transitions sélectionnées dans ce dernier cas sont exécutées en parallèle. L'étape de calcul se termine lorsque toutes ses transitions se terminent. Pour celà, on parle de **parallélisme synchrone.**

Notons cependant, que puisqu' une seule transition est sélectionnée de maniere non deterministe pour chaque étape de calcul d'un systemactivity, on a un comportement purement non deterministe dans chaque systeme.

Au contraire, les systbmes operant de maniere asynchrone sont completement independant l'un de l'autre. La vitesse relative des systemes n'est pas du tout contraiunante.

L'execution d'une transition

- peut changer l'état local d'un module. En particulier, elle peut modifier une variable locale ou un état de contrôle. elle peut créer des modules fils et de nouveaux liens,

- ne peut pas influencer ni le choix d'une transition présélectionnée par les autres systemes. ni le choix des transitions dans le même systeme lors d'une même étape de calcul.

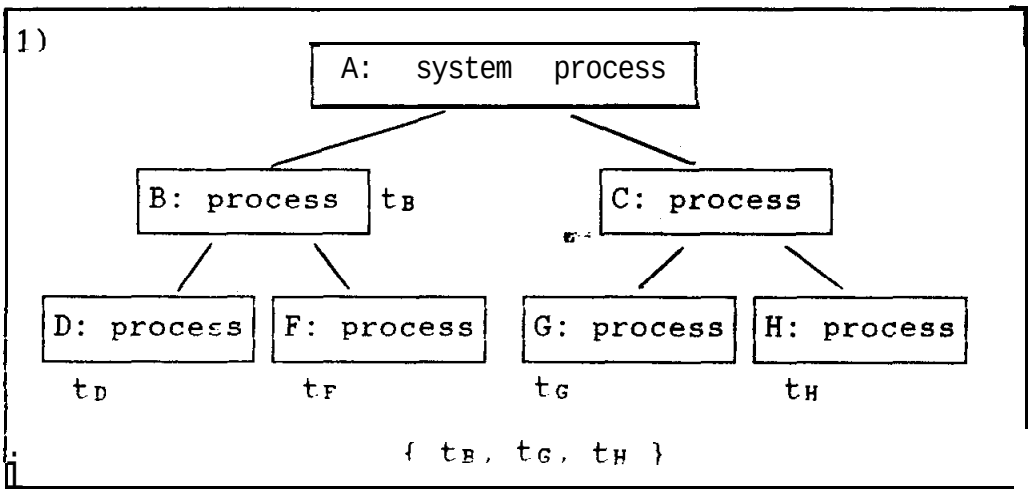
La selection d'une transition est réglée par:

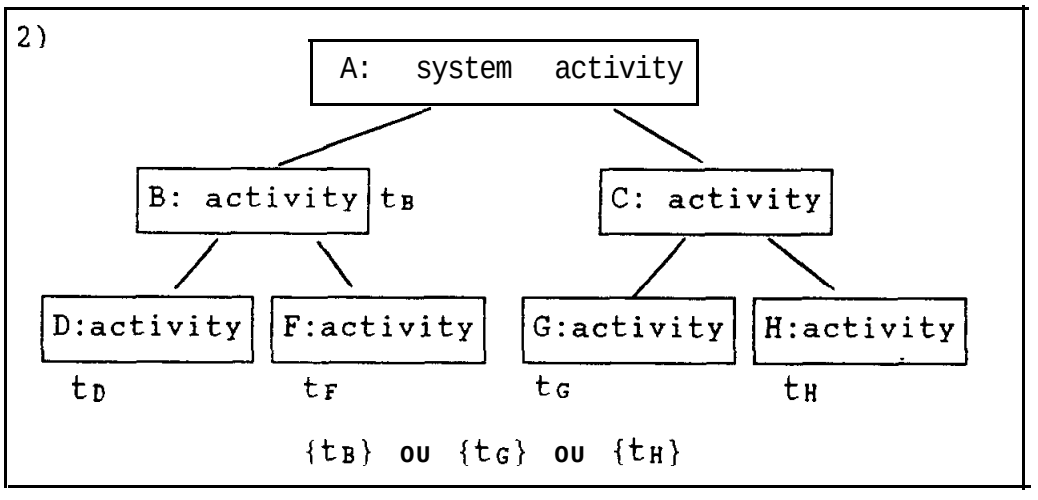
- le principe de priorite pére/fils.

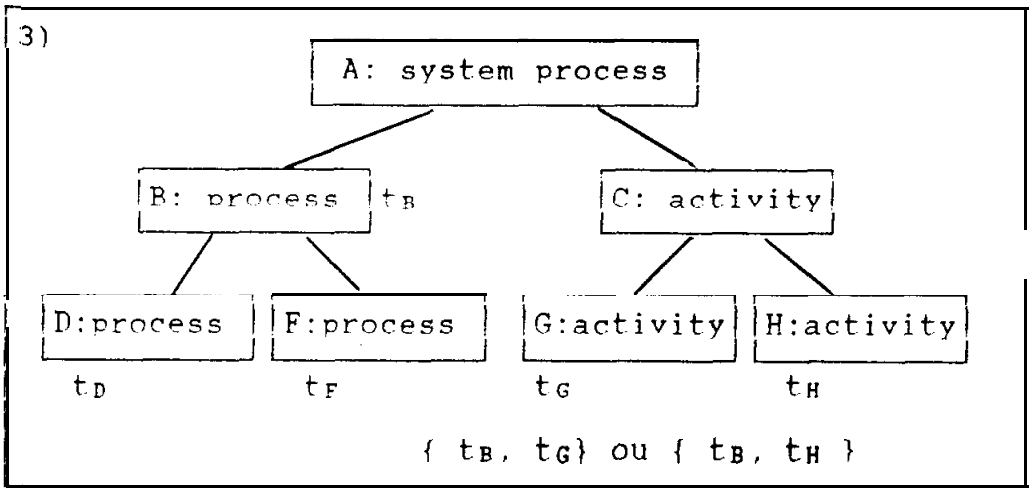
- l'attribut du module.

La selection d'une transition d'un module interdit la selection d'une transition d'un module fils; cette priorité exclut le parallélisme entre père/fils (de maniere générale ancêtres/descendants).

Pour illustrer les différents comportements des modules, on propose les différents cas de figures:


1)
A: system process
B: process
t_B
C: process
D: process
F: process
G: process
H: process
t_D
t_F
t_G
t_H
{ t_B, t_G, t_H }


2)
A: system activity
B: activity
t_B
C: activity
D:activity
F:activity
G:activity
H:activity
t_D
t_F
t_G
t_H
{t_B} OU {t_G} OU {t_H}


3)
A: system process
B: process
t_B
C: activity
D:process
F:process
G:activity
H:activity
t_D
t_F
t_G
t_H
{ t_B, t_G} ou { t_B, t_H }

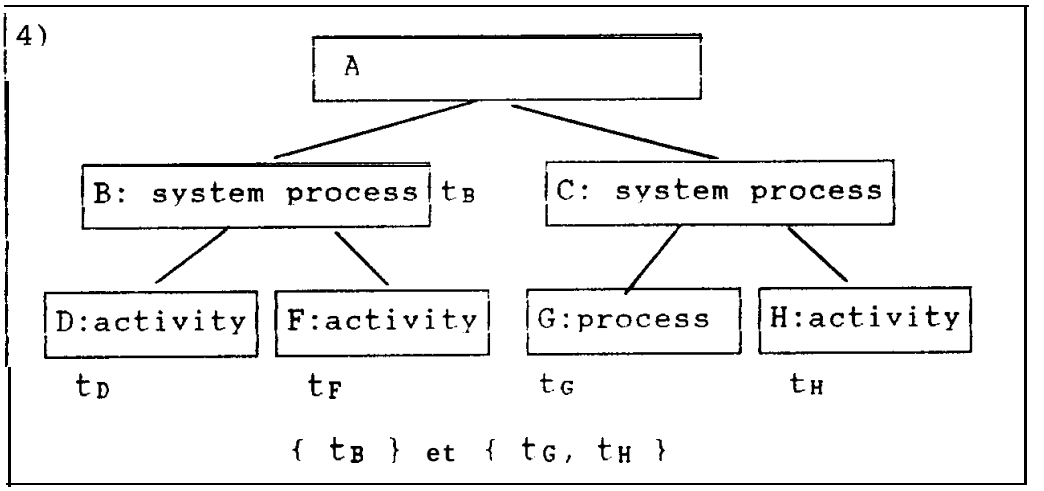


Figure III.3 Comportement global d'une specification

III.5 **Restrictions Pascal**

Certaines restrictions Pascal ont été introduites dans le langage Estelle normalisé par l'ISO. Toutes les operations sur les fichiers sont interdites ainsi que le mot reserve "program". L'utilisation du aoto a été sévèrement restreinte: il n'est jamais utilisé dans une transition et est interprété tel un "return" dans les procedures et fonctions.

Les fonctions utilisées doivent être purement demontrahles, c'est à dire qu'elles ne peuvent modifier ni directement ni indirectement les variables non locales.

La syntaxe des fonctions ne permet pas de retourner des donnees de types complexes. Cependant. Estelle permet de préciser que la definition d'une procedure, fonction, ou module est donnée ailleurs. Ceci est indiqué par le mot reserve "external".

Dans Estelle, les pointeurs ne peuvent pas être utilisés comme parametres d'interactions ou de modules ni de fonctions ou procedures.

Les constructions Estelle ne peuvent pas être utilisées dans des procedures ou des fonctions.

III.6 Exemple de spécification Estelle

L'exemple que nous allons traiter est l'alsorithme d'élection proposé par Chandy et Roberts [CHA 791.

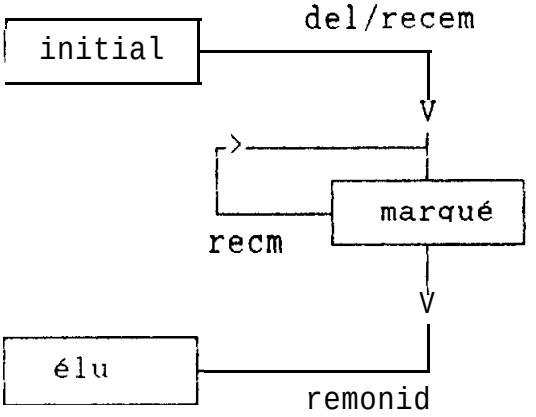
Les processus sont placés sur un anneau unidirectionnel. Chacun des processus Pi est dote d'un numero qui l'identifie de manibre unique.

Principe:

11 s'agit de trouver le maximum des identités. Pour cela, chaque processus Pi qui connait son numéro i, le transmet à son voisin. A la reception d'un tel message, celui-ci compare son numero a celui reçu et transmet le plus grand à son voisin.

Initialement, un ou plusieurs processus commencent l'election en emettant un message. Un tel processus est dit marqué. L'arrivee d'un message à un processus à l'état initial le fait passer à l'état marqué. Le processus marqué recevant sa propre identite sera l'élu.

La specification est represent&e par l'automate d'états fini suivant:



recm: reception de message du voisin
del : decision de lancer l'election
remonid: reception de mon identite.

Algorithme:

```
specification    roberts;

type nbsite= 0..10; nb_site0= 1..10;

channel canal (ext1,ext2);
by extl: message (élu:integer);
by ext2: ;

Module t_site process(moi:integer);
ip  entree: canal(ext2);
   sortie: canal(ext1);
end;

Body site for t site;
   state élu, marque, initial:
   var  voisin : integer:

   initialize to initial begin end;
    trans from initial to marque
           begin  output sortie.message(moi) end:

    trans from initial, marque to marqué
          when entrée.message(voisin)
          begin
            if voisin > moi then
                  output sortie.message(voisin)
              else output sortie.message(moi)
          end;

    trans from marque to élu
          when entrée.message(voisin)
          provided voisin= moi
          begin   (* je suis Clu *)
          end:
end;

nodvar sites :array[nb site1 of t site:

initialize
   begin
      all i:nbsite do init sites[i] with site(i);
      all i:nb site0 do
      connect site[i-1].sortie to sites[i].entrée;

      connect sites[0].entrée to sites[10].sortie;
   end:
end.
```

CONCLUSION

Specifier des aromrammes en Estelle permet de décrire un comportement à caractbre distribué en un seul programme de manière très souple; car, les proqrammes distribues se présentent souvent sous forme de transitions d'états.

Les recherches actuelles autour des problèmes tels que:

- la manipulation des descriptions formelles
- la verification des alqorithmes
- la mise au point
- la simulation

la generation de code pour les systèmes distribues,

utilisent des mdthodes qui s'appliquent au langage Estelle ou autres lansames internationaux normalises.

Divers outils ont été développés dans ce sens. Parmi ceux aui tournent autour d'Estelle nous citons:

- ECHIDNA [JEZ 891: qui offre un compilateur Estelle to C pour machines parallèles.

- VEDA [JAR 88] developpé pour la simulation, son langage de description @St- un sous ensemble d'Estelle.

- Projet SEDOS [VIS 891 a produit
  - un éditeur orienté syntaxe
  - un translateur
  - un noyau de simulation ESTIM [ANS 891.

Il est a signaler que divers compilateurs ont été réalisés. On cite, les travaux de G.W Gerber [VUO 881 qui a propose un compilateur Estelle-Pascal a l'université d e Montreal. et les travaux de Per Rerqsten à l'universite de Gothenburq en Suede qui propose un traducteur Estelle to C. Un traducteur Estelle to C a été aussi propose dans [VUO 88].

CHAPITRE IV

UN GENERATEUR DE CODE POUR SYSTEMES REPARTIS

Dans le but d'etudier les performances des algorithmes distribués spécifiés en langage formel, nous avons développé un générateur automatique de code distribué. Celui_ci accepte en entree une specification Estelle pour générer du code Pascal. Le code généré est ensuite lie au noyau d'exécution reparti pour obtenir un code executable réparti.

IV 1 Présentation de l'interpréteur

Théoriquement, un générateur de code doit satisfaire des exigences très strictes. Le code produit doit être correct et de bonne qualité. De plus le code produit doit Ctre optimal afin d'induire une execution rapide. Mathematiquement le problème de la production de code est indecidable. En pratique, on se contente de produire un code quasiment bon [AHO 891.

Une conception modulaire et l'utilisation de constructeur pour implanter l'analyseur lexico-syntaxique d'un compilateur permet d'integrer plus facilement les changements dans la definition syntaxique du lanqage et donc, d'apprehender 1 'evolution de celui-ci. On doit wrendre en compte avec le plus grand soin la nature et les limitations du lanaaae cible et de l'environnement d'exécution. car ils ont tous les deux une influence forte sur la conception de l'analyseur.

La syntaxe d'Estelle que nous avons implement&e est tirée de [ISO_2 891. Cette grammaire enrichie de routines sémantiques est soumise a l'analyseur syntaxique pour générer le code désiré. L'évaluation des routines incorporees à chaque production de la grammaire permet le contrôle de la cohérence de types. de sauvegarder de l'information nécessaire a la traduction. la generation de messages d'erreurs et la production de code.
L'analyseur syntaxique est lui-même un code généré par YACC [JOH 781. YACC est l'acronvme de << YET ANOTHER COMPILER-COMPILER>>. Il est disponible en tant qu'utilitaire sur le système UNIX et est utilisé pour faciliter l'implantation de compilateurs. Le programme généré est une implantation d'analvseur LALR ecrite en C, complétée par des routines utilisateur. Dans notre cas, elle représente le générateur de code distribué.

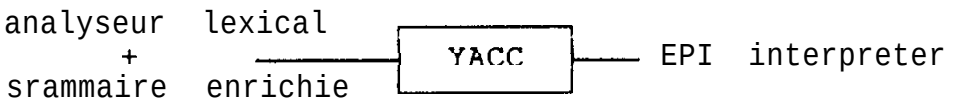


Figure IV.1: Construction de l'interpreteur

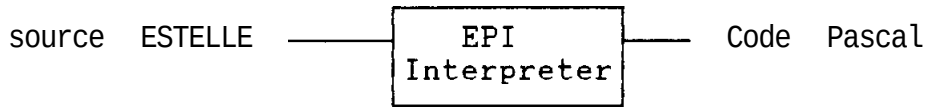


Figure IV.2: Generation de code distribué

IV 1.1 Fonctions de l'interpréteur

Une fonction de l'interpret.eur est d'enresistrer les donnees utilisees dans le programme source et de collecter de l'information sur divers attrihuts de chaque objet Estelle. Ces attributs vehiculent des connaissances sur, par exemple, le type de l'objet, sa portée, son niveau hierarchique s'il s'agit d'un module. et d'autres informations.

Une autre fonction est la detection et le sisnalement d'erreurs. Les erreurs detectahles à ce niveau sont surtout liées à la syntaxe et la semantique d'Estelle, comme par exemple, verifier que les deux extrémités d'un lien de communication Ctabli par l'instruction "Connect" sont de rôles opposes.

Une troisième fonction est la génération du code Pascal en respectant la sémantique du code source Estelle.

Afin de réaliser ces trois fonctions, l'utilisation de structures de données intermediaires s'impose pour conserver les informations necessaires à l'élaboration du code Pascal.

Dans ce gui suit, nous mettrons en evidence, pour les entités les plus importantes d'une specification Estelle, les structures élaborées.

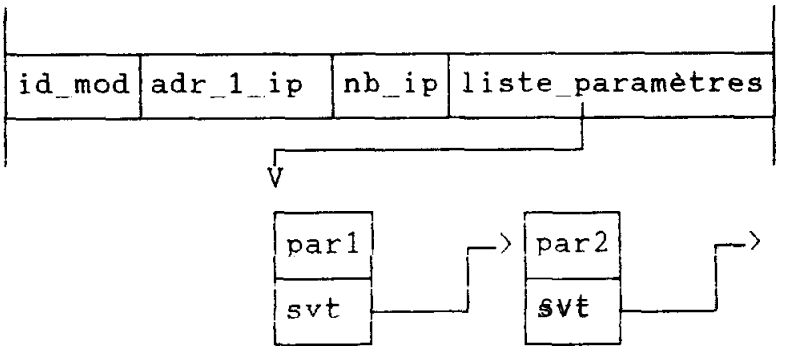
1. Entité module

Un module dans Estelle est constitud de deux parties: entête et corps.

a. Entête de-module

Representant essentiellement la visibilité externe d'un module, l'entête est constituée de declarations de points d'interaction et de parambtres de module. Ces informations, propres aux instances asseciees à cette entête, sont conservées dans la table de descripteurs de modules.

table des descripteurs de module

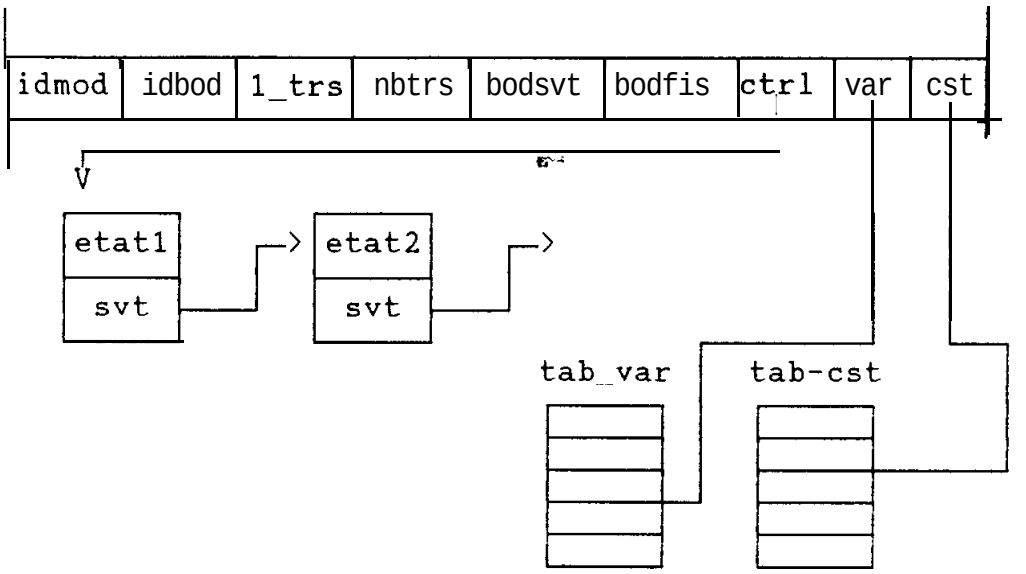


id mod: identificateur du module.
adr_1 ip: adresse du premier point d'interaction,
nb iv nombre de points d'interactions ,
liste paramètres: listes des paramètres du module ,
par1,par2,..: identificateurs des parametres,
svt : pointeur sur le parametre suivant.

b. Corps de module

Rappelons que plusieurs corps peuvent être associés à une même entête de module. C'est aussi dans cette partie qu'est définie la hiérarchie de module. Le corps ou 'body' définit le comportement d'un module, et comprend la partie initialisation des modules fils et la pat-tie transition. Nous rasssemblons donc, ces informations dans la table des descripteurs de 'body' en aardant les liens de parenté entre sous modules.

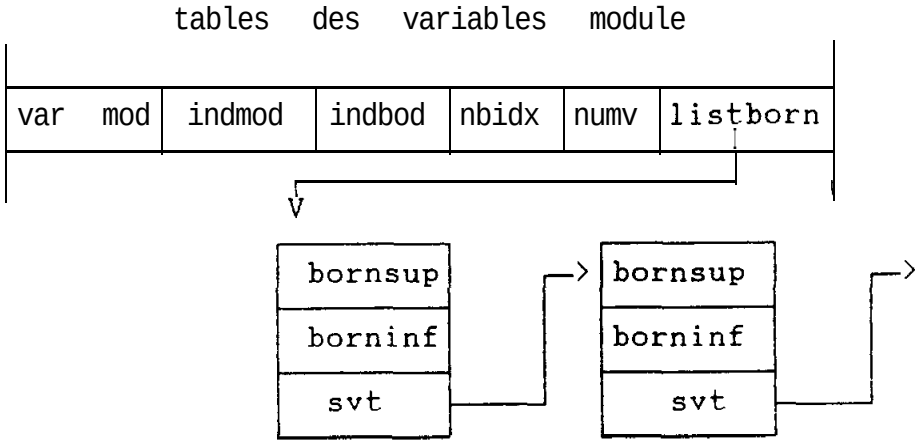
tables des descripteurs body



idmod: identité du module,
idbod: identificateur du body,
1 trs: numéro de la premiere transition,
nbtrs: nombre de transitions,
bodsvt: indice du body de même niveau hierarchique,
bodfis: indice du premier body fils,
ctrl: liste des états de contrôle,
var: adresse de la liste des variables dans tab-var,
cst : adresse de la liste des constantes dans tab_cst,
$etat_i$: identificateur de l'état de contrôle,
svt : pointeur vers l'état suivant.

tab_cst, tab-var, tab_typ sont, respectivement, les tables de constantes, de variables, et de types. Elles contiennent, entre autres, des informations telles que la valeur d'une constante, et la dimension du type. Les champs cst, et var servent a delimiter respectivement, le champ de visibilite des entites constante et variable.

Les variables module. variables spécif iques pour référencer les instances de modu1e sont aussi dotée d'un nombre d'informations utiles.

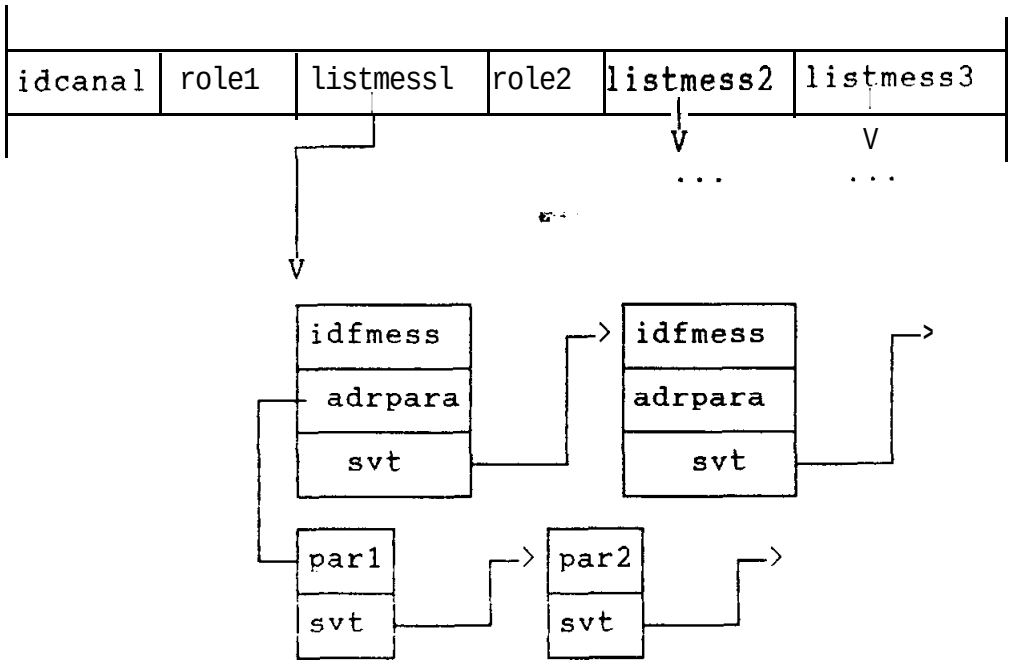


var mod: identificateur de la variable.
indmod : indice de l'entête du module dans la table module.
indbod : indice du corps du module dans la table bodv

nbidx : nombre d'indexes s'il s'agit de type tableau,
listborn: liste des bornes du tableaux,
numv : numero de sequence.

2. Entité canal

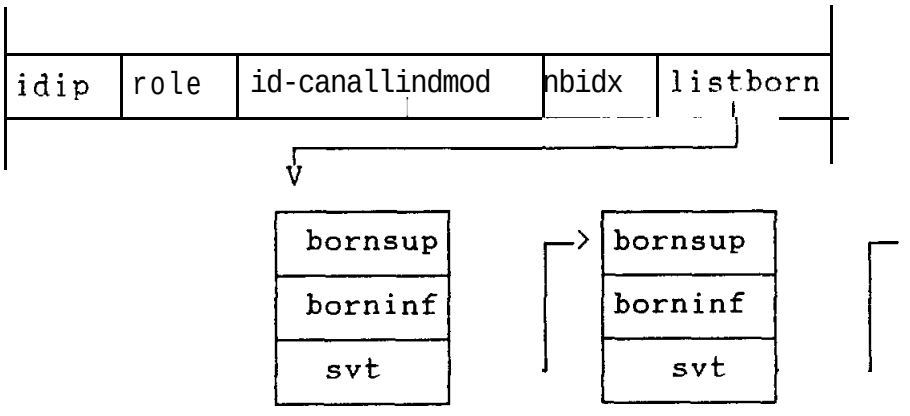
Le canal, objet Estelle, constitue le lien entre deux points d'intéraction. Il est identifié par un type et deux roles: rôle de l'extremite une et rôle de l'extremite deux. A chaque extrémité est associee une file de messages definissant le type des messages pouvant traverser le canal dans un sens, dans l'autre. ou dans les deux.



idcanal: identificateur du canal,
role1, role2: identificateurs des roles,
listmessl. listmess2. listmess3: listes des messages associés à un sens, au sens inverse ou les deux sens.
idfmess : identificateur du message,
adrpara : adresse de la liste des paramètres du messages,
pari: identificateurs des paramètres du message.

3. Entité point d'intéraction

Les points d'intéraction sont le seul moyen de communication entre instances. Ils sont de type canal et doivent oblisatoirement jouer un rôle. Ceci leur définit implicitement les messages qu'ils peuvent transmettre et recevoir.



idip : identificateur du point d'interaction,
role : identificateur du role du point d'interaction,
id-canal: identificateur du canal,
indmod : indice de l'entête du module dans la table modul e
nbidx : nombre d'indexes s'il s'agit de type tableau,
listhorn: liste des hornes du tableau,
bornsup,borninf:: bornes du tableau.

4. Imbrication de blocs

Lorsqu'on se trouve dans un cas de structure en blocs et surtout *en* imbrication de blocs, on se confronte obligatoirement au problème de localité et globalité d'objets. Les objets se trouvant dans des blocs différents et ayant le même nom posent un probleme d'ambiguite. Il faut disposer d'un moyen qui nous permet de distinguer entre eux en assignant chacun à son bloc proprietaire. Afin d'éviter toute ambiguité, nous prefixant chacun par le nom du bloc englobant. Mais, pour ce faire, il faut savoir a quel bloc l'objet appartient-il?. Ceci a été résolu en utilisant une pile dans laquelle on empile, à chaque entree dans un bloc, son identificateur. L'opération inverse se fait à la sortie. A tout instant, le sommet de pile indique le bloc courant. Ceci lors des déclarations. Lors d'une référence à l'objet, on fait int.ervenir les tables tab cst tab typ afin de retrouver son propriétaire. Rien sûr, ces ohiets sont assignés au bloc le plus interne parmi les blocs oh apparaissent 1 eur déclaration. Cette pile permet de contrôler, aussi, la portée d'un objet.

5. Traitement des procédures et fonctions

Dans Estelle, les fonctions et procedures sont soumises à un certains nomhre de contraintes. Elles ne peuvent faire reference aux obiets Estelle telles aue les variables module, les points d'interaction, les interactions et les états de contrôle. Donc, les instructions Estelle mettant en jeu ce type d'objets ne sont pas permises dans les procedures et fonctions. Ces contraintes preservent des effets de bord:

Exemple:

```
var   x:integer;
Function  A:boolean;
begin
      A:=false;   x:=x+10;
      if  x>=20  then  A:=true;
end;
  ...
trans  from  etatl
       provided (x=10) and A
       to  etat2
       begin . . . end:
```

Dans cet exemple la seconde condition de tir de la transition est aue x soit égal à 10: or, après l'appel de A, x prend la valeur 20, la fonction devient vraie et la transition est tirahle avec x=20 au lieu de x=10.

Ainsi, une nouvelle notion a été introduite: la notion de "pureté". Dans Estelle. toutes les fonctions doivent être pures par definition. Cette propriété obeit aux contraintes suivantes qui doivent être contrôlées par notre interpreteur:

- les paramètres ne doivent pas être de type pointeur,
- le passage de parametres doit se faire par valeur,
- la fonction ou procedure pure ne doit pas manipuler des variables globales.
- elles ne doivent pas contenir des appels à des fonctions ou procedures non pures.

IV 1.2 Interprétation Estelle Pascal

Tous les obiets Estelle sont interprétés par des instructions Pascal ou structures de données manipulables par ce langage. L'ensemble des obiets inclut:

- les instances,
- les points d'interaction,
- les interactions,
- les canaux de communication,
- et les transitions.

IV.1.2.1 Instructions Estelle

1) Création d'une instance

Les instances sont créées sur le site où elles doivent s'executer. A chacune est associé un espace de travail constitué d'un bloc d'informations de contrôle et d'un contexte propre. On y retrouve les informations de contrôle suivantes:

- le numero du site d'execution de l'instance,
- le nombre de points d'interaction,
- l'adresse du premier point d'interaction,
- le nombre de transitions,
- l'adressse de la premiere transition,
- la liste des sous_instances,
- et, l'état d'activité de l'instance.

Quant au contexte de l'instance, il comprend:

- la liste des parametres et variables de l'instance,
- son état de contrôle courant,
- et la liste des descripteurs de ses points d'interaction.

L'initialisation d'une instance se fait grace à l'instruction Estelle "INIT" dont la syntaxe est la suivante:

INIT idf mod var WITH idf bodv (<liste paramêtres du module>)

Cette instruction alloue l'espace nécessaire au bloc de contrôle et le met à jour. De plus, si l'instance doit s'executer sur le site local. elle alloue l'espace nécessaire au contexte et un descripteur pour chacun de ses points d'interaction. Mod_var est la variable de module adressant l'espace de travail alloué pour l'instance créée.
Sur tous les sites, on retrouve les blocs de contrôle de toutes les instances initialisées. Ceci permet d'accéder, plus facilement, aux informations de contrôle des instances distantes tel que le site de localisation.

2) Interconnexion

A l'initialisation d'une instance, un descripteur est associé à chacun de ses points d'interaction en lui affectant une identite unique. Ce descripteur contient:

- l'identité unique du point d'interaction,
- l'identitb du point d'interaction correspondant,
- l'identité du site du correspondant,
- l'adresse de l'instance à laquelle il appartient,
- les indicateurs de debut et de fin de la file d'attente des interactions,

Les descripteurs des points d'interaction des instances s'exécutant sur un site sont chaînés entre eux formant une liste globale de points d'interaction du site. Celle ci est parcourue à la réception d'intéractions distantes afin de retrouver le point d'intéraction destinataire designé par son identité. Les informations concernant un point d'interaction sont mises à jour à l'initialisation de l'instance. Les informations relatives au correspondant sont complétées lors de l'établissement d'un lien de communication. 11 existe deux types de liaison réalisées par les instructions Estelle "CONNECT" et "ATTACH".

Connexion

Une demande de connexion de deux points d'interaction P_1 et P_2:

CONNECT P_1 TO P_2

ne peut être satisfaite que

- si les deux points d'interaction référencent le même canal et jouent des rôles opposés,
- et, si aucun des deux points d'intéraction référencés n'a été déjà connecté.

Une connexion entrafne la mise à jour des identités des correspondants dans chacun des descripteurs des points d'interaction mis en jeu.

Les points d'interaction peuvent être locaux ou distants. Lors de l'interpretation, un appel à la procedure run_time "connect" est généré en faisant passer en parambtres les adresses des points d'interaction concern&s.

Raccordement

Une instance peut attacher ou raccorder un de ses points d'interaction à un point d'intbraction d'une instance de sous instance fille. A l'exécution de l'instruction

ATTACH P_i TO P_{i_fils}

si le point d'interaction p_i est connecté à un correspondant P_j, les informations de contrôle de P_{i_fils} et de P_i sont mises à jour de manière à exprimer un lien de communication entre P_{i_fils} et P_j.

Les extrémités d'un lien de communication sont appelees points terminaux. Si le point terminal se trouve sur un site distant, ceci entraîne la migration de la file d'attente des interactions héritées.

La file d'attente associée a un point d'interaction est mise a jour lors des emissions et receptions d'interactions. En Estelle, ces deux operations sont respectivement réalisées par l'instruction output et la clause when.

3) Communication

Emission d'interaction

Une émission d'interaction d'un point d'intéraction P_i à un point d'intéraction P_j ne peut être satisfaite que si au préalable il y a eu une connexion entre ces deux points d'intéraction.

L'émission distante d'une interaction se fait par appel aux primitives de communication du novau d'execution distribué. Dans tous les cas, l'intéraction émise est rattachée à la file du point d'intéract ion destinataire. De plus, si l'instance receptrice est à l'état passif, elle redevient active; une instance passe à l'état passif si aucune de ses transitions n'est satisfaite, dans ce cas seule l'arrivée d'un message pourrait éventuellement rendre satisfaite une de ses transitions.

L'émission de l'interaction déclarée m(t:t1; f:F1) exprimée par l'instruction Estelle:

OUTPUT P m(i,j)

est interpretee par:

* l'initialisation des par-ambtres de l'interaction 't' et 'f' avec les valeurs effectives 'i' et 'j'.

* appel de la primitive du noyau d'exécution reparti

récupérer adresse de P dans adrP
SEND (adrP, mess);

où adrP est l'adresse du point d'interaction émetteur et mess le message à transmettre.

Reception d'interaction

La clause when d'une aarde permet la reception d'une interaction en deux étapes:

- verification de l'existence, en tête de file, de l'interaction attendue lors de l'évaluation de la cfarde.

consommation de l'interaction, si la transition est tirée,

Les interactions distantes recueillies durant une étape de calcul sont distribuees sur les différentes files selon l'identité des points d'interaction au debut de chaque Ctape de calcul.

La reception d'interaction est interprétée par:

la recuperation I'adresse du point d'interaction.
- l'appel à la fonction Verif message0 dans la fonction garde i():
- et, l'appel à aet message() dans la procedure trans i() pour la consommation.

4) Instruction ALL

Elle est interpretee par l'instruction Pascal FOR.

ALL i: val initiale . val finale DO instructions:
où i est une variable qui n'est pas déclarée.

est interprétée par:

```
FOR $i_00: val_initiale TO val_finale DO
        instructions;
```

Les variables de contrôle sont générées de la manière suivante:

$i cpt oh cpt est un compteur de variables de contrôle des instructions ALL utilisées dans la specification. Ces variables doivent être déclarées.

IV.1.2.2 Interpretation d'une transition

A chaque transition est associée, à l'interpretation:

1- une fonction d'évaluation de la garde (Garde n°transition();), qui retourne la valeur -1 si la qarde n'est pas satisfaite. Dans le cas contraire, elle retourne la priorité de la transition.

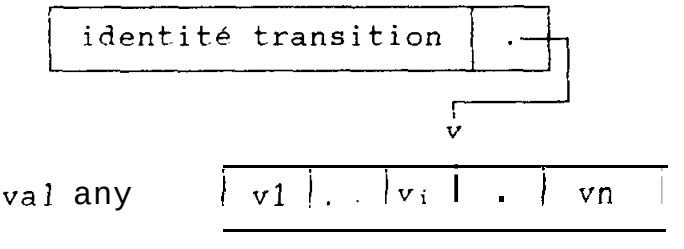
A une meta transition (transition ayant la clause "Any") on associe en plus une fonction meta qarde où la qarde est évaluée pour chacune des transitions de la metatransition.

Lors de l'évaluation d'une garde, si celle ci est satisfaite, la table des transitions satisfaites est mise 4 jour comme suit :

* pour une transition simple, on insère le descripteur suivant dans la table des transitions satisfaites:

identité transition	Nil

* pour une transition d'une metatransition. on insère le descripteur suivant:



où **val_any** est la liste des valeurs des variables référencées dans la clause "any" de la transition pour lesquelles elle est satisfaite.

A chaque étape de l'exécution, une transition tirable par instance, sera tirée a partir de la table des transitions satisfaites.

2- on associe aussi, a chaque transition, une procedure de tir qui comporte la partie action de la transition. Une fois le choix des transitions tirables fait, on execute la procedure de tir de chacune d'elles. S'il s'agit d'une meta_transition, nous avons une procedure meta_trans() qui executera le bloc action de la meta_transition pour un ensemble de valeurs attribuées aux variables de la clause "any".

Exemple:

```
trans from etatl to etat2
any i:1..5; j:2..7
when P[i,j].m
begin
      instructions:
end;
```

est interprétée comme suit:

```
function garde num(adrinstance,i,j): integer;
begin          end:

function metaaarde num(adrinstance): integer;
var prior,maxprior:integer:=-1;
heqin
  for i:=1 to 5 do
    for j:=2 TO 7 do
      begin  prior:=garde(adrinstance, i,j);
             if prior > maxprior then
              beqin  maxprior:=prior:
                     val any[K]:=i;
                     val any[k+1]:=j;
             end;
       end:
metagarde_num:=maxprior;
end;

urocedurr metatrans num(adrinstance) :
beqin
 ind:= tab trans[num].ind 1 any:
 trans num(adrinstance.val any[ind] .val any[ind+1]):
end:
```

Nous rappelons que la partie "initialize" d'un module n'est qu'une transition particuliere. Elle est traduite comme tout autre transition par une fonction d'évaluation et une procedure de tir.

La transition peut cornporter la clause "delay(e1,e2)", cela permet d'exprimer la notion de temps, par exemple modeliser un time-out. Les expressions el, e2 representent respectivement, le temps minimum durant lequel la transition doit être retardee et le temps maximum durant lequel elle peut Ctre retardee.

On associe au delay une structure de donnée adequate oh sont sauvegardées les deux expressions e1 et e2.

La mise en oeuvre de sa sdmantique est totalemnet prise en compte par le scheduler qui est decrit dans le chapitre V.

Les autres clauses constituant la garde sont interpret&es comme suit:

Provided: on traduit toute clause Provided <condition> par l'instruction conditionnelle: IF <condition> THEN

TO : interpret&e par une mise à jour de l'état de contrôle. Cette mise à jour est générée dans la vartie action afin d'être effective uniquement si la transition est tirée.

Prioritv : c'est une valeur retournée par la fonction garde si elle est satisfaite et elle est utilisée pour la selection de la transition à tirer.

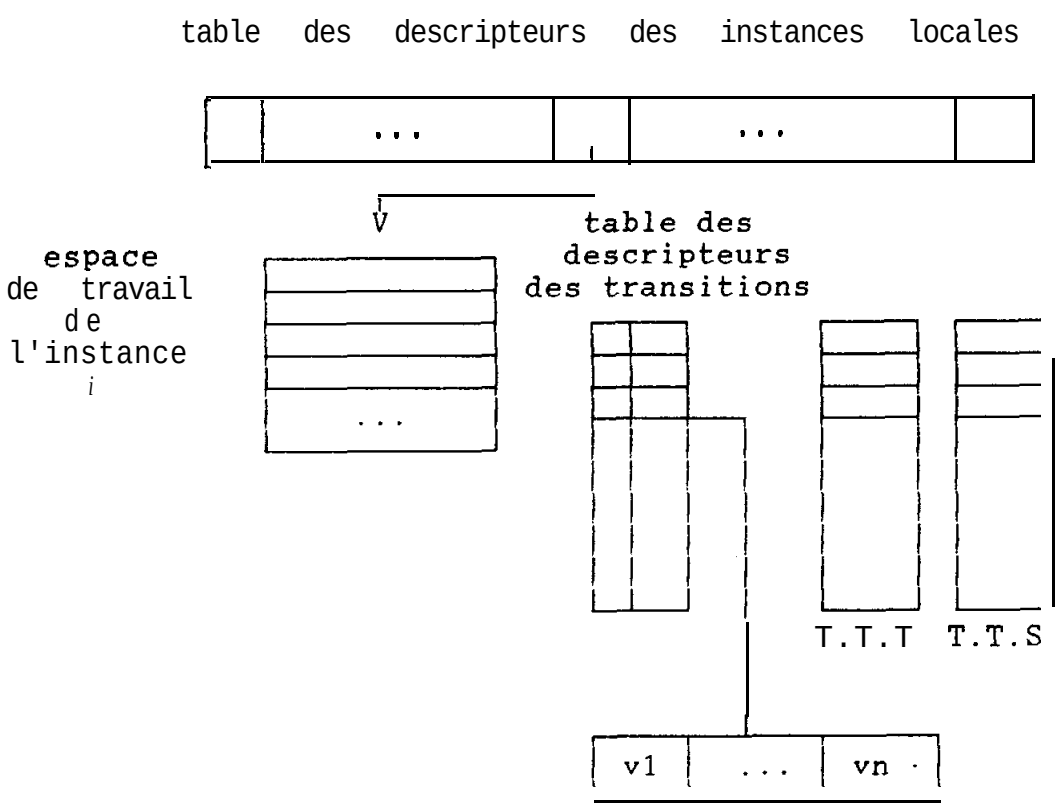
TV.1.2.3 Specification

Une spécification Estelle décrit globalement un algorithme distribué devant s'exécuter sur un ensemhlr de sites. L'interprétation de la specification se traduit par un ensemble de structures de données au niveau de chaque site. Ces structures permettent de contrôler l'execution des instances locales et la manipulation de leurs objets propres (point d'interaction, transition, . ..).

Nous avons donc, sur chaque sitelfigure IV.3):

- une table des descripteurs des instances locales,
- une liste des points d'interaction du site,

- une table des descripteurs des transitions,
- une table des transitions satisfaites et une table des transitions tirables à l'étape de l'exécution.



T.T.T: table de transitions tirables
T.T.S: table de transitions satisfaites.

Figure TV.3 : Structures de donneés sur un site

IV.1.2.4 Structure du code généré

La structure du code Pascal généré par l'interpréteur pour une specification Estelle a la forme générale suivante:

```
Module idf_spécification

 #include cst.pas;        (* liste des constantes *)
 #include typ.pas;        (* liste des definitions
                            de types de l'usager *)
 #include ctrl.pas;       (* decl. des tructures de
                            contrôle du site *)
 #include var.pas;        (* liste des variables
                            declarées par l'usager *)
 #include procuser.pas;
 (* definition des sous programmes de l'usager *)

(* definition des fonctions et transitions
     d'bvaluation et de tir des transitions     *)

 Function garde_0 (<liste paramêtre>);
  begin ...          end;

 Procedure trans_0 (<liste paramêtre>);
  begin ...          end:



 Function meta_garde i (<liste paramêtre>);
  begin
       ...
     < appel garde-i pour chaque transition de la
       meta_transition >
       ...
   end:

 Function garde i (<liste paramêtre>);
  begin . . .          end:

  Procedure meta trans i (<liste paramêtre>);
  heoin
       < appel trans i >
  end:

  Procedure trans i (<liste paramêtre>):
  begin . . .          end:


  Procedure CONFIGURATION:
  begin   <lance la racine de l'arbre >
  end:
end.
```

IV.1.2.5 **Désignation des objets**

Les objets (constantes, variables, types définis, parametres, points d'intéraction) d'une instance (body+module) sont accedes à travers la reference au contexte de l'instance.
Le renommage de certains objets (transitions, canaux, ...) est aussi necessaire, puisque la structure de bloc d'Estelle est partiellement détruite dans le code généré.

IV.2 Restriction d'implémentation:

Pour rbpondre à nos besoins, nous avonsdéfini une premiere interpretation pour le langage ESTELLE: ESTELLE STATIQUE. La mise en oeuvre d'ESTELLE complet ndcessite une technique de scheduling distribuée.

Un concept important d'ESTELLE qui est la priorité père/fils ndcessite, à chaque tir d'une transition, un contrôle distribué sur tous les sites du réseau, puisqu'une transition ancêtre est prioritaire sur celles de ses descendants. D'où la restriction: toute instance non feuille de l'arbre hiérarchique est passive. Cette restriction fait que la notion de variables exportées perd son sens.

La difficulté de mise en oeuvre du contrôle des modules de même filiation et d'attribut "activity" s'exécutant de manière séquentielle déterministe sur des sites distincts a donné une autre restriction sur l'attribut "activity". Donc, la notion d'attribut ou de classes de modules n'est plus prise en compte.

Les feuilles de l'arbre d'execution sont considérées d'attribut système.

Conclusion

Le point crucial de cette interpretation est de preserver la sémantique du langage Estelle en passant au code Pascal.

La notion de localité et de globalité des objets, la structure arborescente, et la structure en bloc ou module est partiellement détruite par l'interpretation Pascal. Ce qui a nécessité de garder des informations sur de tels concepts importants pour la specification, et de préfixer certaines variables pour préserver les droits d'accès des différentes tâches à celles r i

Certaines verifications d'erreurs semantiques qui relèvent de la semantique d'Estelle sont faites au niveau de l'interpreteur; comme par exemple verifier que deux points d'interactions connectés sont de même type et jouent des rôles opposes. Par contre, les verifications semantiques de Pascal sont laissées à la charge du compilateur Pascal.

Le sous-ensemble Estelle interprété est suffisant pour exprimer les programmes distribués a soumettre au système d'experimentation, a savoir les algorithmes de contrôle ou de calcul distribué de manière générale.

Du point de vue performances, les critères mis en jeu tels que:

- la vitesse de compilation,
- la qualité du code généré,
- la pertinence des diagnostics d'erreurs.
- et, la portabilite,

comportent divers compromis. Par exemple, la portabilite peut affecter l'efficacite. Un compilateur destine à une machine cible peut être plus performant qu'un autre portable.

De plus, il n'existe pas de test efficace pour affirmer qu'un générateur produit du code incontestablement correct. De nombreux articles ont abordé ce sujet. Malheureusement les compilateurs sont rarement spécifiés formellement. En outre. comme les compilateurs sont complexes. il reste la question de verifier que la specification elle même est correcte [AHO 89]. En pratique, on a recourt à des méthodes de test dans le but d'accroître notre confiance. Les programmes de test doivent impliquer le plus grand nombres d'instructions du langage.

CHAPITRE V

LA MACHINE VIRTUELLE REPARTIE

Dans les chapitres precedents, nous avons vu les étapes de passage d'une spécification Estelle à un code executable reparti. Ce code est soumis, pour être execute, à une machine virtuelle composée d'un noyau reparti. Ce dernier est une couche qui enveloppe le système d'exploitation de base autour duquel la machine virtuelle a été conçue. Schematiquement, ceci est représenté comme suit:

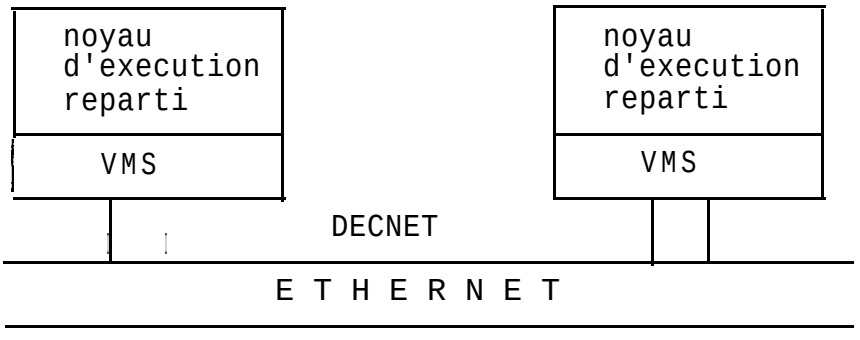


Figure V.1: environnement de travail

V.l Environnement de travail

Materiel: Physiquement, l'environnement est constitué d'ordinateurs de type VAX785 et MICROVAX, relies par un cable coaxial de type ETHERNET d'un debit de 10 MBITS/s avec un taux d'erreurs nul.

Logiciel: Les ordinateurs VAX et MVAX sont dotes des logiciels suivants:

- le système d'exploitation VAX/VMS,
- le gestionnaire de réseau DECNET [DIG 85].

Nous enrichissons ce logiciel par la couche noyau d'execution reparti.

Propriétés et hypotheses:

Avant d'entamer la description du noyau d'exécution réparti il est utile de mentionner les propriétés de l'environnement d'execution et les

hypotheses adoptées:

1. les differents sites evoluent 8 des vitesses différentes.
2. il n'y a pas de perte de messages
3. il n'y a pas de desequencement des messages
4. le délai de transmission est fini mais imprevisible.
5. il n'y a pas de duplication de messages.

Ces propriétés sont vérifiées par la quasi-totalite des reseaux.

V.2 Le noyau d'exécution réparti

La specification des programmes à analyser est soumise à un interpréteur Estelle-Pascal qui génére un code distribub. Ce dernier est ensuite execute par la machine virtuelle Estelle distribuée.

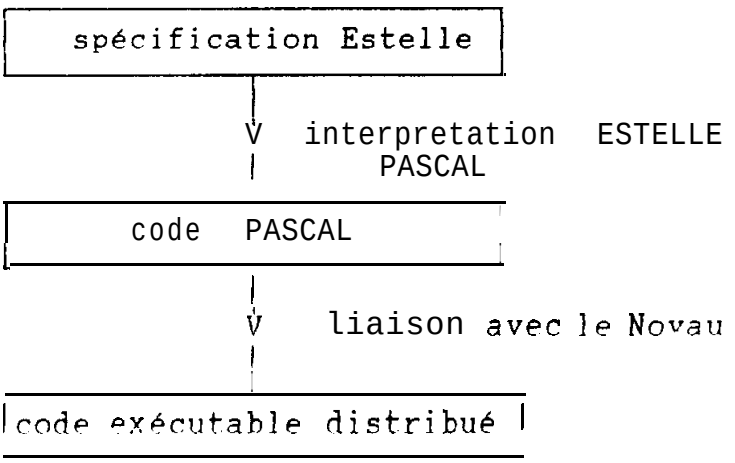


Fig V.2. Etapes du générateur de code distribué

Le novau d'exécution offre les services nécessaires à l'exécution, l'enchafnement d'instances, et la communication inter-instances. Il est charge d'offrir ces fcnctionnalites dans un environnement distribué. Il est construit en trois modules.

1) Un module charge de la configuration, de l'initialisation du système et du lancement du "scheduler" qui assure l'enchafnement des instances locales.

2) Un environnement de travail Estelle offrant les primitives de manipulation des objets Estelle n'existant pas dans le dictionnaire Pascal et prenant en charge la communication distante.

3) Une interface système disposant d'un ensemble de services dont la gestion du temps, la transmission des interactions distantes et la correspondance entre les instances et les identites des sites d'execution.

V.2.1 Lancement d'une spécification Estelle

Les executions des programmes sont parametrables. Ces parametres peuvent indiquer entre autres

une experimentation avec analyse de performances,
- une mise au point,
- une execution.

Ce module est un processus particulier qui gère le déroulement de l'exécution des instances locales. On dispose d'une copie de ce module d'execution sur chaque site. Son principal rôle est

- d'initialiser le système,
- d'analyser la commande d'entree,
- d'initialiser certaines données (numéro du site, horloge, . ..).
- de définir la configuration du système,
- de lancer le "scheduler",
- et de transmettre les traces éventuelles d'une exécution.

Après l'étape de génération de rode et liaison aver le novau d'exécution réparti une copie du code exécutable est disponible sur chaque site.

Un usager peut donc, lancer sa spécification à partir de n'importe quel noeud du réseau. Nous signalons qu'au niveau de chaque site, un veilleur qui n'est autre qu'un processus spécialisé, se charge de la communication distante. Ceci se fait en appelant les services offerts par le gestionnaire du réseau DECNET. Ce veilleur se charge aussi de "dispacher" les messages arrivant sur ce site, entre les différentes spécifications locales.

a) Initialisation d'une spécification

Un usager se presentant au niveau d'un site, lance une specification SPi. Les operations induites, transparentes à l'usager, sont les suivantes:

- creation des boites à lettres de contrôle et de calcul.

- diffusion du message 'L' afin de permettre aux autres sites de créer une image de SPi et les boites à lettres associées (les veilleurs distants se chargent de ceci).

- Attendre les 'OX' vehiculant le statut de creation distante des sites participants.

- Si le statut véhiculé dans le message 'OK' indique une execution normale, le traitement continue normalement, sinon un messaqe de destruction 'D' est diffusé à tous les noeuds du réseau pour détruire les images de la spécification qui ont été créées avec succés L'opération se termine en sianalant le problème 4 l'usager et en stoppant l'exécution.

S'il n'y a pas de problème, un messaqe 'S'(start) est diffuse pour signaler le lancement de l'exécution des imaqes de la specification sur tous les sites avec les parametres d'exécuti on specifies par l'usager à la creation de sa specification.

b) **Configuration du système:**

Lors de l'interprétation Estelle-Pascal, on qénère une procedure configuration. Le rôle de cette dernière est de lancer, sur le site initiateur, le processus racine de l'arbre hierarchique d'exécution representant l'instance particulière qu'est la specification. Cette dernière, à son tour. lance récursivement les instances filles. s'il y'en a. sur leur site de localisation. Si ces instances doivent s'executer sur ce site. l'insertion de leur descripteur dans la table des processus du site. leur assignation d'une identité unique. l'allocation de son espace de travail et de descripteurs pour ses points d'intéraction se fait au niveau de cette phase. Le même traitement se fait pour les instances distantes sur leur site de localisation.

Les sous instances distantes sont lancées par l'émission du messaae 'J' au site de sa localisation. Afin de respecter la priorite pére/fils, une sous instance n'est lancée que si le module pére a termine

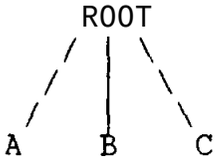
sa partie "initialize". Chaque niveau de l'arbre d'execution, a la fin de l'initialisation de ses noeuds, transmet une confirmation 'F' au niveau supérieur, jusqu'à atteindre la racine. A ce point, la partie configuration est dite terminee. L'enchainement des etapes de calcul peut commencer.

Notons que ce transfer& de connaissances est necessaire pour deux points importants:

- assurer la priorité pére/fils.

l'execution des parties 'initialize' des instances filles sur les sites distants ne peut se faire de manière totalement independante; car il risque d'y avoir des problémes tels qu'illustrés par l'exemple suivant:

Soit la specification ROOT shématisée par l'arbre d'execution comme suit:



Où chacune des instances A B et C s'exécute sur un noeud different. Les vitesses differentes des processeurs et la charge differente des systèmes, induisent un dephasage dans l'évolution de celles-ci. Si 'A' passe a l'execution de ces etapes de calcul, elle peut emettre un message vers 'C' qui n'a pas encore été créée totalement. Le noyau recevant le message tentera de le chaîner à la file de l'instance C qui n'a pas encore été identifiee.

Remarquons que, puisqu'il s'agit d'Estelle statique. l'initialisation d'instances, l'établissement des liens de communication se font à ce niveau.

c) Le "scheduler"

La structure hierarchique d'une specification Estelle a induit une dépendance entre les différents niveaux. Son comportement est régi principalement par deux concepts d'Estelle détaillée dans le chapitre III:

- La notion d'attribut
- La priorité pére/fils: Chaque module act if agit commeun superviseur pour ses fils.

Ainsi, découlent diverses formes d'exécution:

- une execution parallèle synchrone: attribut "process".
  une execution non deterministe: attribut "activity".
- une execution totalement asynchrone: attribut "system".

Cet enchaînement de tâche est assure par le "scheduler". C'est un programme distribué qui prend en rharcre l'exécution des instances initialisées sur les différents noeuds. Chaque processus le composant s'exécute de manière cyclique sur chaque noeud.

Afin de preserver la sémantique d'exécution des instances Estelle, le scheduler doit assurer l'enchafnement des étapes de calcul où chaque étape représente une phase de choix des transitions tirables pour les instances qui doivent s'exécuter en parallèle synchrone ou asynchrone. De plus. 1 'execution d 'une transition doit se faire de manière atomique et le passage à une autre étape de calcul ne se fait que si toutes les transitions sélectionnées à l'étape courante ont été tirées.

Dans le cas de notre système, le parallèlisme est asynchrone entre les différents sites. Le placement des processus sur les sites physiques peut être explicitement défini par l'utilisateur en affectant au premier paramètre d'un module l'identité du site correspondant: l'association étant effective lors de la création de l'instance (chaque noeud a une identité unique). S'il n'est pas précisé, le placement des tâches est assuré par le novau et ceci en distribuant les instances sur les sites participants. Au niveau d'un site, une exécution asynchrone définit un comportement pseudo-parallèle des instances locales. Toutefois. si l'utilisateur désire une exécution parallèle synchrone. i 1 pourra le préciser au lancement du programme.

Une execution déterministe peut être demandée par l'utilisateur grâce aux paramètres d'entrée. Dans ce cas, le "scheduler" doit choisir la transition à tirer selon un crithre prédéfini.

Grossiérement, le scheduler se schématise comme suit:

debut
tant que non fin de la spécification ou tmax non écoulé faire
    récupérer les messages externes
    évaluer les gardes
    tirer les transitions satisfaites
 fait
fin.

où tmax est la durée maximale d'exécution de la specification.

1/ Recuperation des messacres externes:

Au début de chaoue étape de calcul, les messacres externes reçus par le veilleur du site durant l'étape de calcul écoulée sont récupérés puis chainés aux files des points d'interaction destinataires.

2/ Evaluation des *gardes:*

Cette étape consiste à évaluer les gardes de toutes les transitions de chaque instance locale. Deux cas se présentent:

a/ évaluation non déterministe:

Les transitions satisfaites ou valides sont stockées dans tab enable trans dans le but d'en choisir une de manière aléatoire. Tab enable trans contient les transitions les plus prioritaires qui sont déterminées grâce à la clause "Priority".

A la fin de cette étape d'évaluation, au plus une transition par instance locale est choisie pour être tirée. Les instances n'avant aucune garde satisfaite seront désactivées. Ceci est fait dans un but d'optimisation: en effet, il est inutile de les réévaluer pour les prochaines étapes de calcul tant qu'il n'y aura pas d'événements succeptibles d'affecter leur validité. Or, dans notre cas. le seul événement qui risque de valider une garde est l'arrivée d'un message. Dans ce dernier cas, l'instance est activée.

Si la garde comporte la clause "delay", les expressions de temps sont évaluées, une table de gardes "delayed" est mise à jour par les informations concernant:

- le temps minimum de retard du tir, ,
- le temps maximum de retard de tir,
- l'instant initial de validité.

Pour preserver la semantique du **Delay**, le traitement ci-dessous est **réalisé**. Il permet d'assurer qu'une transition comportant une clause **Delay** satisfaite au temps 'T' sera **retardée** d'au moins el unites de temps et d'au plus e2 unites de temps, en garantissant qu'entre temps elle reste valide. Dans le cas contraire, elle est retiree de la table.

```
pour chaque instance active
faire
 pour chaque transition 'tri' faire
      évaluer la garde de la transition 'tri'
      si satisfaite alors
            si delay alors
                 recuperer (time)-->T
                 recherche 'tri' dans tab_trans delayed
                 si trouve alors
                           ts = tab trans delayed.time
                    sinon ts = T
                 fsi
                 si T-ts <e1 ou T-ts <e2 alors
                       inserer(tri,tab trans delayed)
                 fsi
                si T-ts>=e1 alors mettre 'tri' dans
                                        tab enable trans
            sinon mettre 'tri' dans tab-enable-trans
            fsi;
      sinon si delay alors
               supprime(tri,tab_trans_delayed)
     fsi;
     choisir une transitioin tirable
     si la transition choisie est delayed alors
                       supprime(tri,tab trans delayed)

     fsi
   fait
fait
```

Ce traitement assure qu'une transition comportant une clause Delay, une fois validée doit le rester jusqu'à au moins e1 unités de temps.

b/ évaluation deterministe:

Dans cc cas, les priorités sont ignorées, et la transition choisie pour le tir. pour chaque instance. est la première valide.

La possibilité d'offrir le choix du critère de determinisme serait envisaqeable. Pour cela, les critères seront fixes statiquement et l'usager devra préciser le critère de determinisme qu'il desire appliquer. Le scheduler sera alors orienté dans le choix des transitions à tirer.

3/ Tir des transitions:

L'étape d'évaluation des gardes fournit l'ensemble des transitions tirables dans tab-fir-trans, une, au plus, par instance.

Rappelons que deux types de parallélismes sont possibles au niveau d'un site:

- execution synchrone,
- execution asynchrone.

a/ Execution asvnchrone

L'exécution entre les différents sites est asynchrone. Cependant, du fait que le nombre d'instances initialisées peut Ctre superieur au nombre de sites, plusieurs instances peuvent s'exbcuter sur un m&me site. Donc, le comportement au ni veau d'un site n'est que quasi-parallele. Il existent des sémantiques d'asynchronisme que nous explicitons ici en se basant sur [GROZ 891:

Sémantique de l'entrelacement

Une description Estelle correspond à un ensemble d'états glohaux constitué de l'exécution d'un ensemble de transitions locales. Puisque les instances sont asynchrones. si une transition T_i est tirable, l'evolution, ou le changement d'état, isolée de l'instance M_i par le tir de T_i, est une evolution possible du système global. Ceci siqnifie que l'instance M peut executer une transition pendant que les autres ne font rien. C'est à dire que toute evolution du système peut être vue comme une suite d'évolution élémentaires entrelacées des differents systèmes. Mais, cette semantique ne représente pas le croisement des messages.

Exemple: Protocole de connexion et deconnexion.

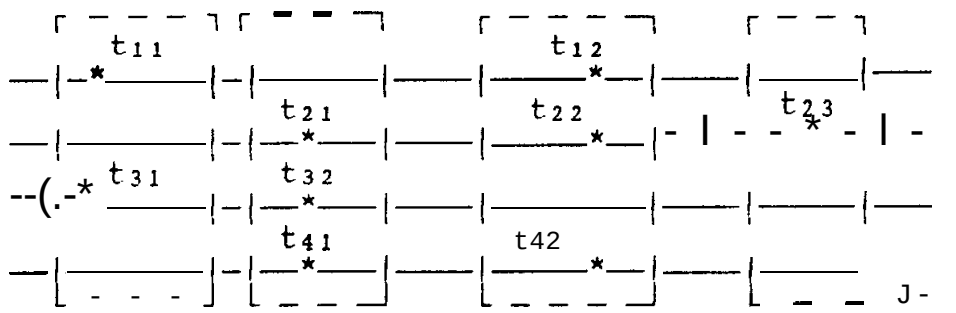
```
initialize to ouvert
from ouvert to ie-ferme output dcnx:
from ouvert when dcnx priority 1 to il ferme
                <t2>:
from ic-ferme when dcnx to nous-fermons <t3>.
```

On remarque qu'avec cette sémantique l'état 'nous-fermons' n'est jamais atteind.

Ainsi, le fait qu'une transition puisse inhiber une transition d'un autre module qui serait tirable en parallèle, rend la semantique d'entrelacement insuffisante pour representer correctement le parallelisme.

Sémantique des lots parallèles

Pour pallier à l'inconvenient de l'entrelacement, cette semantique n'empêche pas l'execution simultanee des transitions de plusieurs modules. Le schema suivant illustre cette idée:



Cette sémantique permet d'explorer plus de comportements qu'une sémantique d'entrelacement (possibilité de croisement de messages) [GRO 891.

Dans notre cas. nous ut ilisons la semantique des lots pararlèles. Cette dernière est réalisée en utilisant *une* fonction aléatoire pour le choix des transitions qui constitueront le lots à tirer.

b/ Execution synchrone

Le mode synchrone décrit en Estelle, consiste à executer dans une même étape de calcul toutes les transitions sélectionnées pour le tir. La mise en oeuvre de ce type de synchronisme n'est réalisé qu'au niveau d'un site. La généralisation sur le réseau nécessite une technique de scheduling plus complexe.

Rappelons que le principe du parallèlisme svnchrone Estelle consiste à executer une transition par instance à chaque étape de calcul. Une étape de calcul est dite terminée si toutes les transitions sélectionnées pour cette dérnière sont tirées. Dans le cas d'un environnement réparti, les instances sont distribuées sur tous lcs noeuds du réseau. Le contrôle

de fin d'une étape de calcul nécessite une cooperation entre les différents sites. Le passage d'une étape à une autre au niveau d'un site devra se faire au même instant global au niveau des autres sites. Ce contrôle distribué induit une surcharge en messages supplementaires de 2(n-1) messages à chaque étapes de calcul; où n est le nombre de noeuds du réseau. A cet effet, et pour cette première version de la machine virtuelle, une execution synchrone d'une specification se fera localement.

4/ Sorties des traces:

Estelle n'offre aucun moyen pour communiquer des résultats à l'usager puisque le langage Pascal utilisé est le Pascal IS0 niveau 0 sans entree/sortie. Néanmoins, nous offrons l'instruction VIEW() pour permettre à l'usager de suivre le déroulement de son programme mais en différé. La machine virtuelle Estelle est construite dans un but d'experimentation d'algorithmes distribués et d'analyse de performances, il y a donc lieu d'éviter toute perturbation de l'execution. Pour cela, on retarde les entrees/sorties à la fin de l'execution. En cours d'exécution, les traces sont mémorisées puis affichees sur écran ou récupérées sur fichier à la fin de la specification.

5/ Terminaisnn de la specification:

Le modèle d'éxécution Estelle n'offre pas un moyen de détecter la terminaiscn de la specification. L'automate à états finis ne se termine pas. Pour assurer la terminaison de la spécification. Le système offre deux alternatives:

- Une fonction $terminate(): l'utilisateur connaissant son prosramme peut faire appel à cette fonction à des endroits bien précis de son prouramme. Une instance qui execute cette fonction se declare terminee et informe son pére. Une instance pére n'étant pas active (car seules les feuilles de l'arbre sont actives) se declare terminee si toutes les instances descendantes sont terminees. Ce processus remonte jusqu'à la racine. A ce point, la specification Ctant la racine, n'ayant plus d'instances filles actives se declare terminee.

- une durée maximum d'exécution: L'usager spécifie cette durée au lancement de sa spécification.

V.2.2 Environnement de travail pour Estelle:"Runtime"

Estelle inclut le langage Pascal et l'encapsule dans des elements d'expression du parallélisme. L'initialisation d'instances descendantes, la creation de points d'interactions, de leurs connexions, l'émission et la reception d'interactions sont exprimees par des instructions Estelle qui ne figurent pas dans le dictionnaire du langage Pascal. Ces instructions sont traduites par des appels a des primitives du runtime qui constituent cet environnement.

1. **Creation d'instances**

Elle consiste en la création du bloc de contrôle de l'instance, l'emission du message 'I' à tous les sites participants afin de créer sur les sites distants son bloc de contrôle, et le chainer comme sous module du père. De plus, si l'instance créée est locale on crée son contexte et ses points d'interaction.

Dans le cas où l'usaqer ne précise pas le site de localisation de l'instance, le noyau prend en charge le placement de celle-ci sur les differents sites participants en les distribuant de manière aléatoire. De même, si l'usaqer spécifie un numero de site inexistant (car le numero de site qui est le premier paramètre du module peut être une expression arithmetique évaluée dvnami auement pouvant donner un résultat different des numéros des sites existants) dans ce cas. le novau calcule leur numéros A partir de ce dernier de manière A les placer sur des sites existants.

2. Connexi-on et raccordements

L'appel aux primitives CONNECT et ATTACH peut mettre en jeu des points d'interaction locaux et/ou distants.

* Cas du Connect:

Si les deux points d'interaction sont locaux, leurs descripteurs sont mis à jour de manière à exprimer une liaison de type "connexion" entre les deux points specifies.

Si l'un des deux est distant, un message 'C' est émis vers le site de localisation du point d'intéraction distant et attend un 'OK' pour confirmer la connexion.

Si les deux points sont distants alors le message 'P' est Cmis vers le site de localisation du premier point d'interaction. Ce site arendra en charge la suite de l'opération.

* Cas d'un attachemment

Le premier point d'interaction doit être local puisqu'il appartient à la tâche appelante. Le second doit appartenir à une instance fille et peut eventuellement être distant. Divers cas se présentent:

- le premier point n'est pas connecté et le second point n'est pas attache à un point d'interaction de la tdche descendante.

- le premier point n'est pas connecté et le second woint est attach6 à un point d'interaction de la tdche descendante.

- le premier point est connecté e t le second point n'est was attaché à un point d'interaction de *la* tdche descendante.

- le premier point est connect6 et le second point est attache à un point d'interaction de la tâche descendante.

Dans les quatre cas, l'instruction ATTACH sera interprétée par l'établissement d'un lien entre les deux extrémités ou points terminaux.

Les deux points terminaux peuvent être dans les situations suivantes:

- les deux extrémités sont locales: une mise à jour locale *des* descripteurs est faite.

- les deux extrémités sont distantes sur des sites différents: un message de code 'Q' est envoyé au site de l'une des deux extrémités qui prendra en charge la suite de l'opération.

- l'une des deux extrémités est locale. 1 'autre est distante: un messaae 'A' est envoyé vers le site de l'extrémité distante et attente d'un message O K pour confirmation de la liaison. puis mise *A* jour du *descripteur* local.

3. Communication:

3.1 Reception de message:

La reception de messages se fait en deux étapes:

- Verif_message(): qui vérifie si l'intéraction spécifiée comme paramètre se trouve en tête de file du point d'intéraction passé en paramètre lui aussi.

- get_message(): qui récupére le message en tête de file pour la consommation.

3.2 Envoi de message:

Pour ce faire, la procedure SEND() est appelée. Deux cas sont pris en compte:

- un envoi local: résumé en un chainage à la file du point d'intéraction destinataire.

- un envoi distant: qui fait intervenir les services offerts par le gestionnaire réseau DECNET.

4. Tt-aitement des messages de contrôle:

Les messaaes de contrôle subissent un traitement spécifique selon le code qui leur est assigné.

cas 'I': la reception de ce type de message entraine:

- la création du bloc de contrôle
- la creation du contexte si l'instance est locale et de ses points d'intéraction.

cas 'J': - lancer l'instance spécifiée dans le message, puis emission d'un message de confirmation à la fin de la partie "initialize" de celle ci.

cas 'A','C': - mise à jour des descripteurs des points d'intéraction et emission d'un messaae de type OK

cas 'O','P': - si les deux points d 'interaction sont locaux, les descripteurs sont mis à jour localement, sinon émettre un messaae de type 'A' ou 'C' et attendre un messaae Ok Dour mettre à jour les descripteurs.

cas 'E' : - afficher l'erreur et détruire les images de la spécification sur tous les sites.

cas 'T' : mettre à jour le nombre de sous

instances. Si ce dernier est nulle, envoyer un message de terminaison à l'instance père (operation recursive).

cas 'S' : récupérer les parametres d'execution et commencer l'exécution des instances locales.

cas 'K' : comptabiliser le nombre de messages de ce type en verifiant le statut véhiculé. Si aucun n'est un statut d'erreur alors emettre le message 'S' véhiculant les paramètres d'execution, sinon emettre le message 'D' pour detruire les images distantes de la specification.

5. Fonctions prédéfinies d'Estelle implémenté

En plus des fonctions prédéfinies d'Estelle (sin, cos, log,..), les fonctions suivantes sont offertes:

La fonction VIEW(): citée précédemment, permet un traçage en différé.

La fonction RANDOM(): retourne une valeur aleatoire dans un intervalle spécifié en parametre.

La fonction getim_loc(): retourne le temps local à un site.

La fonction getim_glb(): retourne le temps global estimé grace à un algorithme d'estimation d'un temps global décrit dans [BOU 92].

V.2.3 Interface système:

Tout ce qui dépend étroitement du système cible a été rassemblé dans un module. Cette structurat ion rend la localisation des modifications plus aisée et facile en passant d'un système à un autre.

Les primitives du Runtime ayant trait au réseau ou au système d'exploitation font reference aux primitives du noyau Le lancement des veilleurs, la creation des boites à lettres et des liens de communication. l'envoi et reception de messages, la recuperation du temp.5 local sont des fonctions de la bibliothèque système. Certaines d'entre elles font appel à des services offerts par le gestionnaire de réseau DECNET.

V. 2.4 Les processus veilleurs de sites

Les veilleurs sont des processus qui une fois lances s'executent de manière permanente.

A chaque specification, sont associées deux boites a lettres; une pour les messages de contrôle et une pour les messages de calcul. Les messages de contrôle sont reçus de maniere asynchrones et un traitement specifique leur est associé. Une troisieme boite à lettres destinée a l'émission vers l'exterieur est commune a toutes les specifications locales à un site (Figure V.3).

Ces processus se chargent de la reception des messages distants, et *la* transmission distante des messages locaux.

Traitement des messages par *les veilleurs*

Aprés l'initialisation du réseau et l'établissement des liaisons. le veilleur se met *en* attente des messages externes et locaux destinés à l'exterieur

1. A la réception des messages distant-s:

Cas du type du messaqe:

'L' : - création de la specification et de ses boites à lettres de contrôle et de calcul.
- emission du message Ok au site initiateur comportant le statut de creation.

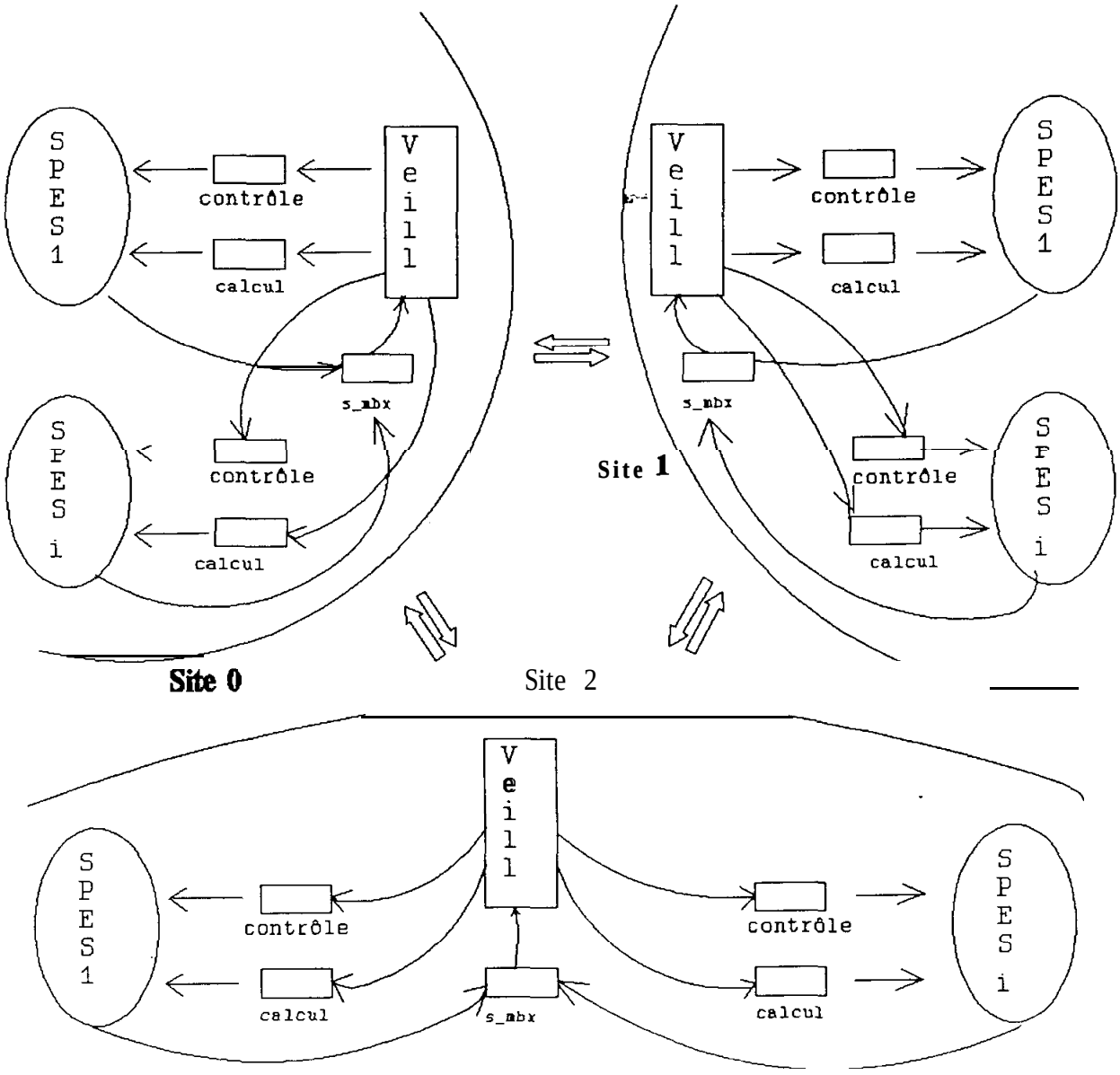
'D' : destruction de la specification, de ces boites à lettres, et ses canaux de communication.

default: emission *sur* la hoite à lettres appropriée locale (contrôle ou calcul).

2. A la réception des messages locaux :

Cas du type du message:

'I' :- création des boites à lettres de contrôle et de calcul.



communication locale

communication distante

Figure V.3: Système de communication

- transmission du message à son destinataire distant.

'D' : - destruction de la specification, de ces boites à lettres, et ses canaux de communication.
- transmission du message à son destinataire distant.

default: transmission du message à son destinataire distant.

V.3 Outils autour de la machine Estelle répartie

La machine virtuelle Estelle représente une couche permettant l'implantation de programmes distribués specifies en Estelle et sur laquelle se greffent des outils de mise au point dans un environnement distribué (Figure V.4).

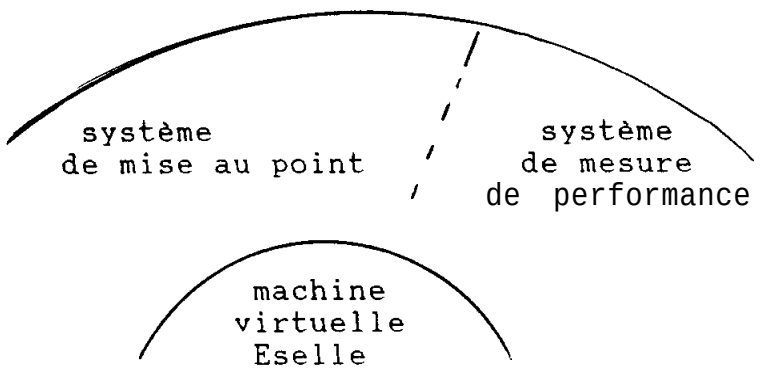


Figure V.4: Outils autour de la machine virtuelle

Parmi ces outils nous citons:

1. Un outil de mesure et d'analyse de performance: Ce système constitué d'un ensemble d'observateurs et d'analystes intégrés à cette machine Estelle. Cet outil est détaillé dans [BOU 921. II offre aussi. un outil de construction d'un temps global afin de pouvoir mesurer certaines entités telle que le temps de transmission d'un message. Dans ce système un programme réparti exprimé à I'aide d'une technique de description formelle est décrit comme un automate à

transitions d'états fini. Ce programme réparti peut Ctre modélisé selon une structure hiérarchique: un programme réparti est un ensemble de processus s'executant sur differentes machines, chaque processus est un ensemble de transitions, une transition peut être un ensemble d'appels de procedures, et chaque procedure peut être un ensemble d'appels de primitives. La figure V.5 nous donne un exemple de representation hierarchique d'un programme ou specification distribuée comportant plusieurs activites parallèlles sur differentes machines. Cette representation peut, Cvidemment, être etendue [YAN 87]. Les différents niveaux d'abstraction sont:

Niveau programme (spécification): A ce niveau, le programme est vu comme une boite noire s'executant sur un certain systeme pour lequel l'usager fournit des entrees et récolte des résultats. Les mesures a ce niveau concernent le comportement général de tout le programme. Par exemple, le nombre de processus, le temps total d'execution, le temps total d'attente, trafic en messages . . .

Niveau machine: Le programme est constitué de multiples composants (processus) s'executant simultanement sur les machines du système. On s'interesse, à ce niveau, aux informations sur chaque machine et sur les communications entre les differentes machines. Tous les événements sont totalement ordonnés car ils font tous reference a une même horloge. Mais, nous n'avons aucun detail sur la structure des activites d'une machine. La structure du niveau machine peut changer d'une execution a une autre et même durant une même execution dans le cas de miqration de processus. Ce niveau est sssentiel pour l'étude, par exemple, de l'impact de l'affectation des processus aux differentes machines sur les performances d'un prosramme distribué.

Niveau processus: Le prooramme distribué est vu comme une collection de processus. On peut voir ces processus répartis en groupes;- Chaque qroupe comportant les processus s'exécutant sur *une même* machine-, comme on peut les *voir en* un seul groupe en ignorant les limites des machines. Dans le premier cas, on peut s'intéresser à mesurer la competition des processus pour le partage des ressources locales. Dans le second cas, le comportement des processus est étudié independamment des machines.

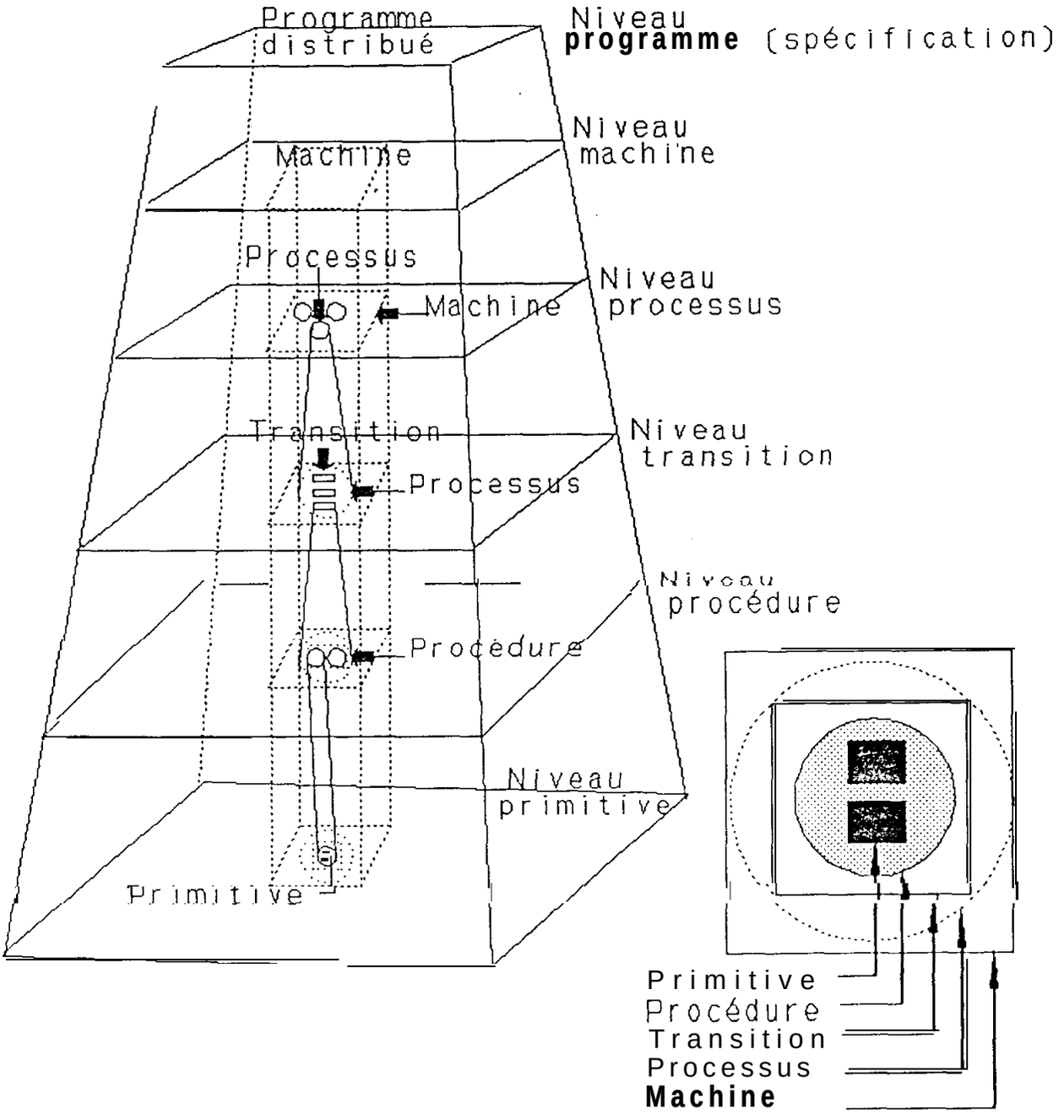


Figure V.5: Modèle d'observation hiérarchique.

Niveau transition: La spécification du programme distribué étant décrite formellement par un ensemble de transitions d'états [ISO2 89][JAR+ 87], la transition constitue une entité importante à prendre en compte; un processus est l'exécution d'une sequence de transitions atomiques.

Niveau procédure: Une transition peut être représentée Dar une chaîne séquentielle d'appels de procedure. On peut avoir besoin d'informations sur l'exécution des subdivisions d'un processus afin d'etudier les performances d'un programme.

Niveau activité-primitive: C'est le niveau le plus bas de la hiérarchie où est manipulée la plus petite entité: l'événement (blocage, deblocage, emission-reception de messages, creation destruction d'objets,..).

2. Mise au point: Une application intéressante est la réexécution [BADb91] et l'insertion de points d'arrêt[HAB 88]. Ceci est un projet qui a été lance et qui va se areffer à la machine virtuelle Estelle dans le but de faire du "debugging" de programmes spécifiés en Estelle[BEN 92].
Les caractéristiques principales de ce système sont:

- l'introduction d'un minimum de code supplémentaire et une perturbation moindre de l'exécution originale.

-l'enregistrement d'un script minimal afin de permettre la réexécution de spécifications longues.

- la non utilisation d'un temps global.

Cette méthode de réexécution se classe parmi les techniques de réexécution concurrentes [LEU 90].

De plus. l'insertion de points de contrôle permet d'arrêter le prouramme à des points vérifiant certains prédicats; pour ce dernier point, la determination d'un état global est necessaire.

V.4 Utilisation de la machine virtuelle:

Après avoir écrit une specification Estelle sous 1'éditeur disponible, on la soumet au compilateur Estelle en faisant appelà la commande EST suivante:

> EST nom spes.est

l'extension du fichier est ".est" (par defaut).

Cette commande appelle l'interpreteur Estelle pour traduire la specification en code Pascal en générant un code "nom.spes.pas", qui sera soumis au compilateur Pascal local puis lies au noyau d'execution réparti afin de générer du code executable réparti "nom_spes.exe", si aucune erreur n'est décelée.
Pour executer ce code , il suffit de faire appel a la commande RUN:

> RUN nom_spes

l'execution de cette commande induit :

l'apparition d'une fenêtre dans laquelle l'utilisateur spécifie les paramètres d'exécution et le but de cette dernière.

A la fin de l'exécution de son programme (fin normale ou expiration du délai maximum), l'utilisateur est en presence d'un ensemble de fenêtres, chacune representant un noeud du réseau. Dans chacune seront rapatriés les résultats partiaux de l'execution de la specification, si l'usager a fait appel à la primitive VIEW() permettant l'affichage de résultats.

CONCLUSION

La réalisation de la machine virtuelle Estelle répartie permet:

* d'implanter des algorithmes distribués dans le but de les experimenter afin de confirmer ou d'infirmer leur compléxité théorique,

* de mettre en evidence des phénomènes non perceptibles par la simulation ou la verification formelle,

* de les mettre au point grâce aux outils intégrés, à savoir: un outil de mise au point de programmes distribués et un outil de mesure et d'analyse de performances.

Un prototype de cette machine a été conçu et développé sur un ensemble d'ordinateurs de type VAX sous le systeme d'exploitation VMS (Virtual Memory System) connectés par un réseau local de type Ethernet.

Il offre un environnement d'experimentation permettant la generation et l'execution d'un code distribue obtenu à partir d'une description formelle Estelle.

L'interpretation de la specification est caractérisée essentiellement par:

1) La modularite et l'uniformite du code généré.

2) La conformité entre la specification et la traduction automatique.

3) La portabilité du système écrit en Pascal et C. La partie dependante du système est facilement localisable.

Cependant, la génération de code proposée est valable pour un sous ensemble Estelle: Estelle statique. Or, l'aspect dynamique est un aspect important d'Estelle. Il est donc intéressant, afin d'obtenir un générateur de code pour Estelle complet, d'inclure le partage d'objets entre père et fils et les différentes assignations d'attributs en elargissant le novau d'exécution réparti. Dans cc cas. une technique de scheduling distribuée est nécessaire lors de l'exécution. Un contrôle est effectue à chaque tir de transition pour verifier que ces deux concepts sont

respectés à l'execution. Ces deux extensions concernent aussi bien le générateur de code que le noyau d'execution.

Remarouons que la conception et la realisation de cette machine ont été faites sur un environnement homogéne. Neanmoins, l'extension de l'environnement actuel à un environnement hétérogène (adjonction de machines d'autres types) n'entralnera que la revision du module intrinsequement lié au système d'exploitation hôte. En cas de changement du gestionnaire de réseau DECNET, il faudra revoir les processus veilleurs des sites charges de la communication distante.

La conception et la realisation de cette machine permet l'implantation de specifications de protocoles et de programmes distribués. lors de l'écriture de ces specifications, l'usager doit se soucier de l'asynchronisme des communications et des problèmes induits par ce nondeterminisme. ETHERNET supporte physiquement le principe de diffusion. Aussi, afin d'ameliorer les performances du systbme, il serait avantageux d'intégrer dans le système de communication un protocole de diffusion ou de communication multidestinataires [BIR 861 [BIR 87] [RAYb 91]. L'adionction de plusieurs variétés de ce protocole mettant en oeuvre des propriètès d'ordre général. causal et atomicité permettra de simplifier les spécifications et la conception de prosrammes et systèmes répartis [RAYb 91].

BIBLIOGRAPHIE

[ADA 88] M. ADAM, P. INGELS et M. RAYNAL
Algorithmes distribués synchrones et systèmes répartis asynchrones: Concepts, mise en oeuvre et experimentation.
RR 862 n°411, INRIA, centre de l'IRISA, RENNES, France, JUILLET 1988.

[ART 87] Y. ARTSY, H.Y. CHANG, R. FINKEL,
Interprocess communication in Charlotte.
IEEE software, january 1987.

[AUD 86] E.AUDUREAU,L.FARINAS DEL CERRO, P. ENJALBERT
Théorie de la programmation et logique temporelle, premiere partie: Validation d'algorithmes séquentiels
TSI,Vol 6, n°6 1986.

[AUD 88] E.AUDUREAU,L.FARINAS DEL CERRO, P. ENJALBERT
Théorie de la programmation et logique temporelle, deuxieme partie: Validation d'algorithmes paralleles.
TSI,Vol 7, n°2 1988.

[AHO 89] A. AHO, R. SETHI, J. ULLMAN,
Compilateurs, principes, techniques et outils.
Intereditions, collection IIA. 1989.

[ANS 89] J.P. ANSART, M. DIAZ, J.P. COURTIAT, P.AZEMA
V. CHARI,
titre: Rapid prototyping of an Estelle simulator: ESTIM.
The formal description technique Estelle.
Results of the ESPRI/SEDOS project, 1989.

[BADa91] N. BADACHE, S. BOUALLAG, N. SEMAOUNE,
Une machine virtuelle estelle répartie.
First maghrebin symposium on programming and systems, Institute of computer science, USTHB, Oct 21-23/1991, pp:115-131, Algiers.

[BADb91] N. BADACHE, M. BENCHAIBA,
Reexecution d'une specification Estelle.
First maghrebin symposium on programming and systems, Oct 21-23 1991. Algiers.

[BAL 89] H.E. BAL, J.G. STEINER,A.S. TANEMBAUM
Programming languages for distributed computing systems.
ACM computing surveys Vol 21. n°3 sept 1989.

[BEN 92] M. BENCHAIBA,
Outils pour la mise au point de programmes distribues.
These de magister en informatique, USTHB, 1992 (à apparaitre).

[BER 87] G. BERNARD, A. DUDA, Y. HADDAD, G. HARRUS,
Primitives for distributed computing in a heterogenous local area network environment.
Université de Paris-Sud, Centre d'orsay, rapport interne N°60, avril 87.

[BER 89] G. BERNARD
Performance et systèmes repartis: de la modelisation à la rdalisation
These de docteur en science en informatique, Unrversite de Paris-Sud, Centre d'orsay. Avril 1989.

[BIR 863 K.P. BIRMAN,
ISIS: a system for fault-tolerant distributed computing,
Department computer science, cornell university of Ithica of New york 14853-7501, Technical report TR 86-744, avril 86.

[BIR 87j K.P. BIRMAN, T.A. JOSEPH,
Exploiting virtual synchrony in distributed system,
Department computer science, Cornell university Ithica New york 14853-7501, Technical report TR 87-811, feb 87.

[BIR 86] K.P. BIRMAN,
ISIS: a system for fault-tolerant distributed computing,
Department computer science, Cornell university Ithica New york 14853-7501, Technical report TR 86-744, avril 86.

[BRO 87] J. Y. BROSSARD
Comparaison d'outils de validation pour les algorithmes distribues et protocoles.
NOTE Techniques NT/LAA/SLC/263
Centre national d'études des telecommunications (CNET LANNION). JUIN 87.

[BOU 92] S. BOUALLAG,
Outils pour l'analyse de performances de programmer distribues.
These de magister en informatique, USTHB, 1992.

[BUR 89] H. BURKHART et R. MILLEN
Performance-Measurement Tools in a Multiprocessor Environment
IEEE Trans. on Comp., Vol. 38, n°5, May 1989.

[CHA 79] E.J. CHANG, R.ROBERTS,
An impoved algorithm for decentralized extrema-finding in circular configurations of processors,
Comm. ACM vol 22,5, May 79, pp:281-283.

[CHO 88] J. D. CHOI, B.P. MILLER,
A mechanism for efficient debugging of parallel programs,
Proceeding of the SIGPLAN'88 conference on programming language design and implementation ATLANTA, Georgia, June 22-24 1988.

[CLA 83] D.W. CLARK,
Cache performance in the VAX11/780.
ACM transaction on computer systems, 1:24-37, 1983.

[COR 81] CORNAFIAN,
Systèmes informatiques repartis: Concepts et techniques.
Edition DUNOD, 1981.

[COU 89] J.P. COURTIAT, M. DIAZ, J.P.ANSART, P.AZEMA, V. CHARI,
titre: introduicing a rendez-vous mechanism in Estelle: Estelle*.
The formal description technique Estelle.
Results of the ESPRI/SEDOS project.

[CRI 89] F. CRISTIAN,
Probabilistic clock synchronisation distributed computing, 3(3):146-158, Springer-Verlag, July 1989.

[DEM 87] P. DEMBINSKI, S. BUDKOWSKI
An introduction to estelle: a specification ianguage for distributed systems.
Computers networks and isdn systems 14, 1987
North holland.

[DIA 89] M. DIAZ
Environnements logiciels pour la conception des protocoles dans ies systemes distribués.
Séminaire franco_bresilien sur les systemes informatiques repartis. 11-04/09/89
Florianopolis-sc Bresil.

[DIG 85 j DIGITAL,
Les réseaux: mise en oeuvre de DECNET.
Educational services AOUT 85.

[DIJ 743 E.W. DIJKSTRA,
Self-stabilizing systems inspite of distributer control.
Corn. ACM vol 17. N° 1 Nov 1974.

[DIJ 80] E.W DIJKSTRA, C.S. SCHOLTEN,
Terminaison detection for diffusing computations.
Information proceeding letteres, vol 11, n°1, aout 80.

[DOL 84] D.DOLEV, J.Y. HALPERN, R. SHUNG,
On the possibility and impossibility activing cloc synchronization.
In proceeding of 16th ACM symposuim on theory o: computing, page 504-511, April 1984.

[DUD 87] A. DUDA, G. HARRUS, Y. HADDAD et G. BERNARD
Estimating global time in distributed systems
Proc. 7th Int. Conf. on Distributed Computing Systems. Berlin 1987.

[ELL 73] C.E. ELLIGSON, R.J. KULPINSKI ,
Dissemination of system time.
IEEE transaction on comm., vol corn-21, N°5, page. 605-623, MAi 1973.

[FAN 88] A. FANTECHI, S. GNESI, C. LANEVE,
Comparing LOTOS and Estelle formal desciption,
Research into networks and distributed applications R. SPETH (editor), Elsevier science publishers B.\ (North Holland).
ECSC, EEC, Brussels and Luxembourg, 1988.

[FAN 90] L. FANCELLI,
Méthode de conception mixte asynchrone/synchrone pou la réalisation de systèmes de commande de processu discontinus.
These de docteur en sciences. université Nice-sophia-antipoiis, juillet 90.

[FID 89] C. J. FIDGE
Dynamic analysis of event orderings in Message-Passing Systems.
Thesis of doctor of philosophy, Department computer
science, the Australian National University, Mar 89.

[GRO 89] R. GROZ,
Verification de propriétés logiques des protocoles r systemes répartis par observation de simulations, CNF LANNION, JANV 89.

[GUS 89] R. GUSELLA et S. ZATTI
The accuracy of the clocks synchronization achieved by TEMPO in Berkley UNIX 4.3 BSD
IEEE Trans. Sof. ENG., Vol. 15, n°7, July 1989.

[HAB 88] D. HABAN, W. WEIGEL,
Global events and global breakpoints in distributed systems.
Proceeding 21th hawai ACM-IEEE, International conference on system services, pp:166-175, 1988.

[HAD 88] Y.Z. HADDAD
Performances dans les systèmes repartis: des outils pour les mesures.
These de Doctorat en Science, n° d'ordre 687, Université de Paris-Sud, Centre Orsay, septembre 1988.

[HEL 87] J.M. HELARY, C. JARD, N. PLOUZEAU, M. RAYNAL
La détection de propriétés stables dans les applications distribuees
Rapport de recherche n°628, INRIA-RENNES, Fevrier 1987.

[HEL 85] J.M. HELARY, C. JARD, M. RAYNAL.
Controlling knowledge transfers in distributed algorithmes: application to deadlock detectian;
rapport de recherche n°483, INRIA, mars 85.

[ISO$_1$ 89] ISO/IEC JTC l/SC 21
Information retrieval transfer and management for ISO.
secretariat: USA(ANSI)
TITLE:WORKINGdraft for a tutorial for Estelle, Aout 1989.

[ISO$_2$ 891 ISO 9074
Information processing systems.
Open systems interconnection. estelle(formal description technique based on an extended state transition model),
1989(E).

[JARa87] C. JARD, J.P. COURTIAT, P. DEMBINSKI, R. GROZ
ESTELLE: un langage IS0 pour les algorithmes distribués et les protocoles.
TSI vol 6 n°2 1987.

[JARb87] C. JARD et 0. DRISSI-KAITOUMI
Deriving trace checkers for distributed systems
IRISA, Publication interne n°347, fevrier 1987.

[JAR 88] C. JARD, R. CROZ, J. F. MONIN
Development of VEDA: a prototyping tool for algorithms.
In IEEE trans. on software Engeneering; March 1988.

[JEZ_ 89] J. M. JEZEQUEL, C. JARD
Un compilateur estelle multiprocesseurs pour l'experimentation d'algorithmes distribues sur machines paralleles.
IRISA, RI n°153, Janv 89.

[JEZ_ 89] J.M JEZEQUEL
Outils pour l'experimentation d'algorithmes distribues sur machines paralleles.
These, université de Rennes, octobre 1989.

[JEZ 91] J. M. JEZEQUEL, C. JARD,
L'experimentation d'algorithmes distribues sur machines paralleles avec Echidna,
IRISA, publication interne n°603, sept 91.

[JOH 78] S. C. JOHNSON,
YACC: Yet Another Compiler Compiler,
Computer science technical report N32, BULL laboratories, Murray Hill, N.J, 1978.

[KAD 89] B. KADRI
Etude et implementation des algorithmes de contrôle distribues: interblocage et terminaison.
these de magister, USTHB, 1989

[KOP 87] H. KOPETZ et W. OCHSENREITER
Interval measurements in distributed real time systems
CH2439-8/87/0000/0292$01.00(c)1987 IEEE.

[LAI 87] Ten H. LAI, Tao H.YANG
On distributed snapshots.
Information processing letters, 25 may 87.
North holland.

[LAM 78] L. LAMPORT
Time, clocks, and the ordering of events in a distributed system
Communication of the ACM, Vol. 21, n°7, pages 558-565, July 1978

[LAM 85] L. LAMPORT, K. M. CHANDY
Distributed snapshots: determining global states of distributed systems.
ACM Trans. on Comp. Systems, Vol. 3, n° 1, February 1985, pages 63-75.

[LAN 883 C. LANEVE, A.FANTECHI, S. GNESI
A method to compare formal protocol specification ,
Proceeding of the first europeen conference on information technology organisational systems, Athenes, Greece, 16-20 may 1988.

[LAN 89] C. LANEVE, S. GNESI, A.FANTECHI
Two standards means problems: A case study on formal protocol descriptions.
North holland computers standards & interfaces n°9, 1989.

[LEU 89] E. LEO, A.SCHIPER,
Technique de deverminage pour programmes paralleles,
Ecole polytechnique fédérale, departement informatique CH 1015 Lausanne,
Rapport interne 89/01, Mai 1989.

[MAT 89] F. MATTERN
Virtual and global states of distributed systems
Parallel and Distributed Algorithms, Elsevier Science Publishers B.V. (North-Holland), 1989.

[MAH 88] R. MAHMOUD-BACHA
Etude et implementation des algorithmes de contrôle distribues: exclusion mutuelle et election.
these de magister, USTHB, 1988.

[MEN 82] M. Menashe,
Analyse de réseaux de petri temporises et application avec les sytèmes distribués.
These de docteur ingenieur, universite de Paul SABATIER Toulouse, Nov 82.

[MIL 85] B. P. MILLER,
Parallelism in distributed programs: measurement and prediction
Computer Science Technical Report n°574,
Univ. of Wisconsin Madison, 1985.

[MIL 86] B.P. MILLER, C.MACRANDER, S. SECHREST,
A distributed programs monitor berkeley UNIX,
Software-practice and experience vol 16(2) Feb 86.

[MILa 88] B.P. MILLER, J.D. CHOI,
Breakpoints and halting in distributed programs,
8th international conference on distributed computing system, Juin 88.

[MILb 88] B.P. MILLER
DPM: A measurement system for distributed programs
IEEE transactions on computers, vol 37 n°2, 1968.

[MIL 90] B. P. MILLER, M. CLARCK, J. HOLLINGSWORTH,
s. KIERSTEAD, S. LIM, T. TOUZEWSKI.
IPS-2: The second generation of parallel program measurement systems.
IEEE transactions on parallel and distributed computing 1990.

[MIL 80] R. MILNER
CCS : A calculus of communicating System
lecture notes in computer science-92
Edited by G.Goos, J.Hartmanis, Springer-verlag, Berli
1980.

[RAY 851 M. RAYNAL
Algorithmes distribués et protocoles.
Edition Eyrolles, 1985

[RAY 90] M.RAYNAL,
Order notions and atomic multicast in distribute
systems: short survey,
Publication interne n°524, mars 1990.

[RAYa 91] M. RAYNAL, M. MISUNO, M.L. NEILSON
Synchronization and concurrency measures fo.
distributed computation.
Publication interne N°610, oct 1991, IRISA.

[RAYb 91] M. RAYNAL,
La communication causale dans les systèmes répartis,
First maghrebin symposium on programming and systems.
Oct 21-23 1991, Algiers.

[RAYc 91] M. RAYNAL,
La communication et le temps dans les réseaux é
systèmes repartis,
Edition Eyrolles 1991, 256 pages.

[SEG 83] Z. SEGAL, A. SINGH, R. SNODGRASS, A.K. JONES,
An Integrated instrumentation Environment r
Multiprocessors
IEEE Trans. on Comp., Vol.C-32, n°1,jan 83.

[SNO 88] R. SNODGRASS,
A relational approach to monitorinq compl
systems
ACM Trans. on computer system, Vol. 6,
n° 2, Mai 1988, pp:157-196.

[SRI 85] T.K. SRIKANTH, S. TOUEG,
Optimal clock synchronization.
IN 4th annual ACM symposuim principles of distribute
computing. pages 71-86, August 1985.

[VIS 89] C. VISSERS, M. DIAZ
SEDOS: Designing open distributed systems
IEEE software Nov 1989.

[VUO 88] S. T. VUONG, A. C. LAU, R. I. CHAN
Semiautomatic implementation of protocoles using an estelle-C compiler
IEEE transactions on software ingeneering
vol 14 n°3 March 88,

[YAN 87] C. Q. YANG et B. P. Miller,
IPS: An interactive and automatic performance measurement tool for parallel and distributed programs
Proc. of 7th international conf. on distributed comp. syst.
IEEE comp. society Berlin FRG, sept. 1987, pp21-25.

[YAN 88] C. Q. YANG et B.P. MILLER
Critical path analysis for the execution of parallel and distributed programs.
9th international conference on distributed computing systems June 88.

[WUU 84] G.I.J. WUU, A.J. BERNSTEIN,
Efficient solutions to the replicated log and distionnary problems.
Proceeding, 3rd ACM symopsuim on PODC, PP:233-242, 1984.

[WYB 88] D. WYBRANIETZ et D. HABAN
Monitoring and performance measuring distributed systems during operation
ACM, 1988, pp:197-205.

A N N E X E A

GRAMMAIRE ESTELLE

Cette annexe présente la grammaire Estelle telle qu'elle est donnée dans [ISO 89] en éliminant les cas non implémentés (i.e: Estelle statique). Les termes et symboles entre suillemets représentent les mots et symboles reserves.
L'axiome est représenté en majuscule.

```
actual module parameter = expression

actual module parameter_list= actual_module_parameter
                              {"," actual_module_parameter }

actual parameter = expression¦variable_access
                 ¦  procedure-identifier
                 ¦ function identifier

actual parameter list= "("actual parameter {"," actual parameter
                       }")"

adding-operator = "+" ¦ "-" ¦ "or"

all_statement = "all" ( domain_list | module_domain )
                "do" statement

any clause = "any" domain list "do"

array typ = "[" index typ { "," index typ } "]" "of"
            component typ

array variable = variable-access

assignement statement = (variable acess ¦ function-identifier )
                        ":=" expression

attach statement= "attach" external-ip "to" child-external ip

base typ = ordinal type

block = constant definition-part
        type-definition part
        variable declaration part
        procedure_and_function_declaration_part
        statement-part

body-definition = declaration part
                  initialisation part
                  transition declaration part

body identifier = identifier
```

```
boolean-expression = expression

case-constant = constant

case-constant-list = case-constant { "," case_constant }

case-index = expression

case-list-element = case-constant-list ":" statement

case-statement = "case" case-index "of" case-list element
                 { ";" case-list-element } [";"] "end"

component-type = type-denoter

component-variable = indexed-variable | field-designator

component-statement = "begin" statement "end"

constant = [sign] (unsigned-number | constant-identifier)
         | string

constant-definition = identifier "=" constant

constant-definition-part = { "const" constant-definition ";"
                             { constant-definition ";" } ]

constant-identifier = identifier

channel-bloc = +{ interaction_group }

channel definition = channel_heading channel-bloc

channel heading = "channel" identifier "(" role-list ")" ";"

channel identifier = identifier

child-external ip = module-variable "." external.-ip

clause group = [provided-clause ]
               [ from-clause ]
               [to-clause ]
               [ any-clause]
               [delay-clause]
               [when-clause ]
               [ priority-clause]

connect_ip = child-external-ip | internal-ip

connect-statement ="connect" connect-ip "to" connect-ip

declaration-part = {declarations}
```

```
declarations = constant_ definition-part
             | type_definition_part
               variable-declaration-part
               channel-definition
             | module_geader_definition
               module-body-definition
               interaction-point-declaration--part
               module-variable-declaration_part
               state-definition-part
               state-set-definition-part
               procedure-function-declaration-part

delay-clause = "delay" "(" ( expression "," expression
                           | expression "," "*"
                           | expression )
                     ")"

domain list = identifier list ":" ordinal typ
              { ";" identifier-list ":" ordinal-typ }

domain typ = type identifier

else_part = "else" statement

entire variable = variable identifier

enumered-typ = "(" identifier-list ")"

expression = simple-expression
             [relation-operator simple-expression ]

external_ip = interaction_point_reference

factor = variable-access
       | unsigned-constant
       | function-designator
       | set constructor
       | "(" expression ")"
       | "not" factor

field-designator = record_variable "." field specifier
                 | field designator identifier

field designator_identifier = identifier

field specifier = identifier

field_list = [ ( fixed-part [ ";" variant-part ]
               | variant part
             ) [";"]
           1

fixed-part = record-section { ";" record-section }
```

```
final_value = expression

for-statement = "for" control_variable ":=" initial-value
                ("to" | "downto") final-value "do" statement

formal parameter-list = "(" formal-parametersection
                        {";" formal-parameter-section } ")"

formal-parameter-section = value_parameter_specification
                         | variable_parameter_specification
                         |   procedural-parameter-specification
                         |   functional-parameter-specification

function block = block

funstion_declaration = function _heading_ ";" diractive
                     | function_identification ";" function--block
                     | function-heading ";" function-block

function-designator = functionidentifier [actual_parameter_list]

functionheading = "function" function-identifier

function identifier = identifier

functional parameterspecification = function-headinq

from clause = "from" from list

from-element = state-identifier | state_set_identifier

from list = from element { "," from element }

header-identifier = identifier

identifier-list = identifier ("," identifier )

if-statement = "if" boolean-expression "then" statement
               [ else_part]

index--expression = expression

index typ = ordinal typ

indexed_variable = arrav variable "[" index-expression
                   { "," index expression } "]"

initial value = expression

index typ list = index typ { "," index typ }

init statement = "init" module variable "with"
                 body-identifier
                 ["(" actual module parameter list ")"]
```

```
initialization part = {"initialize" transition-group }

interaction argument identifier = identifier

interaction argument list =
              "(" interaction-argument-identifier
             {"," interaction-argument-identifier } ")"

interaction definition = identifier
         [ "(" value-parameter-specification
         {"," value Parameter specification } ")" ] ":"

interactiongroup = "by" role identifier [ "," role identifier1
                       ";" + {interaction_definition}

interaction-identifier = identifier

interaction_point declaration-part = "ip"
                          +{ interaction-point-declaration ";"}

interaction point identifier = identifier

interaction-point _reference = interaction-point-identifier
           [ "[" index_expression {"," index_expression } "]" 1

interaction_point_type=channel_identifier "(" role-identifier ")"

interaction reference = interaction point reference "."
                        interaction-identifier

internal ip = interaction point reference

ip index = constant ¦ variable identifier

member designator = expression [ ".." expression ]

module body definition= "body" identifier "for" header identifier
               ":" (body definition "end" ";" )

module_domain = identifier ":" header-identifier

module header definiton = "module" identifier
                [ "("parameter_list")"] ";"
                [ "ip" +{interaction point declaration ":" }]
                "end" " ; "

module variable = module variable identifier
                ¦ module_variable_identifier "[" index expression
                          [ "," index expression 1 "]"

module variable declaration=identifier list ":" header identifier
          ¦ identifier list ":" array "[" index typ list "1" "of"
                                header-identifier
```

```
module_variable declaration_part =
                    "modvar" +{ module-variable-declaration ";" }

module_variable_identifier = identifier

multiplying_operator = "*" ¦ "/" ¦ "div" ¦ "mod" ¦ "and"

new_ordinal_typ = enumered-typ ¦ subrange_typ

new_structured_typ = ["packed"] unpacked-structured-typ

new typ = new ordinal typ ¦ new structured typ

ordinal-typ = new ordinal-typ ¦ ordinal-typ-identifier

ordinal typ identifier = identifier

output-statement = "output" interaction-reference
                   [ actual parameter list ]

parameter list = value parameter-specification
                 { ":" value parameter specification }

priority-clause = "priority" priority-const

priority const = unsigned-integer ¦ constant identifier

procedure_parameter specification = procedure-heading

procedure and function declarartion_part =
                  {( procedure-declaration
                     functiondeclaration ) ";" }

procedure block = block

provided clause = "provided" ( boolean_expression )

procedure heading = "pure" "procedure" identifier
                                       [formal parameter list1

                    "procedure" identifier
                                       [formal parameter list]

procedure identification= "pure" "procedure" procedure identifier
                        ¦ "procedure" procedure identifier

procedure identifier = identifier

procedure statement=procedure identifier([actual-parameter list])

real typ identifier = identifier

record--section = identifier_list ":" type denoter
```

```
record typ = "record" field list "end"

record variable = variable access

record variable list = record-variable { "," record variable }

relational. operator = "=" | "< >" | "<" | ">" | "≥" |"≤" | "in"

repetitive-statement = when-statement
                     | for-statement
                     | all-statement

result typ = identifier

role identifier = identifier

role-list = identifier "," identifier

set_constructor = "[" [ member_designator
                        { "," memher desianator } 1
                  "1"

set typ = "set" "of" base typ

scale factor = signed-integer

simple statement = | attach statement
                 | connect-statement
                 | init statement
                 | output-statement

sign = "+" | "-"

signed-integer = [siqn] unsigned-integer

signed number = signed_integer | signed real

signed-real = [sign] unsigned-real

simple expression = [sign] term {adding operator term }

simple statement =
                 | assignment- statement
                 | pracedure statement

simple typ = ordinal typ
           | real_typ_identifier

simple typ identifier = identifier

SPECIFICATION = "specification" identifier ";"
                body definition
                "end" " ; "
```

```
statement = simple_statement | structured-statement

statement part = compound stat.ement

statement sequence = statement f ";" statement)

structured statement = compound statement
                     | conditional-statement
                     I repetitive statement

structured-typ = new-structured-typ | structured-typ-identifier

structured typ identifier = identifier

subrange-typ = constant ".." constant

state_definition_part = "state" identifier-list ";"

state-identifier = identifier

state_set_ constant="["state_identifier {","state_identifier } "]"

state-set-definition = identifier "=" state-set-constant

state set definition part="stateset" +{state set definition ";" }

state-set identifier= identifier

tan field = identifier

taq typ = ordinal typ identifier

term = factor { multiplying onerator factor }

type--definition = identifier "=" typ_denoter

type definition part = [ "type" type definition ";"
                                { type-definition ";" } ]

typ denoter = identifier | new type

to clause = "to" to-element

to element = "same" | state identifier

transition-block = constant_ definition-part
                 | type definition part
                 | variable declaration-part
                 I procedure function declaration part
                 | statement-part

transition declaration = "trans" transition group
```

```
transition-declaration-part  =  {  transition_declaration  }

transition-group = + { clause-group transitionblock ";" }

unpacked-structured-typ  =  array-typ  |  record-typ  |set_typ

unsigned_constant  =  unsigned-number
                    | string
                    |  constant-identifier
                    |  "nil"

unsigned_numder  =  unsigned-integer  |  unsigned-real

unsigned  real  =  unsigned  inteaer  "."  fractional  nart
                                    [ "e" scale-factor ]
                 |  unsigned  integer  "e"  scale  factor

unsined  integer  =  nb

value-parameter-specification  =  identifier_list ":" identifier

variable  access  =   entire variable
                    |  component-variable
                    |  identified-variable

variable_declaration=  identifier-list  ":" typ_denoter

variable_declaration_part = [ "var" variable  declaration  ";"
                                { variable_declaration ";" } ]

variable  identifier  =  identifier

variable-parameter-specification  =  identifier  list  ":"  identifier

variant  =  case  constant  list  ":"  "("  field list ")"

variant-part=  "case"  variant-  selector  "of"  variant  {  ";"  variant}

variant  selector = [ tag field ":" 1 tao typ

when  clause  =  "when"  when  ip  reference  "."  interaction  identifier
                 [interaction argument list ]

when  ip  reference  =  interaction  point  reference  [  "["  ip  index
                    { "," ip index } "]" ]

while_statement = "while"  booleanexpression  "do"  statement
```

A N N E X E B

```
specification roberts:   (* election sur un anneau *)
const nbsites=2;
      nbsites_1=1;
type  num_site=1..nbsites;
var  crd, g:integer;
channel  liaison  (sortie,entree);
by  sortie:  elir (numero:integer);
             eu   (numero:integer);
             termine (nb1:integer);
by entree:;

module typ_mod (mon num:integer);
ip s:liaison(sortie);
   e:  liaison(entree);
end:

body typ_bod for typ_mod;

state  participant,  nonparticipant, fin, elu;

var   coordonateur: integer;

initialize   to   nonparticipant
 begin
      view (%S 'mon  numero', %I mon_num);
      coordonateur:=0;
 end:

trans   from   nonparticipant
        provided mon num = 1
        priority  0
        to  participant (*provoquer  election  *)
        begin
          a:=  getim loc:
          view (%S 'provoquer election '):
          output  s.elir(mon_num);
        end ;

trans   from  nonparticipant
        when  e.elir
        to   participant
        begin
         view (%S 'mon num ', %I mon_num);
         view  (%S  'num recu ', %I   numero);
         if  numero < mon_num     then    output  s.elir(mon_num)
                                  else    output  s.elir(numero);
        end:

trans   from   participant
       when  e.elir
```

```
        to  participant
         begin
          view  (%S  'mon  num ', %I  mon_num);
          view  (%S  'num  recu ', %I  numero):
          if  numero  <  mon_num  then  output  s.elir(mon_num)
                                 else  output  s.elir(numero);
         end;

trans    from   participant
         when  e.eu
        to  fin
         begin
              view(%S 'elu= ', %I numero);
              terminate;
         end:


trans    from   participant
         when  e.elir
         provided  (mon  num  =  numero)
         PRIORITY   10
         to  elu
         begin
            view(%S  ' je  suis  elu', %I  mon_num);
            output  s.eu(2);
            terminate,;
         end;
end;

modvar  noeud  :  array[l..nbsites]  of  typ_mod;

initialize
type  s=  char:
var  x,n,h:inteqer;
begin
   all  i:  1..nbsites do init noeud[I] with typ_bod(i);
   all  i:  1.. nbsites_1  do  begin  connect  noeud[i].s  to  noeud[i+1].e
   connect  noeud[nbsites].s  to  noeud[1].e;
end:
end.
```

```
(** 23/01/91 (USTHB-CERIST) - Estelle Pascal Interpreter (EPI) -
    Version 1.1 **)

   (*** code derived from specification roberts ***)

[inherit('sys$library:starlet')]
MODULE roberts (output);

 %include ' CST.PAS '
 %include ' TYP.PAS '
 %include ' CTRL.PAS '
 %include ' VAR.PAS '
 %include ' PROCUSER.PAS '

 %include ' RUNTIM_KERNEL.PAS '
 (external) procedure init_sub_inst ($W:$ptrmod);extern;
 [external] procedure view (t1:char;int:integer:=0;rel:real:=0;
                                              str:$pac1:=' ');exter
 [external] function randomint(t1,t2:integer):integer;extern;
 [external] function getim_glb:double;extern;
 [external] function getim_loc:double;extern;

PROCEDURE ERROR(no_line :integer);
beain
   writeln('ERROR on line ', no-line,' :indexes out of range');
   $exit;
end:

[global]FUNCTION $garde 1($W:$ptrmod):integer;
begin
$garde 1        := -1;
Delay           := nil:
$tab_trans[1].nb_any := 0;
$W^.ctx^.ctxt.idbody := 1:
$garde_1 := 0;
end;

[global]PROCEDURE $trans_1 ($W:$ptrmod);
begin
$W^.ctx^.ctxt.typ_bod.etat := nonparticipant:
VIEW('S', str := 'mon numero' );
VIEW('I', int := $W^.ctx^.ctxt.typ_bod.para.mon_num);
VIEW('S', str:= '[EOB]') ;
$W^.ctx^.ctxt.typpbod.vari.coordonateur := 0 ;
end:
```

```
[global]FUNCTION $garde_2($W:$ptrmod):integer;
begin
$garde_2        := -1;
Delay           := nil;
$tab trans[2].nb_any := 0;
$W^.ctx^.ctxt.idbody  :=  1;
if $W^.ctx^.ctxt.typ_bod.etat IN ([ nonparticipant]) then
beqin
 IF $W^.ctx^.ctxt.typ_bod.para.mon_num = 1  then
begin

$garde_2 := 0;
end:
end:
end:

[global]PROCEDURE $trans_2 ($W:$ptrmod);
beqin
$W^.ctx^.ctxt.typ_bod.etat  :=  participant:
$W^.ctx^.ctxt.typ_bod.vari.a :=  getim_loc  ;
 VIEW('S', str  :=  'provoquer  election ' );
 VIEW('S', str:= '[EOB]') ;
 $iip_0 :=1+0;
$m_o.inter := $elir;
$m_o.elir.numero :=$W^.ctx^.ctxt.typ_bod.para.mon_num;
SEND($W^.ctx^.ips[$iip_0],  $m_o)  ;
end:

[global]FUNCTION $garde_3($W:$ptrmod):integer;
begin
$garde_3    :=    -1;
Delay           :=  nil;
$tab_trans[3].nb_any := 0;
$W^.ctx^.ctxt.idbody      := 1;
if $W^.ctx^.ctxt.typ_bod.etat  IN  ([  nonparticipant])  then
begin
$iip 0 :=1+1;
If verif message($W^.ctx^.ips[$iip_0], $elir) then
beain

$garde 3 := 0:
end:
end;
end:

[global]PROCEDURE $trans_3 ($W:$ptrmod);

begin
$W^.ctx^.ctxt.typ_bod.etat  :=  participant;
$iip 0 :=1+1;
get_message($W^.ctx^.ips[$iip_0], $m_i);
VIEW('S', str :=  'mon num ' );
VIEW('I', int := $W^.ctx^.ctxt.typ_bod.para.mon  num);
VIEW('S', str:= '[EOB]') ;
```

```
VIEW( 'S', str :=  'num recu ' );
VIEW( 'I', int  := $m_i^.elir.numero);
VIEW('S', str:=  '[EOB]')  ;
if $m_i^.elir.numero  <  $W^.ctx^.ctxt.typ_bod.para.mon_num  THEN
 begin
 $iip_0 :=1+0;
 $m_o.inter := $elir;
 $m_o.elir.numero :=$W^.ctx^.ctxt.typ_bod.para.mon_num;
 SEND($W^.ctx^.ips[$iip_0], $m_o)
end
else
begin
$iip_0 :=1+0;
$m_o.inter := $elir;
$m_o.elir.numero :=$m_i^.elir.numero;
SEND($W^.ctx^.ips[$iip_0], $m_o)
end  ;

end:

[global]FUNCTION $garde_4($W:$ptrmod):integer;
begin
$garde_4       := -1:
Delay           := nil:
$tab_trans[4].nb_any := 0;
$W^.ctx^.ctxt.idbody       := 1;
if $W^.ctx^.ctxt.typ_bod.etat  IN  ([  participant])  then
begin
$iip_0 :=1+1;
If verif_message($W^.ctx^.ips[$iip_0], $elir) then
begin
$garde_4 := 0:
end:
end;
end:

[global]PROCEDURE $trans_4 ($W:$ptrmod);
begin
$W^.ctx^.ctxt.typ_bod.etat   :=   participant;
$iip_0 :=1+1:
get_message($W^.ctx^.ips[$iip_0], $m_i);
VIEW('S', str :=  'mon num ' ) :
VIEW('I', int := $W^.ctx^.ctxt.typ_bod.para.mon_num);
VIEW('S', str:= '[EOB]') :
VIEW('S', str :=  'num recu ' );
VIEW('I', int := $m_i^.elir.numero):
VIEW('S', str:= '[EOB]')  ;
if $m_i^.elir.numero < $W^.ctx^.ctxt.typ_bod.para.mon_num  THEN
beain
$iip_0 :=1+0;
$m_o.inter  := $elir;
$m_o.elir.numero :=$W^.ctx^.ctxt.typ_bod.para.mon_num;
SEND($W^.ctx^.ips[$iip_0], $m_o)
end
```

```
else begin
$iip 0 :=1+0;
$m_o.inter := $elir;
$m_o.elir.numero :=$m_i^.elir.numero;
SEND($W^.ctx^.ips[$iip_0], $m_o)
end ;

end:

[global]FUNCTION $garde_5($W:$ptrmod):integer;
begin
$garde_5   :=   -1;
Delay          := nil;
$tab_trans[5].nb_any := 0;
$W^.ctx^.ctxt.idbody      := 1;
if $W^.ctx^.ctxt.typ_bod.etat 'IN ([  participant])  then
begin
$iip_0 :=1+1;
If verif_message($W^.ctx^.ips[$iip_0], $eu) then
beain

$garde_5 := 0;
end;
end:
end:

[global]PROCEDURE  $trans_5  ($W:$ptrmod) ;

begin
$W^.ctx^.ctxt.typ_bod.etat := fin:
$iip_0 :=1+1;
get_message($W^.ctx^.ips[$iip_0], $m_i);
VIEW('S', str :=  'elu= ' );
VIEW('I', int := $m_i^.eu.numero);
VIEW('S', str:= '[EOB]') ;
TERMINATE($W^.no_mv) ;
end:

[global]FUNCTION $garde_6($W:$ptrmod):integer;
begin
$garde_6      := -1;
Delay          := nil:
$tab_trans[6].nb_any := 0;
$W^.ctx^.ctxt.idbody      := 1:
if $W^.ctx^.ctxt.typ_bod.etat  IN  ([  participant])  then
begin
$iip_0 :=1+1;
If verif_message($W^.ctx^.ips[$iip_0], $elir) then
begin
 IF ($W^.ctx^.ctxt.typ_bod.para.mon_num = $m_i^.elir.numero) then
beain

$garde_6 := 10;
end;
```

```
end:
end:
end:

[global]PROCEDURE $trans_6 ($W:$ptrmod);

begin
$W^.ctx^.ctxt.typ_bod.etat := elu;
$iip_0 :=1+1;
get_message($W^.ctx^.ips[$iip_0], $m_i);
VIEW('S', str :=  ' je suis elu' );
VIEW('I', int := $W^.ctx^.ctxt.typ_bod.para.mon_num);
 VIEW('S', str:= '[EOB]') :
 $iip_0 :=1+0;
$m_o.inter := $eu;
$m_o.eu.numero :=2 ;
SEND($W^.ctx^.ips[$iip_0], $m_o) ;
  TERMINATE($W^.no_mv) ;

end;

[global]FUNCTION $garde_7($W:$ptrmod):integer;
begin
$garde_7      := -1;
Delay          := nil;
$tab_trans[7].nb_any := 0;
$W^.ctx^.ctxt.idbody     := 0:
$garde_7 := 0;
end ;

[global]PROCEDURE $trans_7 ($W:$ptrmod);

TYPE
  s= char ;

VAR
 x ,n ,h : integer ;

begin
FOR $i_00 := 1 TO roberts_nbsites DO
begin
IF NOT (( $1_00 IN [ 1.. roberts_nbsites])) THEN    ERROR(86);
$imv_0:=1+(ord($I_00)- ord(1));
rec_para.idbody := 1;
rec_para.typ_bod.para.mon_num := $i_00;
creat_instance($W, $MV[$imv_0], 1, $imv_0, 2, 6, 1, $i_00, rec_para)
end  ;

FOR $i_01 := 1 TO roberts_nbsites_1 DO
 begin

IF NOT (( $i_01 IN [ 1.. roberts_nbsites])) THEN    ERROR(87):
$imv_0:=1+(ord($i_01)- ord(1));
$iip_0 :=1+0;
```

```
IF  NOT  (($i_01+1)  IN  [  1..  roberts-nbsites]))  THEN    ERROR(87);
$imv_1:=1+(ord($i_01+1)- ord(1));
$iip_1 :=1+1;
Connect ($imv_0,  $iip_0, $imv_1,  $iip_1) ;
end  ;

IF  NOT  (( roberts-nbsites  IN  [  1..  roberts-nbsites]))  THEN    ERROR(
$imv 0:=1+(ord(roberts nbsites)- ord(1));
$iip_0 :=1+0;

IF  NOT  (((1)  IN  [  1..  roberts-nbsites]))  THEN    ERROR(88);
$imv 1:=1+(ord(1)- ord(1));
$iip_1 :=1+1;
Connect($imv_0,  $iip_0, $imv_1,  $iip_1) ;
end:

[global]procedure  configuration:
begin
  $root:=nil;
  creat_instance($root, $root, 0, 0, 0, 1, 7,  -1, rec_para);

  init_sub_inst($root);
end;

%include 'PROC.PAS '
end.

      (***  end  of  program  code  ***)
```

```
******  CST.PAS


CONST
         (* definition des cst globales de EPI *)

$max_any   = 30;
$max_trans = 50;
$max_proc  = 100;
           (*  definition  des  constantes  de  l'usager      *)

Const
 roberts_nbsites=2  ;
 roberts nhsites 1=1 ;

$max_task = 2;  (* nb max de taches ds le systeme *)

$maxports = 20:



*******  TYP.PAS
                  (*  definition  des  types  *)


TYPE
 roberts_num_site=1 . . roberts_nbsites ;

                (*  variable  declaration  for  roberts  *)


type
$var_roberts_typ=  record
crd,g:  integer  ;
 end;


type
$typ_elir  =  record
numero:   inteaer   end:


type
$typ_eu = record
numero:   i  nteaer  end:


type
$typ_termine  =  record
nbl:   inteaer   end:
```

```
                (*    typ_mod parametres declaration *)

type
$typ mod mod para = record
mon num: integer end:


                (* variable state declaration for typ_bod *)
type
$typ_bod_state = ( participant, nonparticipant, fin, elu);

              (* variable declaration for typ bod *)


type
$var_typ_bod_typ= record
coordonateur: integer ;
 a: double ;
 end:


******      CTRL.PAS

            (* declaration des types qlobaux de EPI *)

$enum mess = ( $elir, Seu , $termine);
$typ_mess = record
id ip : integer:
stamp : inteser:
inter: $enum mess of
  Selir: (elir : $typ_elir);
  $eu: (eu : $typ_eu);
  $termine: (termine : $typ_termine);
end:


$pac1     = varying[80] of char;

$typ pack = packed array [1..15] of char;

$ptrfifo  = ^$typ_fifo;

$ptrip    = ^$typ desc ip;

Sptrmess = ^$typ_mess;

$ptrmod   = ^$typ modules:

                (* fifo queue declaration *)
$typ fifo = record
     m   :$ptrmess;
     svt : $ptrfifo:
end;
```

```
list-mod   =  record
                no_mod  :  integer:
                adr_mod_svt:^list_mod;
              end:

                    (*   interaction   print   declaration   *)

$typ_desc ip  =  record
     typ_con                0..2;
     id, site               integer;
     id-corresp             integer;
     no corresp             integer:
     no-modvcorresp         integer;
     site_corresp           integer:
     qg, qd                 $ptrfifo;
     adr   instance         $ptrmod;
     ip_svt                 $ptrip;
end:


                     (***   block   context   ***)

$typ_interv = 0 . . 100;

       (***   block   context   for   process   roberts   ***)

$ctx_roberts  =  record
     vari          :$var_roberts_typ;
end;

       (***   block   context   for   process   typ_bod   ***)

Sctx  typ  hod  =  record
     etat          :$typ_bod  state:
     para          :$typ  mod  mod  para;
     vari          :$var  typ_bod  typ;
end:
 $ctxt1   =  record
     case idbody: $typ  interv  of
     0 :  (roberts:   $ctx   roberts);
     1 :  (typ hod: $ctx typ bod);
end;
 $ptrctx = ^$ctxbody;
 $ctxbody=  record
     ips           : array [1..$maxports]  of  $ptrip;
     ctxt          :  $ctxt1 ;
end:

$typ_modules  =  record
     num_proc       :  integer;
     no_mv          :  integer;
     site           :  integer;
```

```
    adr_1_ip      : $ptrip;
    nb_ip         :  integer;
    list_s_mod    :  ^list_mod;
    num_1_trans   :  integer:
    nb_trans      :  integer:
    etat_act      :  boolean:
    adr_pere      : $ptrmod;
    site_pere     :  integer;
    ctx           : $ptrctx ;
end;
     (*  declaration  du  type  buffer  de  reception/emission  *)
$e1 = [byte(1)] 0..255;

$e = [byte(2)] 0..65535;

$can = record site : $e1;
             canal1, canal2,canal3: $e;
         end:

$tlistcanaux = array [0..10] of Scan;

$typ para    =  record
                  trace, debug, mesure, determin : $e1;
                  duree : integer:
                  seed  : unsigned;
                  particip:set of 0..255;
            end;
$t1  =record   site : $e1; chan :$e; end;

$buf = record
       emet, recept      : $t1;
       case code: char of
          'M': (info      : $typ_mess;);
          'T' : (no-mod    :  integer:);
          'J': (no-modvarl  :  integer;):
          'I': (no-modvar:   integer;
                no-site, sit-pere : $e1;
                no_per,num per    : integer;
                idbod,nb_ips,nb_trans,first_trans : integer:
                para :$ctxt1;
                );
          'Q' : (id1,noi1,noip1,id2,noi2,noip2 :integer;);
          'P'  'C'. 'A': (no i1, no ipl. no i2 no ip2. id ip1: ir'
          'O' : (id_ip, no-i, no_ip, err : integer;);
          'R': (no fils : integer;);
          'S': (sts1 : integer;
               prcn2: $typ pack:
                liste_canaux1 :$tlistcanaux;
                list para    : $typ para;);
          'L': (nonspes, prcn: $typ_pack;);
          'D': (prcn1 : Styp pack;
          liste_canaux :$tlistcanaux;);
          'K': (sts : integer:
          chan1,chan2, chan3 : $e;);
```

```
          'E': (er    :integer;);
      end:


                  (*  transition  description  *)
$desc  trans  =  record
    num_trans   : integer;
    ind  val  any:  integer;
end;

$trans_typ  =  record
    nb_any    :  integer;
    ind_1_any:  integer;
end;

 $typ_delay = ^typ_d;
 typ_d       = record
                 el ,e2: integer;
              end;

                      (*      globales  variables  declaration  *)

VAR

 $iip 0, $iip_1     :  integer;
 $imv 0. Simv 3     : integer;
 rec_para: $ctxt1;


*******   VAR.PAS


VAR
                      (*       declaration  des  variables  de  l'usager

$i_00: 1..  roberts_nbsites;
$i_01: 1..  roberts_nbsites_1;
 $MV                : [global] array [0..$max_task] of $ptrmod;

                      (*      globales  variables  declaration  *)
 $tab proc          : [global]array [1..$max proc ] of $ptrmod:
 Stab-enable  trans  : [global]array [1..$max_trans] of $desc trans;
 $tab fir trans     : [global]array [1..$max trans] of $desc trans:
 $tab  trans        : [global]array [1..$max_trans] of $trans typ:
 $val anv           : [global]array II..$max anv] of integer:
 Sm i               : [global]$ptrmess:
 $m 0               : [global]$typ mess:
 $root              :  [global]$ptrmod;
 $tete list in.
 $queue_list ip     : [global]$ptrip:=nil;
 $local             : [global]integer;
 Sk                 : [global]integer:=1;
 delay              : [global]$typ  delay;
```

A N N E X E C

ERREURS GEREES PAR LA MACHINE VIRTUELLE REPARTIE

Erreurs gérées par EST:

0 : no indexes defined for interaction point x
1 : double declaration of channel x
2 : undefined role x
3 : x must be an other role
4 : double declaration of module x
5 : double declaration of interaction point x
6 : x undefined identifier of channel on this level
7 : undefined identifier x
8 : x undefined state
9 : x undefined referenced interaction point
10 : undefined constante/type or must be scalaire
11 : undefined referenced interaction x
12 : x undefined variable module
13 : incompatible module type
14 : interaction points referenced in 'connect'/'attach' must be declared using identical channel identifier
15 : interaction points referenced in 'connect' must be declared usina opposites roles
16 : ')' expected . . . too many parameters
17 : double declaration of interaction x
18 : type or constante identifier x not defined
19 : too many indexes used for interaction point x
20 : more indexes defined for interaction point x
21 : x must be indexed
22 : x must be passive (implementation restriction)

23 : INIT, CONNECT, and ATTACH statements must not appears in transition block (implementation restriction)
24 : Restriction of implementation: EXPORT, SYSTEMACTIVITY, SYSTEMPROCESS, ACTIVITY, PROCESS, DISCONNECT and DETACH not implemented
25 : Function or procedure declared demonstrably pure
26 : Procedure must be declared demonstrably pure or variable must be locale
27 : Procedure or function called must be demonstrably pure
28 : undefined procedure or function identifier
29 : Double declaration of procedure or function
30 : actual parametres expected
31 : x must be indexed
32 : too many indexes used for module variable x
33 : Double declaration of module variable
?A . more indexes defined for module variable x
35 : no indexes defined for module variable x
36 : interaction points referenced in 'attach' must be declared usinq identical roles
37 : SAME not allowed in initialize part

Erreurs gérées par le noyau:

1: Interaction point already attached to child
2: Too many processus
3: Inexistant interaction point
4: Interaction point already connected
5: Physical connexion not allowed
6: Message lost
7: Invalid interaction point task
8: Message lost, no correspondant
9: No task correspondant to referenced interaction point
10: No enough space
11: Connected interaction point can't be attached
12: Second interaction point must be child interaction point
13: First interaction point must be Local