THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de

DOCTEUR ingénieur et 3ème cycle

Spécialité : GENIE INFORMATIQUE

par

Alain TARABOUT



UN OUTIL DE RECHERCHE EN SYSTEMES INFORMATIQUES

PRINCIPES ET BILAN DU PROJET OURS



Thèse soutenue le 11 décembre 1978 devant la Commission d'Examen

Président S. KRAKOWIAK

Examinateurs R. BALTER
J. BRIAT
C. KAISER
J. MOSSIERE
J.P. VERJUS

SOMMAIRE

Pages

I. PRESENTATION 1

I.1. Systèmes et machines 1
I.2. Découpage fonctionnel 3
1.3. Processus 4

II. EVALUATION CRITIQUE DU MODELE DE DECOMPOSITION 6

II.1. Traitement de données et contrôle 6
II.2. Niveaux de contrôle 7
II.2.1. L'étoile 8
II.2.2. L'anneau 10
II.2.3. L'automate 12
II.2.4. Synthèse 14
II.3. Outils de construction 15
II.3.1. Unités, modules et machines abstraites . 15
II.3.2. Machines multiprocesseurs 16
II.3.3. Moyens de communication 17

III. PERSPECTIVES 19

IV. CONCLUSION 21

V. BIBLIOGRAPHIE 22

I. PRESENTATION

Le projet OURS a pour objet essentiel le développement d'outils pour la recherche en systèmes.

I.1. Systèmes et machines

D'une manière générale, un système informatique peut être considéré sous deux aspects principaux :

- soit comme l'*interpréteur* d'un langage particulier, grâce auquel sont masquées certaines contraintes du matériel. Le langage définit alors une *machine abstraite* [BRI1] - ou "virtuelle", comme l'on voudra [CRO] - plus facile à manipuler que le matériel brut.
- soit comme un *allocateur* de ressources [HOA1], gérant le matériel pour le compte des utilisateurs et des programmes de service de manière aussi fiable et efficace que possible.

Nous allons voir bientôt que ces deux aspects sont présentés de manière trop restrictive et demandent à être généralisés.

Il est abusif de vouloir privilégier l'un des aspects par rapport à l'autre. Même un système APL [MAU], par exemple, qui apparaît immédiatement sous l'aspect d'un interpréteur du langage APL, doit pouvoir gérer plusieurs utilisateurs en parallèle et fonctionne donc aussi en tant qu'allocateur des ressources mémoire et temps de calcul.

On peut donc dire en général qu'*un système informatique est une entité duale interpréteur/allocateur*.
Il offre, à travers le langage de la machine abstraite qu'il définit, des *fonctions d'accès* aux ressources physiques qu'il gère en les répartissant entre ses utilisateurs. Ces fonctions d'accès apparaissent comme des objets typés : les *ressources logiques* mises en oeuvre par la machine abstraite. Les ressources logiques peuvent être considérées comme des réalisations de *types abstraits* [LIS, JAQ].

Il y a alors analogie entre les ressources physiques et le langage d'une machine "matérielle" d'une part et les ressources logiques et le langage d'une machine abstraite d'autre part. On peut donc définir des systèmes

informatiques en termes d'autres systèmes, c'est-à-dire construire des machines abstraites sur d'autres machines abstraites.

C'est par l'opération d'*abstraction* [DAH] que l'on définit une machine abstraite, c'est-à-dire que l'on spécifie les ressources logiques offerte en interface. Construire une machine abstraite en termes d'autres machine c'est effectuer une opération de *raffinement* [WIR]. On dit qu'on définit un nouveau *niveau d'abstraction* - ou niveau de machine - par ce procédé [DIJ1, DIJ2].

Le terme de machine désignera dans la suite une machine abstraite d'un niveau quelconque, aussi bien qu'une machine réelle. De même, une ressour sera aussi bien une ressource logique qu'une ressource physique.

La partie "interpréteur" d'un système défini par un certain niveau i de machine réalise l'interface entre les ressources fournies à ce niveau et les ressources des machines du niveau inférieur. Son complément indispensable est l'allocateur des ressources du niveau i. Nous retrouvons par ce biais le schéma connu de la répartition des ressources par niveaux [WUL].

Reprenant la présentation au début de ce chapitre de ce qu'est un système informatique, nous dirons maintenant que c'est une entité duale
- interpréteur d'une machine-langage, réalisant l'interface entre les ressources qu'il fournit et celles dont il dispose ;
- allocateur des ressources qu'il fournit à ses utilisateurs.

Remarquons maintenant que les ressources fournies à un certain niveau ne sont pas forcément plus riches ou plus simples à manipuler que les ressources du niveau inférieur. Dans le cas de [BEN], par exemple, on implante une machine "générateur de 10.070 virtuel" sur la machine "SIRIS 7" définie sur la machine 10.070 réelle ; l'utilisateur du générateur de 10.070 virtuel dispose alors d'un jeu d'instruction (celles du 10.070 réel) très appauvri par rapport à celui dont dispose le générate lui-même (les instructions de la machine SIRIS 7).

La notion de niveaux de machine n'implique donc aucune hiérarchie des langages, mais seulement un certain ordre dans la construction.

Il est clair qu'à la base de la construction, on doit trouver la machine réelle.

I.2. Découpage fonctionnel

La notion de niveau d'abstraction est d'un grand secours dans la réalisation de systèmes. En effet, ces derniers étant des programmes complexes et souvent gros [MOS], on a avantage à diviser la difficulté que présente leur réalisation en procédant par étapes au moyen d'abstractions et de raffinements successifs en partant de la description du système à réaliser [DES].

Cette méthode présente en outre l'avantage fréquent de permettre le regroupement dans une machine de base des fonctions communes à différentes sortes de services ; ces derniers sont réalisés dans des machines construites sur la machine commune de base.

L'expérience prouve qu'il est possible en particulier de définir une machine de base qui soit commune à une très grande variété de systèmes [BEK, BÖS, ORG, WUL]. On appelle *noyau* une telle machine. Les types de noyaux actuellement connus paraissent souvent différents, mais présentent néanmoins de nombreux points communs au niveau de la gestion des ressources physiques. Ce niveau peut être constitué en noyau élémentaire. L'un des buts de notre projet a été de construire un tel noyau ; nous l'avons appelé MAS (MAchine Système).

On va donc construire des systèmes par niveaux de machines édifiées les unes sur les autres (on dit encore par inclusions les unes dans les autres) Les ressources logiques fournies par les machines des divers niveaux sont définies comme des réalisations de modèles d'objets, munis des actions portant sur ces objets (les fonctions d'accès). Il est donc naturel de structurer les programmes qui réalisent un niveau de machine en *modules*, chaque module étant défini [PAR1] comme l'association d'objets et de fonctions manipulant ces objets, ce qui recoupe la notion de ressource logique.

En fait, on peut regrouper dans un même module la réalisation de plusieur types de ressources logiques, ou bien placer dans des modules distincts des fonctions pouvant être utilisées en commun par d'autres modules. Le découpage d'un niveau en modules ne coïncide pas nécessairement avec le découpage en ressources logiques, mais il en découle pourtant de manière naturelle [LIN].

Les divers avantages tirés de la décomposition en module sont connus [DES, JAK, MOS] et nous n'y reviendrons pas.

Un niveau sera donc réalisé par un ou plusieurs modules ; de même un module pourra intervenir dans la réalisation de plusieurs niveaux [PAR2, HAB2]. Nous avons réalisé le noyau MAS en terme de modules.

Bien sûr, il n'existe pas d'algorithme de décomposition d'un système en niveaux d'abstraction et en modules. Mais la mise au point de méthodes de décomposition et la réalisation d'outils de construction comptent parmi les conditions essentielles du succès d'une entreprise de productio de systèmes.

Les outils d'assemblage de modules et de construction de machines que nous avons réalisés dans le cadre du projet OURS répondent à cet objectif. Le noyau MAS intègre ces outils et est lui-même construit par leur intermédiaire. Les outils servent évidemment à la construction des machines supportées par MAS, qui est ainsi un noyau adapté à la construction de machines plus évoluées.

Nous venons d'évoquer l'aspect statique de l'architecture des systèmes en termes des composants "machines" et "modules". Venons-en maintenant à l'aspect dynamique des activités qui se déroulent dans les systèmes.

I.3. Processus

Les systèmes modernes sont le siège d'activités parallèles : d'une part ils tiennent compte des possibilités matérielles de traitements simultanés dans les processeurs de la machine réelle, lesquels peuvent fonctionner en parallélisme vrai ; d'autre part ils supportent plusieurs usagers en même temps.

Ces activités se décomposent en une famille d'entités, chacune d'elles s'exécutant séquentiellement, dénommées *processus*.

Considérons un usager soumettant un programme à une certaine machine M, c'est-à-dire une suite d'instructions du langage interprété par M. Chaque instruction donne éventuellement lieu à une cascade d'interprétations dans certaines des machines sur lesquelles M est construite.

L'exécution d'une instruction de M provoque donc l'exécution d'un certain nombre de fonctions localisées dans des modules de machines incluses dans M.

On peut considérer que l'exécution de ces fonctions se fait sous l'autorité d'un même processus utilisateur, avec appels successifs et empilement des contextes d'exécution propres à chaque module dans une pile associée au processus, et dépilement au retour ; c'est le schéma *d'appel procédural* [BRI2, CHE].

Nous avons préféré considérer, pour les raisons exposées dans [EST] (gain de place en mémoire, gain de temps à l'exécution), que chaque module est le support de *processus cycliques*, chacun étant chargé d'une fonction du module. Les appels intermodules sont alors réalisés par transmission de *messages* entre processus cycliques.

Ce schéma, qui est celui de HYDRA [WUL], permet de considérer qu'un processus se comporte comme un producteur et consommateur de messages. L'aspect dynamique d'un système fonctionnant suivant de tels principes est celui d'un *réseau de processus communiquant par messages*.

Nous avons introduit la notion d'*unité* comme étant la projection statique de la notion de processus. En d'autres termes, une unité est le support d'un processus à l'intérieur d'un module.

Il est bien évident que, quel que soit le schéma d'appel intermodules, les processus ne sont pas totalement indépendants, soit qu'ils coopèrent au sein d'une même fonction, soit qu'ils partagent des ressources d'un niveau inférieur.

Les relations interprocessus posent donc des problèmes de synchronisation. De nombreuses études y ont été consacrées (signalons [DIJ3, BRIN, HOA2, CAM, ROB] pour diverses solutions à différents niveaux), mais on s'est

trop souvent borné, dans les réalisations de quelque ampleur, à "résoudr les problèmes de synchronisation que l'on pouvait rencontrer par l'utilisation des primitives P et V sur un sémaphore [DIJ3] - ou un mécanisme analogue - avec tous les dangers d'interblocage que cela comporte, sans parler du problème de l'adressage du sémaphore commun.

Nous avons voulu reprendre le problème à la base et montrer que la synchronisation entre processus appartenant à des modules différents s'exprimait de manière naturelle à l'intérieur du modèle de communicatio par messages, et qu'il suffisait de contrôler les transitions entre processus pour être à même de résoudre les problèmes classiques de synchronisation dans un système.

II. EVALUATION CRITIQUE DU MODELE DE DECOMPOSITION

Nous allons tenter de faire une évaluation qualitative du modèle de décomposition de systèmes informatiques que nous avons utilisé dans le cadre du projet OURS, en mettant l'accent principal sur les problèmes de contrôle.

Les outils qui permettent la mise en oeuvre de cette décomposition sont présentés dans [VAT2] et illustrés dans [MAI2] ; nous supposerons que le lecteur a pris connaissance de ces deux ouvrages.

II.1. Traitement de données et contrôle

Nous avons adapté la méthode d'abstraction et raffinement [DAH, WIR] au découpage d'un système en unités, modules et machines abstraites.

Les modules représentent l'aspect statique de la décomposition : il s'agit d'un regroupement de fonctions avec des données communes [PAR1, CHE]. On réalise donc dans les modules d'une machine tout ce qui concerne le traitement des données d'un programme s'exécutant sur la machine.

Or, l'exécution d'un programme comporte un deuxième volet :

le *contrôle*, en donnant à ce mot le sens "d'organisation et de commande des différents morceaux d'un traitement".

Si l'organisation modulaire rend compte du traitement des données, par contre le contrôle concerne entre autres la succession des signaux émis, reçus ou échangés par les modules. Cet aspect dynamique de l'exécution d'un traitement est figuré dans notre modèle par la communication de messages entre processus s'exécutant dans les modules.

Nous faisons dériver la notion d'unité de celle de contrôle. Une unité est le support du processus effectuant dans un module la fraction du traitement de données qui est comprise entre la réception d'un message et l'émission du message de sortie. Par un abus de langage qui est licite car il n'introduit pas d'ambiguïté, nous disons que le deuxième aspect de notre modèle de décomposition est celui d'un *réseau d'unités communiquant par messages* (alors qu'en vérité seuls les processus communiquent) [WUL].

Cet aspect du modèle semble mieux adapté à l'expression du contrôle que ne le serait un modèle de décomposition basé sur la notion de procédure et d'appel procédural, à moins de faire intercepter les appels inter-modules par un noyau de contrôle, ce qui serait une manière d'introduire "en fraude", c'est-à-dire hors modèle, un schéma de communication entre unités par messages-paramètres.

II.2. Niveaux de contrôle

Nous venons de voir que l'exécution d'un programme comportait du traitement de donnée et du contrôle. Il faut bien voir à ce propos que ce qui apparaît comme un programme à un certain niveau peut n'être plus qu'une instruction au niveau immédiatement supérieur. C'est une conséquence du mode de décomposition en niveaux d'abstraction.

Un exemple concernant le matériel illustre cette propriété : une opération de transfert sur un organe périphérique est lancée, au niveau d'un programme écrit en langage machine IRIS 80, par une instruction SIO ; ceci se traduit au niveau de l'unité d'échange concernée par le déroulement d'un programme canal [CII].

De même qu'il existe des niveaux d'interprétation dans le traitement des données, auxquels correspondent des niveaux de représentation des données (les ressources logiques), il existe des *niveaux de contrôle* dans le déroulement d'un traitement. Chaque composant d'une machine (unité ou machine incluse) contient ainsi son propre contrôle.

De quelle manière, au niveau immédiatement supérieur, peut-on réaliser le contrôle entre les composants de la machine ?

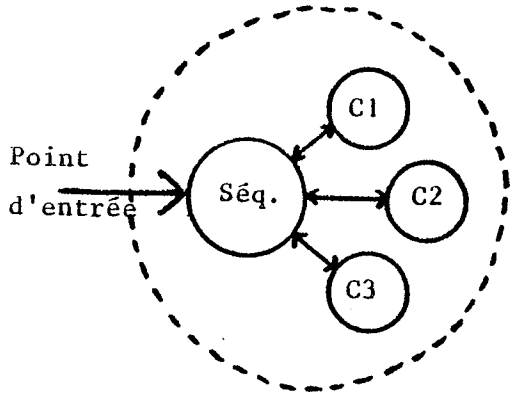
Plusieurs éléments de réponse viennent à l'esprit :

- le contrôle est centralisé dans un des composants de la machine,
- le contrôle est réparti entre les composants de la machine,
- une part plus ou moins grande du contrôle est supportée par l'automate de transition de la machine.

Nous allons illustrer par des exemples ces trois éléments de réponse.

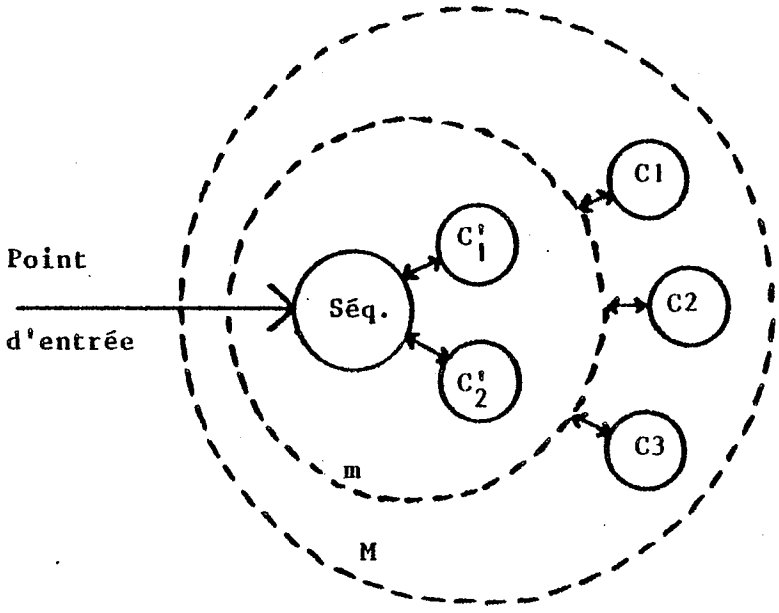
II.2.1. L'étoile

Lorsque le contrôle dans un niveau de machine est centralisé, un des composants effectue en sus de son contrôle propre le contrôle entre les composants [ALT]. Appelons *séquenceur* ce composant. On a alors une architecture en étoile dont le coeur est le séquenceur de la machine.



C_i : composants de la machine

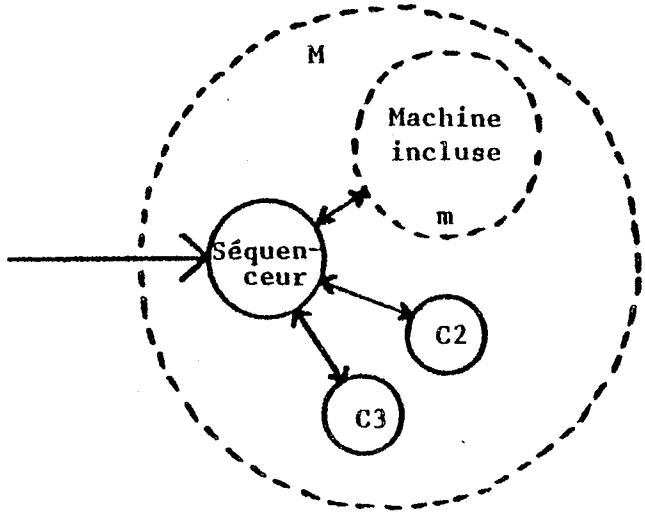
Le séquenceur peut être une unité ou une machine incluse m. Dans ce cas, le séquenceur de la machine englobante M est le même que celui de la machine incluse m. Les composants de la machine englobante ne sont appelés que depuis la machine incluse et non depuis l'extérieur.



Nous reconnaissons dans ce dernier exemple le cas de la machine IRIS incluse dans la machine MAS. Les unités de MAS sont appelées à travers la machine IRIS.

Les unités de la machine englobante servent alors à *enrichir* le langage de la machine incluse, en interprétant de nouvelles instructions rajoutées au jeu d'instruction de la machine incluse.

Dans le cas où l'un des composants C_i autre que le séquenceur est une machine incluse m, celle-ci n'est pas accédée directement depuis l'extérieur de la machine englobante M.

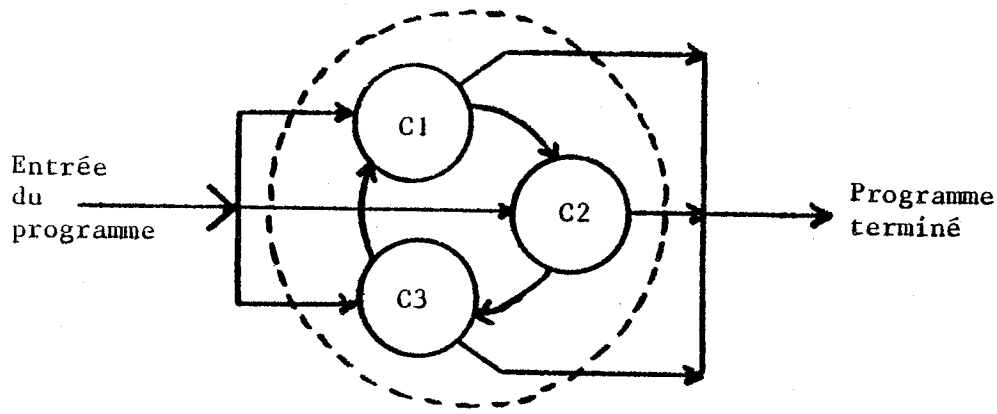


Le séquenceur filtre alors les accès à la machine incluse m. On peut ainsi *appauvrir* le langage de la machine incluse.

Nous venons de voir brièvement deux procédés de modification, par enrichissement ou appauvrissement, du langage défini par une machine m en l'incluant dans une machine M ; ceci dans le cadre d'un contrôle centralisé.

II.2.2. L'anneau

Lorsque le contrôle est réparti entre les composants d'une machine, le programme interprété circule à travers toutes les unités de la machine. On a alors une architecture en anneau.



Les entrées et sorties de programmes peuvent être réparties entre les unités (c'est le cas représenté ici) ou restreintes à une unité, qui réalise alors un peu plus de contrôle que les autres (phases d'initialisat ou de terminaison).

Les exemples de machines réelles organisées selon ce schéma abordent : réseaux de machines en anneau, microprocesseurs branchés sur bus uniques [MAZ]. Ce cas d'un anneau avec entrée et sortie uniques est illustré par l'ensemble composé d'une unité d'échange IRIS 80 connectée aux unités de liaison des organes périphériques [CII].

Les informations de contrôle véhiculées par le programme interprété comprennent entre autres un pointeur d'instruction courante. A l'entrée du programme, ce pointeur repère la première instruction. Un composant de la machine recevant le programme interprète, à partir du pointeur d'instruction courante, la suite des instructions qu'il peut reconnaître, faisant progresser le pointeur jusqu'à la première rencontre d'une instruction inconnue. Le contrôle est alors passé au composant suivant de la machine qui agit de même, et ainsi de suite. La rencontre de la "fin de programme" provoque le renvoi du programme selon la transition finale de l'automate de la machine.

Selon ce schéma tel qu'on vient de le présenter, on ne sort de l'anneau que lorsque le programme est terminé. Ceci a deux graves inconvénients :

- si une instruction n'est reconnue par aucun composant de la machine, le programme circule indéfiniment de composant en composant ;
- le deuxième inconvénient découle du premier : on ne peut enrichir le langage de la machine que par insertion d'un ou de plusieurs composants dans l'anneau, c'est-à-dire par modification de la machine existante et non par construction d'une machine englobante gérée elle-même en anneau (puisque dès que le contrôle passe à la machine incluse, on est assuré de ne pas en sortir). Même chose évidemment pour l'appauvrissement.

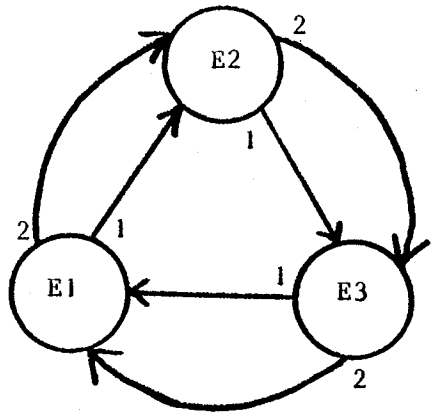
On peut évidemment remédier à cela en introduisant, parmi les données de contrôle, une variable d'état (compteur) qu'on gèrera dans chaque composant de manière à éviter le bouclage éternel. Ceci revient alors à introduire un élément de contrôle centralisé dont la gestion est dupliquée dans chaque composant .... Cette solution bâtarde est inévitable lorsque le matériel est effectivement décentralisé. Mais il vaut mieux revenir dans les autres cas au schéma de contrôle général par séquenceur, tout en admettant parfaitement la possibilité d'anneaux locaux dans une machine, contrôlés globalement par le séquenceur de la machine.

II.2.3. L'automate

On peut enfin gérer une part plus grande du contrôle au niveau de l'automate de transition en multipliant les états.

Illustrons cette assertion en présentant une solution par automate au problème du bouclage infini dans un anneau (cf. § II.2.2 précédent).

Automate initial

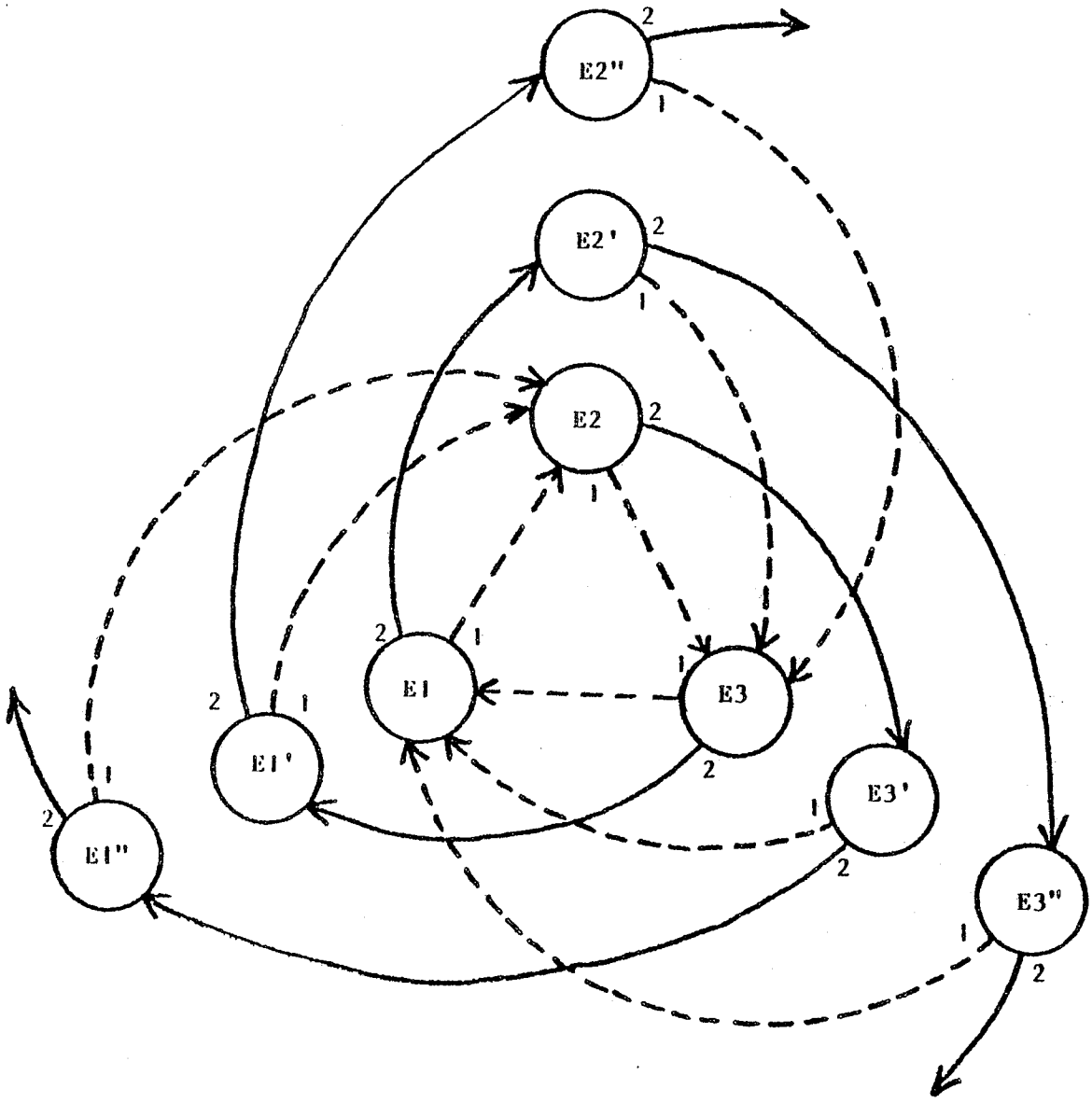


Pour simplifier, nous n'avons pas représenté les sorties correspondant à la fin d'exécution du programme, ni l'état final associé. Les états E1, E2 et E3 sont des états initiaux ; ils correspondent respectivement aux lignes d'entrée des composants C1, C2 et C3.

Les transitions 1 sont les transitions normales entre composants de la machine lorsqu'au moins une instruction du programme a été interprétée dans le composant qui passe le contrôle. Les transitions 2 correspondent au cas où aucune instruction n'a été interprétée dans le composant qui passe le contrôle au suivant.

Pour résoudre le problème du bouclage infini, nous introduisons pour chaque composant deux états supplémentaires.

L'automate de transition obtenu est le suivant :



Nous avons représenté en traits discontinus les transitions correspondar aux sorties n° 1 des composants (cas où des instructions ont été exécuté

Les transitions n° 2, à partir des états E1", E2" et E3", doivent provoq normalement une sortie de la machine du type "instruction inexistante".

Un anneau avec n composants aurait demandé la définition de n^2 états de l'automate !

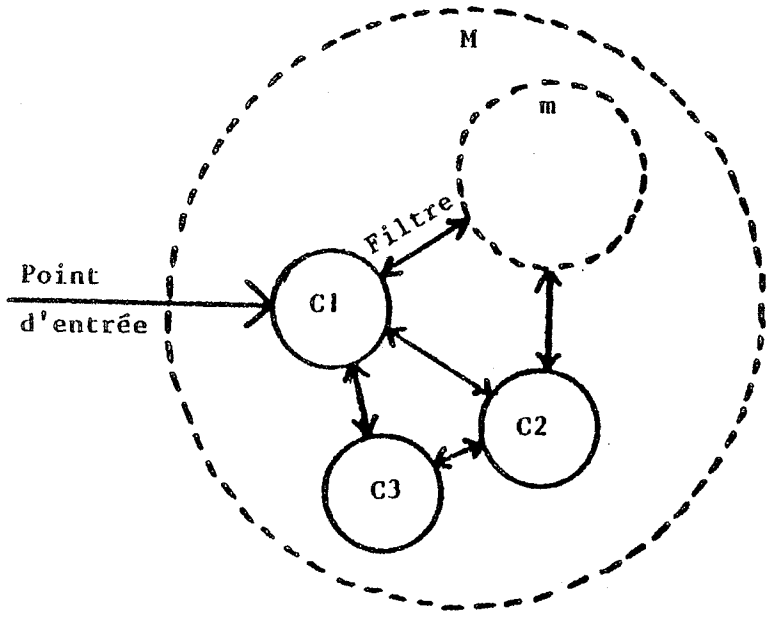
Nous voyons sur cet exemple que la prise en compte d'une partie du contr par l'automate de transition n'est pas chose facile à définir.

II.2.4. Synthèse

Toute architecture de système est réductible à une combinaison des trois procédés que nous avons évoqués. Notre modèle de décomposition est donc théoriquement apte à prendre en compte tout type de contrôle et réaliser tout type d'architecture. Toutefois, l'efficacité et la souplesse militent en faveur d'un contrôle global centralisé dans un séquenceur, lorsque la chose est possible [JAM], les autres formes de contrôle pouvant être admises localement.

Le procédé qui permet de constituer dans le cas général un niveau de langage à partir du langage de la machine de niveau inférieur dont on dispose est dérivé du procédé d'enrichissement et appauvrissement évoqué au § II.2.1. Il suffit de construire une nouvelle machine constituée de nouveaux composants (unités supplémentaires pour l'enrichissement, séquenceur-filtre pour l'appauvrissement) et de la machine initiale.

Exemple :



m : machine initiale définissant le langage ℓ

C1 : filtre pour les appels à m, appauvrissant ℓ

C2 et C3 : composants interprétant de nouvelles instructions enrichissant ℓ

M : machine construite définissant le langage L, dérivé de ℓ par enrichissement/appauvrissement.

II.3. OUTILS DE CONSTRUCTION

II.3.1. Unités, modules et machines abstraites

Deux procédés de découpage d'un système informatique en éléments simples coexistent donc dans notre modèle : le découpage en unités et en modules. Il découle de ce que nous avons vu jusqu'à présent qu'un module contient une ou plusieurs unités mais qu'une unité ne peut pas s'étendre sur plusieurs modules.

Les machines sont construites par connexion d'éléments. Or, la connexion est du domaine du contrôle, et nous avons vu que ceci concerne les unités et non les modules. Il est donc clair que les outils de construction que nous fournissons privilégient les unités au détriment des modules :

une machine est un assemblage d'unités et de machines incluses obtenues le même procédé. Une unité peut éventuellement constituer une machine.

Il est donc théoriquement possible dans notre modèle de construire des machines ne regroupant qu'une partie des unités de plusieurs modules, les autres unités des mêmes modules pouvant faire partie de machines disjointes. A la limite, un module regroupant plusieurs unités peut contenir plusieurs machines.

On s'éloigne ainsi de la notion de machine abstraite découpée de manière fonctionnelle, telle qu'elle a été présentée au début de cette étude (cf. § I.2). Les composants d'une machine gèrent alors des ressources qui risquent d'être accédées en dehors de la machine au sein d'une machir concurrente, ce qui est mauvais car la définition de modules dans un système répond à la nécessité de protéger des ressources et de garantir la cohérence de leur représentation [CHE].

Le risque existe de voir la notion de machine abstraite, présentée en introduction, se diluer dans celle, plus générale mais beaucoup moins structurée et contrôlable, de réseau de processeurs. Les deux notions sont, certes, connexes, mais néanmoins situées en marge l'une de l'autre (cf. § II.3.2).

Il est donc important de respecter un certain *mode d'emploi* dans l'utilisation de nos outils : un système informatique ne peut être valablement considéré comme une machine abstraite formée par un assemblage d'unités et de machines incluses qu'à la condition suivante : dès lors qu'une unit est un composant d'une machine, toutes les autres unités qui appartiennen au même module doivent être aussi des composants de la machine.

II.3.2. Machines multiprocesseurs

Le parallèle s'impose et n'est pas fortuit : étant donné l'expansion des microprocesseurs et son incidence prévisible sur l'architecture des machines, nous avons cherché à développer des outils qui ne soient pas incompatibles avec les procédés d'assemblage de processeurs. De ce fait, la possibilité d'évaluer de telles architectures est une retombée importante de notre projet (cf. § III.2).

Si l'on considère les composants d'une machine comme des processeurs particuliers, on peut distinguer deux types de processeurs [MAZ] :

- les processeurs spécialisés qui correspondent aux unités,
- les processeurs banalisés qui correspondent aux machines incluses.

Certains processeurs partagent une mémoire commune ; ce sont les unités regroupées dans un module.

En fait, si l'on considère qu'une unité est multipliée en autant de processeurs que de processus qu'elle supporte, les composants de base d'une machine multiprocesseurs sont alors les processus serveurs et les machines incluses. Il est clair que les machines incluses peuvent elles-mêmes être réalisées en terme de réseaux de processeurs.

A la hiérarchie de machines - à laquelle correspond déjà la hiérarchie des automates de transition - correspond ainsi, dans notre modèle, une hiérarchie de processeurs organisés par niveaux.

II.3.3. Moyens de communication

Deux modes de communication coexistent dans notre modèle :

- la communication par messages entre processeurs (unités ou machines) gérée par automate,
- la communication par données partagées situées dans une mémoire commune (unités regroupées en modules).

On peut donc représenter de manière structurelle dans notre modèle les modes de communication physique entre processeurs : communication par mémoire commune (grappe-module) ou communication par bus ou ligne entre grappes/machines et modules/machines.

En fait ces deux modes de communication peuvent se combiner pour en former un troisième : communication par messages dans lesquels sont incluses des références à des données partagées, mais éventuellement non directement accessibles.

Donnons brièvement quelques exemples illustratifs.

Communication par données partagées accessibles directement en mémoire commune :
L'unité chargée de lancer les entrées/sorties dans le module de traitement des transferts avec les organes périphériques, et celle qui est chargée du traitement des messages associés aux interruptions d'entrée/sortie dans ce même module, ont toutes deux accès aux tables décrivant l'état de la configuration des organes périphériques, et ceci éventuellement

en parallèle [TAR]. Il est clair que les accès à ces tables se trouvent en section critique au sein du module.

Ceci apporte une retouche supplémentaire au tableau rapidement brossé au § II.2 sur le contrôle : le contrôle dans une architecture de ce type n'est pas seulement localisé à l'intérieur de chaque composant agissant pour son propre compte ou dans l'algorithme, quel qu'il soit, régissant les transitions entre composants ; il est aussi au coeur de cette entité fonctionnelle qu'est un module regroupant plusieurs unités. Le contrôle entre unités n'appartenant pas au même module est associé à la communication par messages ; et parallèlement il réside dans un mécanisme élémentaire de synchronisation en exclusion mutuelle pour les unités - ou plutôt les processus - d'un même module.

Communication par messages contrôlée par automate de transition :

Il s'agit de messages ne contenant que des valeurs et non des références. Du point de vue du contrôle, la sémantique des signaux échangés par ce moyen entre composants est assez pauvre. Son intérêt réside en ce qu'il permet un contrôle, sans synchronisation parasite, par séquenceur centralisé, ayant une vue globale de la charge de la machine et capable, pour peu que les algorithmes ad hoc y soient implantés, de piloter la régulation de charge sur la machine [JAM].

De plus, nous avons déjà vu abondamment que son intérêt du point de vue architectural était fondamental.

Communication par messages faisant référence à des données partagées :

Ce peut être le cas, par exemple, de diverses unités faisant partie d'une machine "Méthodes d'Accès Fichiers", qui ont accès en parallèle aux segments contenant les sous-catalogues associés aux fichiers. Ou encore le cas de deux processus de la même unité de pagination de MAS traitant en parallèle une faute de page sur le même segment et ayant ainsi accès à la même table de pages [VAT1].

On emploie ce moyen de communication dès que le nombre de valeurs à transférer dans un message devient trop grand pour être fait de manière économique. On préfère alors ranger les données dans une zone accessible

en commun et ne transmettre dans les messages que la référence à la zone en question.

La donnée commune est alors considérée comme une ressource critique sur laquelle on peut faire opérer des instructions de synchronisation évoluées (moniteurs, compteurs, ...) réalisées au moyen des outils de base que nous fournissons.

On peut en effet intégrer dans n'importe quel niveau de machine abstraite une unité de synchronisation jouant le rôle d'une section critique pour l'accès à toute donnée sur laquelle des conflits peuvent survenir.
Une telle unité est le support d'un seul processus et fournit des instructions de manipulation d'objets du type abstrait "ressource critique" par exemple CREER-RESSOURCE, et inversement DETRUIRE-RESSOURCE, et des instructions de synchronisation sur la ressource créée. Une telle unité peut servir à enrichir le langage de toute machine abstraite (cf. § II.2.4).

Ce schéma de synchronisation n'est pas incorporé dans notre modèle mais est réalisable au moyen des outils que nous fournissons et facilement intégrable à tout niveau. On peut donc l'ajouter sous forme de "recette" au mode d'emploi de nos outils.

III. PERSPECTIVES

Il y a un certain nombre de prolongements possible du projet OURS [MAI1]. Nous n'évoquerons ici que celui qui concerne l'évaluation de systèmes fonctionnellement distribués, en particulier sur des architectures multiprocesseurs.

Nous avons vu (cf. § II.3.2) que notre modèle de décomposition permettait la représentation d'architectures multiprocesseurs : processeurs en grappes, représentés par des unités regroupées en modules ; réseau de processeurs ou grappes ou machines communiquant par messages. On peut donc représenter des architectures du type CM* [FUL] ou machines réseau [CHUP].

Une restriction concerne le mode de communication : les transmissions de messages se font, dans notre réalisation, par copie de mémoire entre espaces virtuels et non par fil. On ne peut donc pas tester directement le comportement temporel d'un système réparti sur un multiprocesseur.

Il faut intégrer les outils que nous fournissons comme partie d'un syst d'évaluation de systèmes.

Illustrons ce point par un exemple de ce qu'il est possible de faire : soit S un système informatique fonctionnellement défini, dont on veut évaluer une réalisation possible sur multi-microprocesseur.

- On commence par écrire une version simplifiée de S en prenant une uni pour un processeur et en regroupant éventuellement en module les unité à mémoire commune.

- On observe et on mesure (indépendamment du temps) le comportement de l'architecture de S sur des jeux d'essai. On commence par observer le déroulement dans le réseau d'une commande initiale. Puis on fait de mê pour une certaine charge globale du système S. On est alors en mesure d'assigner des coefficients de probabilité aux transitions entre les unités de S. On peut alors fabriquer l'automate probabiliste du réseau de processeurs.

- Cet automate est ensuite utilisé en entrée d'un simulateur de files d'attente [MER], dans lequel les durées correspondant aux divers modes de transmission ainsi que les temps de service à l'intérieur des unité sont également introduits sous forme de paramètres ajustables. On peut alors analyser les performances du système S, compte tenu des loi d'arrivées et des paramètres de l'architecture (temps de service des unités, temps de transmission).

Une telle approche fournit un procédé puissant de définition de systèmes répartis sur des architectures multiprocesseurs. On pourrait, par exemple en faisant varier les paramètres d'architecture, définir les moyens de communication les plus appropriés pour un certain type d'architecture : modularité de certaines unités regroupées sur mémoires communes, transmissions par bus entre certaines unités ou modules, transmissions par lignes synchrones/asynchrones entre d'autres unités ou modules. On pourra ainsi déterminer la meilleure manière de câbler un modèle que l'on aurait évalué par la méthode indiquée.

Il est important de noter que tout ceci ne concerne que l'évaluation de systèmes distribués, non leur réalisation. Une condition première pour l'utilisation de notre modèle dans la réalisation d'un système implanté sur un multiprocesseur serait d'adapter la notion d'automate à une

architecture répartie, c'est-à-dire de faire éclater un automate sur les divers sites. Ceci reviendrait à décentraliser la machine IRIS sur tous les points du site. Mais sous doute faudra-t-il définir, pour réaliser cela, un langage de programmation évolué dans lequel l'expression du contrôle, et en particulier du contrôle réparti, serait aussi puissante, sinon plus, que celle du traitement des données.

IV. CONCLUSION

Notre modèle de décomposition de systèmes informatiques, basé sur la méthode d'abstraction et raffinement, s'appuie sur les notions de processus et de communication par messages, qu'il privilégie.

Il a été réalisé par des mécanismes de construction d'unités et de machines, connectés par l'intermédiaire d'un mécanisme universel : l'automate de transition des machines construites.

Il a été appliqué à la construction de la machine MAS, ce qui en montre la validité.

L'expérience qu'il a permis d'acquérir dans les domaines de l'architecture de systèmes et la mise en oeuvre du contrôle à l'intérieur de systèmes fonctionnellement distribués peut être utile dans la construction de systèmes répartis sur des machines multi-microprocesseurs, et bien entendu également de systèmes centralisés.

Pour le reste, le docteur Queuille disait :

"Il n'est pas de problème que l'absence de solution ne finisse par résoudre".

V. BIBLIOGRAPHIE

[ALT] J. ALTABER
Basic description of the software operating system for the computer control of the SPS.
CERN Lab.II - CO/75-1.

[AUR] A. AUROUX, C. HANS
Introduction au système CP/67 - CMS.
Monographies Informatiques. L. Bolliet, Dunod.

[BEK] Y. BEKKERS, D. HERMAN, M. RAYNAL
Conception et réalisation d'une machine-langage de haut niveau adaptée à l'écriture des systèmes.
Thèse 3ème Cycle, Rennes, Septembre 1975.

[BEN] M.J. BENNETT
A virtual evaluation tool for a system having a virtualizeable processor.
First annual SIGME Symp. on measurement and evaluation, Palo Alto, Calif., Février 1973.

[BÖS] H. BÖSMAN, A. TARABOUT, W. WERUM
Der Kern eines allgemeinen PEARL-Betriebssystems.
Lecture notes in computer science, vol.12, Juin 1974.

[BRI1] J. BRIAT & al.
Projet OURS - Manuel de présentation.
Note interne OURS 1000, Avril 1976.

[BRI2] J. BRIAT, S. GUIBOUD-RIBAUD
Espace d'adressage et espace d'exécution du système GEMAU.
Aspects théoriques et pratiques des systèmes d'exploitation, IRIA, 1974.

[BRIN] P. BRINCH HANSEN
Concurent programming concepts.
Computing surveys, vol.5, 4, 1974.

[CAM] R.H. CAMPBELL, A.N. HABERMANN
The specification of process synchronization by path expression
Aspects théoriques et pratiques des systèmes d'exploitation,
IRIA 1974.

[CHE] J.L. CHEVAL & al.
Conception modulaire de systèmes d'exploitation.
Rapport de recherche n° 32, ENSIMAG, Mars 1976.

[CHUP] J.C. CHUPIN, J. SEGUIN
MADRE - Objectifs de définition d'une méthode d'accès direct
réseau. Centre Scientifique CII, Grenoble, Avril 1974.

[CII] CII - Ordinateur IRIS 80
Manuel d'utilisation, N° 4152 E/Fr.

[CRO] CROCUS
Systèmes d'exploitation des ordinateurs. Dunod.

[DAH] O.J. DAHL, E.W. DIJKSTRA, C.A.R. HOARE
Structured programming. Academic Press. 1972.

[DES] R. DESCARTES
Discours de la méthode. 1637.

[DIJ1] E.W. DIJKSTRA
The structure of "THE" multiprogramming system.
CACM Vol. 11 n° 5, 1968.

[DIJ2] E.W. DIJKSTRA
Hierarchical ordering of sequential processes.
Acta Informatica, Vol.1 fasc.2, 1971.

[DIJ3] E.W. DIJKSTRA
Cooperating sequential processes.
Programming languages, F. Genuys ed., Academic Press, 1967.

[EST] J. ESTUBLIER
Processus cyclique et appel procédural.
Thèse 3ème Cycle, INPG, Avril 1978.

[FUL] FULLER, JONES, DURHAM editors
CM* review. Juin 1977.

[GOL] R.G. GOLDBERG
Architecture of virtual machines.
AFIPS National Computer Conference, 1973.

[HAB1] A.N. HABERMANN
Path expressions.
Carnegie Mellon University 1975.

[HAB2] A.N. HABERMANN, L. FLON, L. COOPRIDER
Modularization and hierarchy in a family of operating systems.
CACM vol.19 n°5, 1976.

[HOA1] C.A.R. HOARE
Operating systems : their purpose, objectives, functions and scope. Operating systems techniques, Hoare & Perrott editors, Academic Press, 1972.

[HOA2] C.A.R. HOARE
Monitors : an operating system structuring concept.
CACM Vol.17 n°10, 1974.

[HOA3] C.A.R. HOARE
Communicating sequential processes.
CACM Vol.19 n°8, Août 1976.

[JAK] M.A. JACKSON
Principles of program design.
Academic Press, 1975.

[JAM] A.J. JAMMEL, H.G. STIEGLER
Structural decomposition and distributed systems.
Note interne du Centre de Calcul Leibniz, Munich, Novembre 1977.

[JAQ] P. JACQUET
Les types génériques - Propositions pour un mécanisme d'abstraction dans les langages de programmation.
Thèse 3ème Cycle INPG, Septembre 1978.

[LIN] T.A. LINDEN
Operating system structures to support security and reliable software.
Computing Survey ACM, Vol.8 n°4, 1976.

[LIS] B. LISKOV, S. ZILLES
Programming with abstract data type.
SIGPLAN notices, Vol.9, 4, 1974.

[MAI1] B. MAILLOT, A. TARABOUT, I. VATTON
Un outil de recherche en systèmes informatiques.
Thèses Docteur-Ingénieur et 3ème Cycle INPG, Décembre 1978.

[MAI2] B. MAILLOT
Un outil de recherche en systèmes informatiques : outils de répartition des ressources.
Thèse de Docteur-Ingénieur INPG, Décembre 1978.

[MAU] P. MAURICE, P.C. SCHOLL
Un interpréteur A.P.L. pour le CII 90-80.
Bulletin de l'IRIA, Spécial APL, Mars 1971.

[MAZ] G. MAZARE
Systèmes multi-microprocesseurs : problèmes de parallélisme, définition et évaluation d'un système particulier.
Thèse d'Etat, Grenoble, Juin 1978.

[MER] D. MERLE, D. POTIER, M. VERAN
Un outil d'analyse des performances des systèmes informatiques.
Congrès AFCET 1978.

[MOS] J. MOSSIERE
Méthodes pour l'écriture des systèmes d'exploitation.
Thèse d'Etat INPG, Septembre 1977.

[ORG] E.I. ORGANICK
The MULTICS system : an examination of its structure.
MIT Press, Cambridge Mass., 1972.

[PAR1] D.L. PARNAS
On the criteria to be used in decomposing systems into modules.
CACM Vol.15 n°12, 1972.

[PAR2] D.L. PARNAS
On a "buzzword" hierarchical structure.
IFIP 74, North Holland Publishing Company.

[ROB] P. ROBERT, J.P. VERJUS
Towards autonomous descriptions of synchronization modules.
Proc. IFIP Congress, Toronto, 1977.

[TAR] A. TARABOUT
ESLOGIC et ESPHYSIQ - Réalisation des entrées/sorties.
Note interne OURS 2003, Juin 1976.

[VAT1] I. VATTON
Segmentation - Pagination.
Note interne OURS 2000, Novembre 1976.

[VAT2] I. VATTON
Un outil de recherche en systèmes informatiques : modèle de décomposition.
Thèse 3ème Cycle INPG, Décembre 1978.

[WIR] N. WIRTH
Program development by stepwise refinement.
CACM Vol.14 n°4, 1971.

[WUL] W.A. WULF editor
The HYDRA operating system.
Carnegie Mellon University.