IMACS Congress 1979, Simulation of systems.
Sorrente, Italy, September 24 - 28, 1979.
CEA - CONF 4772

SIMULATION EN TEMPS REEL DE GRANDS SYSTEMES SUR MINI ORDINATEUR

Michel NAKHLE
Pierre ROUX

Centre d'Etudes Nucléaires de SACLAY
B.P. n° 2 - 91190 GIF/YVETTE (FRANCE)

NEPTUNIX 2 est un logiciel permettant de simuler un modèle décrit par un système d'équations algébro-différentielles écrites sous la forme implicite :

$$F(x, \dot{x}, t) = 0$$

Les équations sont écrites, dans le langage d'entrée, sous la forme mathématique ; le compilateur se charge d'effectuer la dérivation formelle pour obtenir la matrice jacobienne du système.

L'algorithme d'intégration est implicite, à pas et ordre variable. Le pas n'est pas limité, comme dans les méthodes explicites, par la plus petite constante de temps du système d'équations.

Des variations topologiques du système, au cours d'une simulation, sont autorisées, par le truchement de variables logiques. L'utilisateur a en outre la possibilité d'empêcher qu'un pas d'intégration chevauche la valeur de la variable indépendante pour laquelle s'est produit un changement d'état d'une variable.

Le temps d'exécution de l'algorithme d'intégration est optimisé par un traitement non numérique préalable, dont les idées directrices sont les suivantes :

- introduction d'une hiérarchie dans les éléments de la matrice jacobienne du système d'équations, de façon à éviter les calculs inutiles (exemple : calculer une seule fois les éléments constants),

- exploitation du caractère 'creux' de la matrice pour n'effectuer que les opérations strictement nécessaires lors de l'inversion de cette matrice,

- ordonnancement de la matrice jacobienne, de façon à minimiser le nombre d'opérations nécessaires pour son inversion.

La phase non numérique de NEPTUNIX 2 (compilation, dérivation formelle, ordonnancement) nécessite un gros ordinateur pour s'exécuter. Elle délivre tous les programmes sources (en FORTRAN) nécessaires à la simulation du modèle soumis. L'intégration peut alors s'éxécuter indifféremment sur le même ordinateur ou sur un ordinateur de taille plus modeste, un langage d'exécution permettant de modifier les paramètres du modèle, de décrire les sorties demandées et d'enchaîner les simulations. Le recours au gros ordinateur n'est alors nécessaire qu'en cas de changement structurel des équations du modèle.

La notion d'intervalle de communication et la présence de dispositifs de synchronisation permet, sur mini-ordinateur, le couplage du système simulé numériquement avec un processus externe fonctionnant en temps réel.

Une version simplifiée de NEPTUNIX 2 est actuellement utilisée pour modéliser une centrale nucléaire représentée par plusieurs centaines d'équations, dont certaines ne sont pas explicitables. Par ailleurs, un réacteur PWR, représenté par une soixantaine d'équations, est simulé sur mini-ordinateur.

SOMMAIRE

La plupart des langages de simulation n'acceptent qu'une formulation explicite des équations différentielles et les variables logiques n'y ont pas un statut particulier. Les méthodes d'intégration proposées ont leur pas limité par la plus petite constante de temps du modèle soumis.

Le logiciel de simulation NEPTUNIX 2 possède un langage qui admet les équations implicites et une méthode d'intégration dont le pas, variable, n'est pas limité par les constantes de temps du modèle. Ce fait, allié à une optimisation poussée en temps et en ressource mémoire du code engendré, font de NEPTUNIX 2 un outil de base pour la simulation sur mini-ordinateur. Les variables logiques étant des entités spécifiques gérées de façon centralisée, cela permet un traitement correct des discontinuités et la synchronisation avec un processus réel.

I - INTRODUCTION

Les algorithmes d'intégration utilisés par les logiciels de simulation supposent que les équations différentielles décrivant le modèle sont écrites sous la forme explicite :

$$\dot{x} = f\ (x,t)$$

De plus, le pas d'intégration est limité par la plus petite constante de temps du modèle simulé. Certains logiciels (CSMP [CI], ACSL [AI]) ont tourné cette difficulté en offrant une méthode d'intégration "STIFF", basée sur l'algorithme de Gear [GI]. Cette méthode suppose le calcul de la matrice jacobienne du système d'équations par différentiation numérique, puis l'inversion de cette matrice. Outre l'imprecision du calcul de la matrice jacobienne, cette méthode à l'inconvénient d'être coûteuse en temps d'exécution.

CSMP et ACSL offrent un certain nombre de fonctions fort utiles en simulation (limiteur, espace mort, fonction step, comparateur).

Ces fonctions amènent des discontinuités sur les variables ou leur dérivée sans qu'il soit possible de recaler le pas d'intégration sur la discontinuité. Cela est dû au fait que les variables logiques n'ont pas de statut particulier dans ces langages.

Enfin, toutes les phases (compilation, édition de liens, intégration) d'un logiciel de simulation s'exécutent sur un même ordinateur. Le couplage en temps réel à un processus extérieur ne peut se faire que si le logiciel est implanté sur un mini-ordinateur.

NEPTUNIX 2 admet la description d'un modèle sous la forme d'équations algébro-différentielles implicites :

$$f(x, \dot{x}, t) = 0$$

La méthode d'intégration (M1), à pas et à ordre variable, est une généralisation de la méthode de Gear, mais la matrice jacobienne est calculée de manière exacte par dérivation formelle des fonctions f au moment de la compilation. Le code de calcul de la matrice jacobienne et le code d'inversion de cette matrice sont spécifiques au modèle soumis et subissent une optimisation poussée, ce qui rend l'algorithme d'intégration très efficace. Aussi NEPTUNIX 2 peut accepter des modèles décrits par plusieurs centaines d'équations.

Une originalité de NEPTUNIX 2 est la gestion centralisée des variables logiques. Cela permet de considérer que les discontinuités divisent l'intervalle d'intégration en une suite d'intervalles élémentaires, sur lesquels toutes les variables sont continues.

Enfin, NEPTUNIX 2 est conçu dans une optique d'informatique répartie, concrétisée par le développement des réseaux informatiques. La phase non numérique (compilation, optimisation, etc.) est exécutée sur un gros ordinateur, la phase numérique (intégration) est exécutée sur gros ordinateur pour la mise au point du modèle et sur mini-ordinateur pour l'exploitation du modèle (figure 1).

La section 2 porte sur la description d'un modèle dans le langage d'entrée de NEPTUNIX 2. La section 3 décrit brièvement la phase non numérique La section 4 décrit la phase numérique. La section 5 aborde la simulation sur mini-ordinateur. La section 6 porte sur quelques applications.

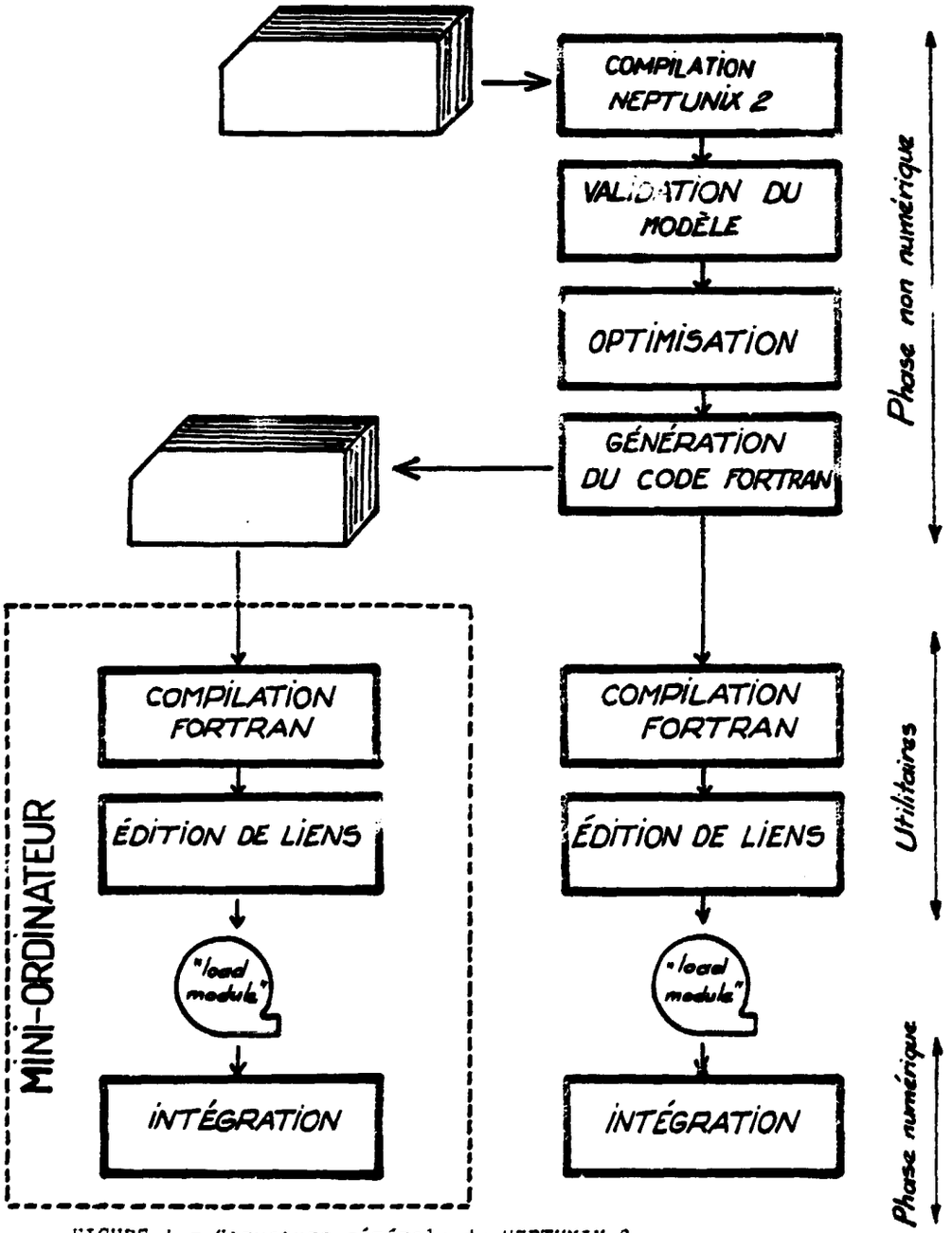


FIGURE 1 - Structure générale de NEPTUNIX 2

2 - LANGAGE D'ENTREE DE NEPTUNIX 2

2.1 - Composants d'un modèle (figure 2)

Un modèle est décrit par un ensemble d'équations algébro-différentielles. Certaines équations peuvent changer suivant le domaine de fonctionnement du modèle. Le modèle utilisé est défini par l'état d'un ensemble de paramètres, prenant des valeurs discrètes durant une simulation, qui sont les sorties d'un automate. Il y a changement d'état de l'automate lorsque le modèle, au cours de son évolution, change de domaine de fonctionnement. Ce phénomène est concrétisé par le basculement d'un comparateur (certains comparateurs agissent directement sur le modèle, sans passer par l'automate). L'ensemble des comparateurs constitue l'interface.

Ce schéma est une généralisation d'un modèle de convertisseur à thyristors (I1) et rappelle la structure d'un calculateur analogique-logique. Les variables annexes recouvrent les compteurs, horloges, temporisateurs, etc. en général disponibles dans la partie logique du calculateur.

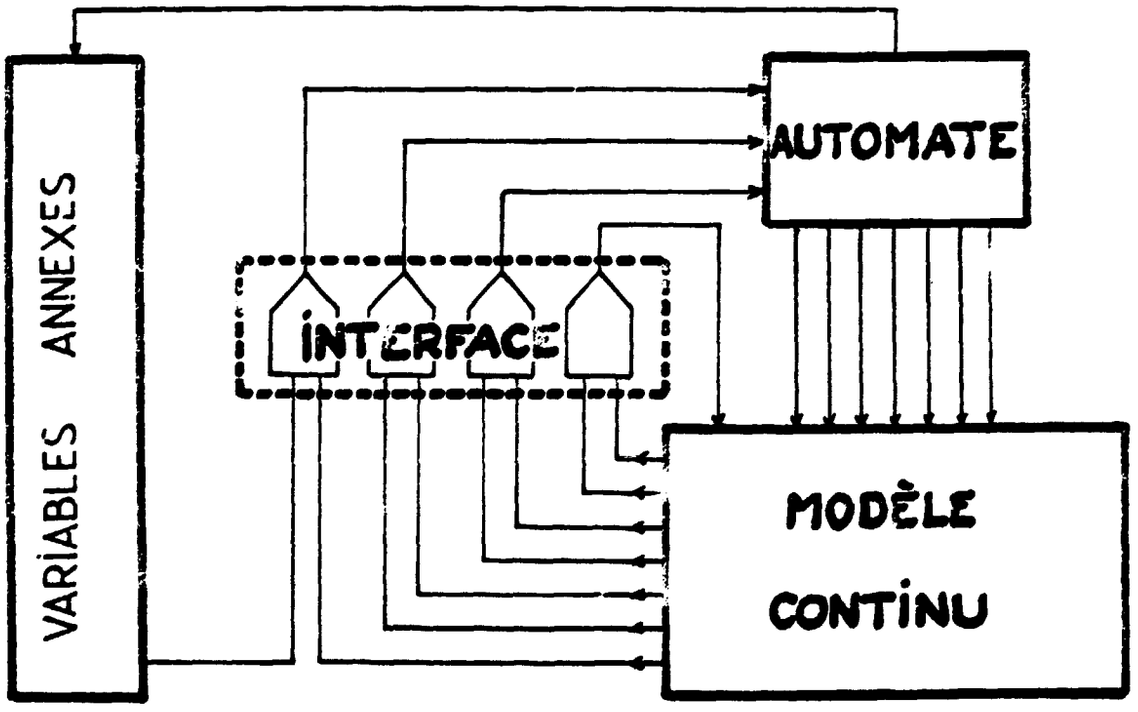


FIGURE 2 - Composants d'un modèle

2.2 - Blocs

Le langage d'entrée se présente sous la forme d'une suite de blocs. Chaque bloc contient des données spécifiques. Un mot-clé identifie le bloc, le mot-clé "FIN" termine un bloc. Les mots-clés sont précédés d'un point d'exclamation (!). Un point (final) suit le dernier mot-clé "FIN". Un bloc peut être désactivé et réactivé ou, autrement dit, deux blocs peuvent être identifiés par le même mot-clé :

EXEMPLE :

```
! DECLARATIONS
    données
! FIN
! EQUATIONS
    données
! FIN
! DECLARATIONS
    données
! FIN
! INTERFACE
    données
! FIN
! AUTOMATE
    données
! FIN
! EQUATIONS
    données
! FIN
! FORTRAN
    données
! FIN.
```

Cinq blocs permettent de décrire complètement un modèle :

- DECLARATIONS : définition des variables paramètres et fonctions intervenant dans le modèle, avec leurs attributs

- EQUATIONS : équations algébro-différentielles du modèle

- AUTOMATE — description de l'automate engendrant les discontinuités lors de l'intégration

- INTERFACE — description des entrées de l'automate issues des variables continues

- FORTRAN — sous-programmes FORTRAN complétant éventuellement le modèle

Tout symbole distinct d'un mot-clé, apparaissant dans les blocs INTERFACE, AUTOMATE et EQUATIONS doit figurer dans un bloc DECLARATION qui précéde. Le bloc FORTRAN doit apparaître après tous les autres.

Les autres blocs, non décrits ici, permettent de donner des directives à chacune des étapes de NEPTUNIX 2 (compilation, génération de code, intégration, etc.).

2.3 - Bloc DECLARATIONS

Exceptées les constantes, tous les objets nécessaires à la description d'un modèle doivent être déclarés :

- paramètres
- variables continues
- variables logiques
- paramètres logiques
- variables annexes
- fonctions

Tous les objets ont leur valeur représentée par un nombre réel [BI].

Tous les objets peuvent être indicés ; dans ce cas, le symbole est la concaténation d'une racine et de l'indice. Ainsi la déclaration Z (20) est équivalente à la déclaration Z1, Z2, Z3, ..., Z20. La variable indépendante doit être déclarée (attribut INDEP).

Les variables logiques ne prennent que deux valeurs distinctes. Elles sont de deux types :

- variable logique pure
- sortie de comparateur

Les variables logiques pures ne peuvent apparaître que dans les blocs AUTOMATE et EQUATIONS.

Un comparateur a l'attribut DERIV ou NODERIV. Dans le second cas, le basculement du comparateur ne provoque pas de changement d'état de l'automate (cela correspond aux comparateurs définis dans CSMP ou ACSL). Implicitement, toutes les variables logiques pures ont l'attribut DERIV.

Un comparateur ayant l'attribut DERIV peut avoir l'attribut TBASC qui exige que, en cas de basculement du comparateur, la fin du pas d'intégration coïncide avec la valeur de la variable indépendante au moment du basculement.

Les variables logiques ayant l'attribut DERIV peuvent avoir l'attribut DEMAR qui indique qu'en cas de basculement, l'algorithme d'intégration procède comme s'il démarrait une nouvelle simulation.

Les paramètres logiques prennent des valeurs discrètes et sont mis à jour par l'automate.

Les variables annexes ne peuvent apparaître que dans les blocs AUTOMATE et INTERFACE.

Les fonctions sont de deux types :

- INTERNE : fonctions mathématiques usuelles (SIN, LOG, etc.)
- EXTERNE : écrites par l'utilisateur. Dans ce cas, un certain nombre d'attributs peuvent leur être attachés (fonction définie par tableau, fonction mémoire, fonction paramétrée, etc.).

EXEMPLE :

```
! DECLARATIONS
  VARIABLES CONTINUES : X, Y ;
  COMPARATEURS : ZPOS (2) DERIV, CHUTE DERIV ;
  VARIABLES LOGIQUES : L (3) ;
  VARIABLES ANNEXES : TZ ;
  VARIABLES CONTINUES : Z (20), T INDEP ;
  PARAMETRES : A, B, C, TD, S ;
  PARAMETRES LOGIQUES : NIV, POS ;
  COMPARATEURS : OPEN DERIV TBASC DEMAR ;
! FIN
```

2.4 - Bloc INTERFACE

Ce bloc définit les entrées des comparateurs.

EXEMPLE :

```
! INTERFACE
  ZPOS1 = Z7>Ø ; ZPOS2 = T>TD ;
  CHUTE = 2 * (Y - X)>Z1 ;
  OPEN  = T>TZ ;
! FIN
```

2.5 - Bloc AUTOMATE

Ce bloc définit le nouvel état de l'automate provoqué par le basculement d'une variable logique, par une suite d'instructions d'affectation. Il peut donc y avoir une suite d'instructions pour le basculement de 0 à 1 et une autre suite pour le basculement de 1 à 0.

EXEMPLE :

```
! AUTOMATE
  L1'>0  : X = (X+0,5)* L2+(X+1.)*'1-L2),L2=1-L2 ;
  Li'<0  : X = X-0,5, L2 = 1-L2 ;
  L2'>0  : Z5,Z6,Z7 = FONC (L1,L3,ZPOS1,ZPOS2) ;
! FIN
```

Les instructions d'affectation peuvent être exécutées dans un ordre quelconque. Ce qui est à droite du signe "=" concerne l'état actuel de l'automate, ce qui est à gauche concerne son nouvel état. Un comparateur ne peut apparaître à gauche du signe "=".

Comme l'exemple le montre, les instructions de l'automate permettent de changer de modèle et d'introduire des discontinuités sur les variables continues.

2.6 - Bloc FORTRAN

Ce bloc est composé des sous-programmes ou fonctions FORTRAN complètant le modèle. Le compilateur ne fait aucune vérification syntaxique, mais, sur demande, insère les déclarations de zone "COMMON" permettant de communiquer avec NEPTUNIX 2.

2.7 - Bloc EQUATIONS

EXEMPLE :

```
Z15-ZPOS2*S*(T-TD) ;
X-Z1*Z2-Y*Z5 ;
Z3'+X*(1-Z6')+3.7"-3 ;
Z7-B*(((Z8-Z9)/C)  2)*EXP(Z11/A) ;
FONC1(Z12,$,Z13,$) ;
```

Seul est codé le premier membre des équations implicites f (x,ẋ,t) = 0. EXP est une fonction interne.

FONC1 est une fonction externe ; au moment de l'exécution, les dérivées partielles de la fonction par rapport à Z12 et Z13 doivent être fournies par l'argument placé à la position marquée d'un "$". S, A, B, C et TD sont des paramètres. ZPOS2 est une variable logique ; l'équation où elle figure s'écrit :

$$Z15 = 0 \qquad \text{si} \quad ZPOS2 = 0$$
$$Z15 - S*(T-TD) = 0 \qquad \text{si} \quad ZPOS2 = 1$$

Ce qui est une façon de décrire une fonction rampe. Tous les autres identificateurs désignent les variables continues. Le caractère ' désigne la dérivée de la variable qui précéde par rapport à la variable indépendante T. Les caractères " et ↑ remplacent respectivement E (ou D) et ** dans les langages FORTRAN ou PL/1.

Aucune variable annexe ne peut apparaître dans une équation. Sur cinq équations, quatre sont non linéaires et implicites, une seule est une équation différentielle.

3 - PHASE NON NUMERIQUE

3.1 - La compilation [N1]

Le compilateur analyse le programme source, effectue la dérivation formelle des équations pour obtenir le jacobien du système, et classe les éléments du jacobien suivant leur type (constant, fonction des paramètres ou variables logiques, fonction de la variable indépendante, non linéaire) en vue de l'optimisation du code engendré. Lors de la dérivation formelle, les paramètres logiques et les variables logiques sont considérés comme des constantes.

3.2 - L'optimisation

Outre le classement des éléments du jacobien suivant leur type, NEPTUNIX 2 choisit les pivots minimisant le nombre d'opérations lors de l'inversion de la matrice jacobienne. La matrice servant au choix des pivots est celle de l'état initial du système.

3.3 - Validation du modèle

Pour le moment, cela se limite à vérifier que la matrice jacobienne initiale n'est pas singulière (LI).

3.4 - Génération de code

Le code de calcul du jacobien et le code correspondant aux blocs INTERFACE, AUTOMATE, EQUATIONS sont engendrés lors de l'étape de compilation sous la forme de quadruplets interprêtables. Le code d'inversion de la matrice jacobienne est engendré sous la même forme lors de l'étape de validation du modèle. L'étape de génération de code consiste à :

- engendrer le code FORTRAN correspondant aux quadruplets,
- engendrer un sous-programme BLOCK DATA pour initialiser les zones COMMON spécifiques au modèle simulé,
- engendrer la table des symboles,
- engendrer des sous-programmes qui transmettent aux modules fixes de la phase numérique les ressources spécifiques au modèle soumis.

Durant cette étape, NEPTUNIX 2 utilise une description de la machine-cible. Pour s'adapter aux caractéristiques de la machine-cible, il est possible de :

- réduire la table des symboles,
- supprimer l'automate
- scinder les programmes engendrés.

4 - PHASE NUMERIQUE

La phase numérique est exécutée après compilation FORTRAN et édition de liens sur la machine hôte. L'exécution est pilotée par un programme source en langage d'exécution.

4.1 - Structure générale

L'interprêteur, l'algorithme d'intégration et le post-traitement peuvent s'exécuter dans la même zone mémoire si l'éditeur de liens de la machine hôte le permet, ce qui minimise l'encombrement.

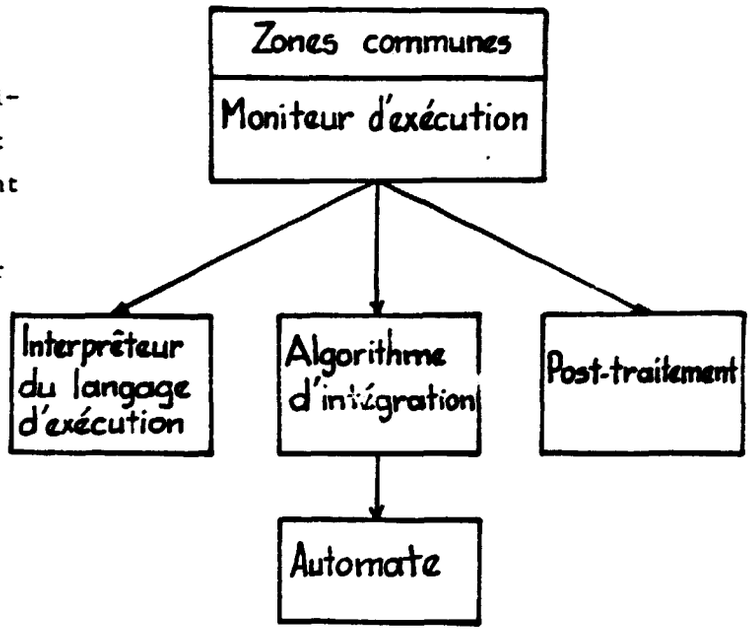


FIGURE 3 - Structure générale de la phase numérique

4.2 - Langage d'exécution

Le langage d'exécution permet de lire ou modifier les paramètres, les variables, les attributs, de définir les sorties et d'enchaîner des simulations.

EXEMPLE :

```
DEBUT ;
        commandes définissant les variables à sauvegarder
      SIMUL  δ = (val1, val2, val3) ;
        commandes
      EXEC ;
        commandes
      SIMUL ;
        commandes
      EXEC ;
        commandes de post-traitement
FIN ;
FINPTX ;
```

Cet exemple décrit un enchaînement de simulations, les trois premières étant enchaînés automatiquement et correspondant à trois valeurs distinctes du paramètre B. Les variables sauvegardées peuvent être éditées après les quatre simulations sous forme de table ou de courbe, ou être conservées en vue de traitements ultérieurs.

4.3 - Algorithme d'intégration (figure 4)

Le modèle continu est décrit par les équations :

$$f(x,\dot{x},t) = 0$$

Un pas normal d'intégration se passe de la façon suivante :

1) interpolation par un polynôme d'ordre k passant par les k + 1èmes valeurs passées de la variable x. La valeur du polynôme et celle de sa dérivée à la fin du pas courant deviennent une prédiction de x et $\dot{x}$.

2) la prédiction est le point de départ d'un processus itératif de NEWTON-RAPHSON qui se décompose comme suit :

   . détermination de l'état des comparateurs NODERIV
   . calcul des fonctions F et de la matrice jacobienne

$$J = \frac{\partial f}{\partial x} - \gamma \frac{\partial f}{\partial \dot{x}}$$

   . résolution du système $J \delta x = - f(x,\dot{x},t)$ par inversion de la matrice jacobienne
   . correction des valeurs de x et $\dot{x}$

$$x = x + \delta x$$
$$\dot{x} = \dot{x} - \gamma \delta x$$

3) si l'algorithme de NEWTON-RAPHSON converge et conduit à des valeurs compatibles avec l'erreur tolérée, le pas est accepté, sinon le pas est diminué et une nouvelle prédiction est fait en 1).

4) traitement des discontinuités (HI). L'automate change d'état si l'un des comparateurs DERIV change d'état (basculement asynchrone). La fin du pas coïncide avec la valeur de la variable indépendante au moment du basculement du premier comparateur ayant l'attribut TBASC, sinon le pas est inchangé. Ce premier changement d'état de l'automate peut provoquer des changements successifs d'état de l'automate (basculements synchrones) jusqu'à un état final stable.

5) si aucun changement de modèle n'est apparu durant le pas, l'algorithme recherche l'ordre k qui donne le pas le plus grand compatible avec l'erreur tolérée. Le pas trouvé devient le pas suivant avec l'ordre k correspondant.

5 - SIMULATION SUR MINI-ORDINATEUR

5.1 - Interaction avec l'utilisateur lors de la mise au point du modèle

La mise au point du modèle s'effectue sur gros ordinateur. Elle bénéficie donc des systèmes conversationnels développés par le constructeur (TSO sur la série 370 d'IBM).

5.2 - Optimisation en temps d'exécution de la phase numérique

L'optimisation porte sur le code exécuté le plus souvent, c'est-à-dire le calcul du jacobien et l'inversion de la matrice. C'est là que le classement des éléments du jacobien suivant leur type est précieux. En effet, le code de calcul est divisé en quatre (voir figure 4) :

- les éléments constants du jacobien et de son inverse sont calculés une seule fois en Ⓒ,
- les éléments du jacobien et de son inverse dépendant des variables et/ou paramètres logiques sont calculés en Ⓛ, à chaque changement de modèle,
- les éléments du jacobien et de son inverse dépendant de la variables indépendante sont calculés en Ⓣ à chaque nouveau pas,

- les éléments non linéaires et l'inversion complète de la matrice sont effectués à chaque itération de NENTON-RAPHSON.

Par ailleurs, NEPTUNIX 2 peut délivrer, pour ces calculs, au lieu du FORTRAN, un code interprétable qui peut servir de base à une exécution micro-programmée.

5.3 - Optimisation en place mémoire de la phase numérique

Au prix d'une limitation des possibilités et d'une moins grande facilité d'emploi, NEPTUNIX 2 peut engendrer un code adapté aux ressources mémoires du mini-ordinateur. Cela est possible grâce à la conception modulaire de la phase d'exécution. Ainsi il est possible de :

- réduire, voir supprimer la table des symboles,
- supprimer l'automate,
- faire les calculs en simple précision au lieu de la double précision,
- de manière générale, fixer la taille des tables de travail de la phase numérique

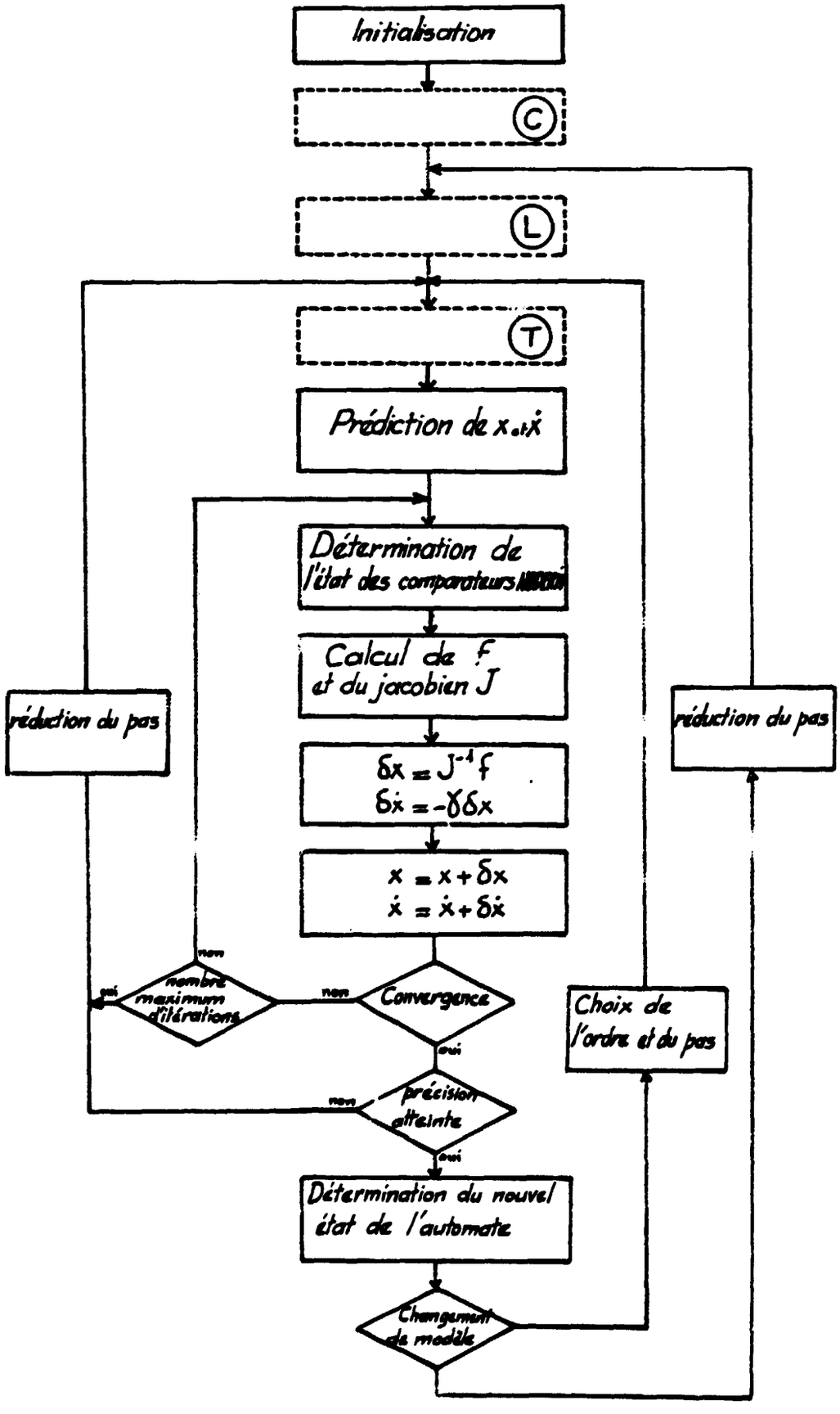
5.4 - Portabilité du code FORTRAN

Les modules fixes de la phase numérique sont écrits en FORTRAN IV le plus élémentaire possible. Une description de la machine-cible permet à la base non numérique de NEPTUNIX 2 d'engendrer un code FORTRAN adapté aux caractéristiques du mini-ordinateur et aux limites d'implantation de son compilateur FORTRAN.

La manipulation de caractères est évidemment le point le plus critique. Le parti a été pris d'écrire des programmes FORTRAN transportables pilotés par un descripteur de la machine-hôte. Ces programmes conduisent, bien sûr, à des performances déplorables de l'interpréteur, mais peuvent être remplacés par des programmes dans le langage assembleur du mini-ordinateur.

5.5 - Simulation en temps réel

L'optimisation en temps et en place mémoire permettent d'avoir, sur mini-ordinateur, un code exécutable de simulation d'un modèle décrit par une centaine d'équations, dont les performances permettent d'envisager une exécution en temps réel.


Initialisation
C
L
T
Prédiction de x et ẋ
Détermination de l'état des comparateurs
Calcul de f et du jacobien J
δx = J⁻¹ f
δẋ = -γδx
x = x + δx
ẋ = ẋ + δẋ
nombre maximum d'itérations
non
oui
Convergence
non
oui
précision atteinte
non
oui
Détermination du nouvel état de l'automate
Changement de modèle
réduction du pas
réduction du pas
Choix de l'ordre et du pas

Le caractère particulier des liaisons d'un mini-ordinateur avec le temps réel nous a fait renoncer pour le moment à résoudre ce problème lors de la génération du code FORTRAN. La solution choisie est de donner le contrôle à un programme écrit par l'utilisateur en plusieurs points :

- à l'initialisation de la simulation
- à chaque pas d'intégration
- lors du basculement d'une variable logique
- à chaque intervalle de communication
- à la terminaison de la simulation

Lorsque le contrôle lui est donné, l'utilisateur a accès à la description du modèle et à l'état de la simulation. Cela lui permet de faire une synchronisation avec le temps réel, d'effectuer des conversions analogique-digital ou digital-analogique, d'envoyer des commandes logiques ou d'enregistrer un événement extérieur.

6 - APPLICATIONS

Une version prototype (NEPTUNIX 1), dans laquelle la phase numérique est écrite en PL/1 et en assembleur, est opérationnelle depuis le 1er septembre 77 sur IBM 370/168 et IBM 360/91 (N2) (B2).

Cette version est utilisée pour la mise au point de modèles de centrales nucléaires. Le plus gros modèle de centrale PWR simulé actuellement résulte du couplage de deux générateurs de vapeur et d'un préssuriseur ; cela représente 190 équations algébro-différentielles. Il est prévu d'y adjoindre un modèle axial de coeur de 150 équations. Les équations du générateur de vapeur, en particulier, sont très non linéaires et implicites, il est donc exclu de les expliciter pour utiliser des programmes de simulation tels que CSMP. Les modèles obtenus ont été qualifiés par comparaison avec les résultats de codes spécialisés.

Un modèle de réacteur HTR (coeur point + générateur de vapeur) de 103 équations a été simulé à l'aide de NEPTUNIX 1 et de CSMP III sur IBM 360/91. En pénalisant NEPTUNIX 1, c'est-à-dire en lui imposant un pas maximum égal au pas fixe de CSMP, le temps d'intégration de NEPTUNIX 1 est le même que celui de la méthode des trapèzes de CSMP.

Une version FORTRAN de la phase d'exécution de NEPTUNIX I a été expérimentée sur SEMS MITRA-15. Le temps d'intégration du même modèle sur MITRA-15 est environ trente fois plus élevé que sur IBM 360/91. L'élément critique sur mini-ordinateur est le temps d'exécution des opérations en virgule flottante.

CONCLUSION

Le langage d'entrée de NEPTUNIX 2 est "portable" puisqu'il exprime les équations d'un modèle sous la forme mathématique. La présence de variables et paramètres logiques permet d'indiquer les différentes formes d'une équation et leur domaine de validité.

La simulation d'un modèle soumis à NEPTUNIX 2 est portable, en particulier, sur mini-ordinateur, avec des performances comptatibles avec le temps réel. Cela permet, dans le cas d'un mini-ordinateur connecté à une installation reelle dont le modèle est simulé sous NEPTUNIX 2, de prévoir les conséquences à moyen terme d'une commande qui serait envoyée manuellement (guide opérateur).

NEPTUNIX I, qui contient bon nombre des caractéristiques de NEPTUNIX 2, a fait la preuve de son efficacité et de sa fiabilité auprès d'utilisateur d'un Centre de Recherche. NEPTUNIX 2 est la version industrielle de NEPTUNIX I.

REMERCIEMENTS

Les auteurs tiennent à exprimer leurs remerciements à leur collègue P. METTON qui a mené a bien la traduction de la phase numérique de PL/I et AL360 en FORTRAN et l'implantation sur SEMS MITRA 15.

REFERENCES

(A1) "ACSL - Advanced Continuous Simulation Language"
User/Guide Reference Manual
Mitchell and Gauthier Assoc. Concord, Mas. (1975)

(B1) BONNEMAY
Description globale des ensembles hybrides
Rapport CEA-R-4751
CEN-SACLAY BP2 91910 GIF SUR YETTE (1976)

(B2) BONNEMAY, DANSAC BON, MONSEF, NAKHLE, ROUX
NEPTUNIX a package for continuous system modeling - Application to nuclear reactor simulation - Large Engineering Systems 2
IEEE Symposium Waterloo Canada (mai 1978)

(C1) "CSMP III - Continuous System Modeling Programm III"
Program Reference Manual
IBM-SH 19-7002 (1975)

(G1) GEAR
Numerical Initial Values Problems in Ordinary Différential Equations
Prentice-Hall (1971)

(H1) HAY
Objet Programm Structures for simumation Language Translators
AICA-Symposium on Simulation Language For Dynamic Systems
London (september 1975)

(I1) IUNG, LE DOEUFF, LOUIS, THOMESSE
Une modélisation des convertisseurs à thyristors adaptée à la simulation numérique
C.R. Académie des Sciences de PARIS (12 juin 1978)

(L1) LAPORTE, VIGNES
Méthode numérique de détection de la singularité d'une matrice
Num. Math. 23 pp 73-81

(MI) MONSEF

Contribution à la résolution de grands systèmes algébro-différentiels.

Thèse de Docteur ex-Sciences Physique - Université de Paris Sud (1977)

(NI) NAKHLE

Etude d'un langage et réalisation d'un système de simulation muni d'une dérivation formelle. Application aux réacteurs nucléaires, au langage de commande et aux réseaux électroniques.

Thèse de Docteur es-Sciences Physique - Université PARIS-SUD (1979)