DEPARTEMENT D'ELECTRONIQUE ET D'INSTRUMENTATION NUCLEAIRE
Service d'Instrumentation pour la Recherche

Saclay, le 15 juin 1982

N° nomenclature : 7520

Clt matière : A3

2. International symposium on applied modeling and simulations.
Paris, France 29 june-2 july 1982
CEA-CONF-- 6392

NEPTUNIX

UN LANGAGE DE SIMULATION DE SYSTEMES CONTINUS

Michel NAKHLE*- Pierre ROUX

DEIN/INT/SIR/82.23

NEPTUNIX : UN LANGAGE DE SIMULATION DE SYSTEMES CONTINUS

Michel NAKHLE
Compagnie Internationale de Services en Informatique
RTI/DES/ES-RP
BP 24 - 91191 Gif-sur-Yvette CEDEX France

Pierre ROUX
Commissariat à l'Energie Atomique
Centre d'Etudes Nucléaires de Saclay
DEIN/SIR
91191 GIF-sur-Yvette CEDEX France

Sommaire

Le progiciel NEPTUNIX accepte la description du modèle mathématique d'un système physique, à partir de laquelle il construit un simulateur portable. L'automatisation de bon nombre d'opérations assure, vis-à-vis de l'algorithme d'intégration, une grande indépendance du langage de description du modèle et du langage d'exploitation du simulateur.

INTRODUCTION

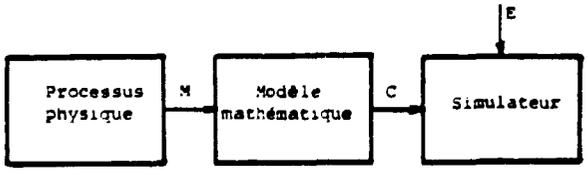
Le coût d'une expérimentation ou même son impossibilité technique impose de plus en plus le recours à la simulation numérique ou analogique pour prévoir le comportement d'un processus physique complexe (centrale nucléaire, avion,...). La simulation permet également de créer un environnement artificiel évoluant en temps réel, servant à l'entraînement du personnel (conducteur de centrale nucléaire, pilote d'avion,...)

Cela contraint à la mise au point de modèles mathématiques les plus fidèles possibles à la réalité. Parmi ceux-ci, une classe importante est constituée par les modèles dont l'expression mathématique est un ensemble d'équations algébriques ou différentielles ordinaires, dont la forme peut changer face à des évènements.

$$f(x,x',l,t)=0$$

t est la variable indépendante, x, qui peut subir des discontinuités, est une variable continue par morceau, x' est la dérivée de x par rapport à t, l est une variable logique prenant des valeurs discrètes et définissant les "morceaux", i.e. le domaine de validité de chacune des expressions du modèle.

Le problème posé est donc de construire le simulateur correspondant à un processus physique, via une modélisation mathématique de ce dernier.



Il apparait trois activités distinctes : l'activité de modélisation (M), aboutissant à un modèle mathématique, l'activité de compilation (C), se concluant par un simulateur en état de marche, l'activité d'exploitation (E) du simulateur. Un progiciel de simulation doit donc se donner pour objectif d'assister l'activité "M", d'automatiser l'opération "C" et de faciliter toutes les exploitations possibles d'un simulateur.

Un certain nombre de critiques peuvent être adressées à la plupart des progiciels de simulation développés jusqu'ici (CSMP, ACSL,...) [C79]

- Dépendance du langage de description des modèles vis-à-vis des méthodes d'intégration du simulateur

En particulier, le caractère nécessairement procédural de l'algorithme d'intégration est conservé dans le langage de description. Cela se traduit par le remplacement de l'égalité mathématique par l'affectation informatique, notamment pour expliciter les dérivées dans les équations différentielles :

$$x' = f(x,l,t)$$

ce qui peut amener, au prix de manipulations longues et fastidieuses et même de simplifications frustantes, à transformer radicalement une formulation mathématique naturelle. Le choix d'une extension du langage FORTRAN comme langage de description de modèles (CSMP,ACSL) ne fait que confirmer cette orientation.

- Inefficacité avec les systèmes "raides", c'est-à-dire ceux qui possèdent des constantes de temps très séparées. Avec les méthodes classiques d'intégration (RUNGE-KUTTA, ADAMS,...), la plus petite constante de temps limite le pas d'intégration, ce qui, outre l'inefficacité, peut entraîner des cumuls d'erreurs inacceptables.

- Aide très limitée à l'activité de modélisation : Un défaut du modèle se traduit souvent par un non-fonctionnement du simulateur. Les faibles indices que fournit le simulateur dans ce cas ne permettent pas de dégager complètement la responsabilité de la méthode d'intégration et, a fortiori, de trouver le défaut du modèle.

- Difficultés avec les discontinuités : Si un évènement se produit durant un pas d'intégration, l'algorithme d'intégration est, en général, incapable de le détecter et, a fortiori, de déterminer l'instant où il se produit, mais, ce qui est plus grave, il peut être perturbé par cet évènement d'une manière qui peut se révéler fatale.

- Dépendance vis-à-vis de l'ordinateur-hôte du simulateur : Bien que la distinction entre le langage de description d'un modèle et le langage d'exploitation du simulateur soit en général assez nette, de fait le "compilateur" et le simulateur partagent le même ordinateur. Cette dépendance est renforcée par le caractère interactif d'un certain nombre de progiciels de simulation.

1 - OBJECTIFS DE NEPTUNIX

La conception d'un nouveau système de simulation ne peut se justifier que s'il ne tombe pas dans les mêmes ornières que les précédents. Aussi NEPTUNIX doit répondre aux objectifs suivants :

1.1 - NEPTUNIX doit offrir une aide substantielle à l'activité de modélisation

NEPTUNIX doit admettre la classe la plus large possible de modèles simulables, en particulier des systèmes non linéaires, "raides", conduisant éventuellement à des discontinuités.

Le langage de description des modèles (langage d'entrée de NEPTUNIX) doit conserver le caractère non procédural du langage mathématique. L'acceptation d'équations implicites évite des manipulations hasardeuses de formules. Un texte en langage d'entrée peut ainsi servir de base de dialogue entre deux experts en modélisation.

Le système NEPTUNIX ne doit pas construire le simulateur d'un modèle mathématique qu'il sait instable, en motivant précisément son refus. Inversement, pour un modèle jugé correct, l'algorithme de résolution construit doit être stable.

Le langage d'entrée doit permettre la construction aisée d'un modèle à partir des sous-modèles déjà au point [E 78] .

1.2 - NEPTUNIX doit faciliter l'exploitation du simulateur qu'il construit

Un langage de commande (langage d'exécution de NEPTUNIX) doit permettre l'exploitation classique du simulateur : modification des paramètres et des valeurs initiales des variables, définition d'une simulation ou de plusieurs simulations enchaînées, exploitation des résultats.

Le simulateur, amputé de ses moyens d'entrée et de sortie, doit pouvoir être utilisé comme un simple sous-programme d'intégration (par exemple dans un programme d'optimisation de paramètres).

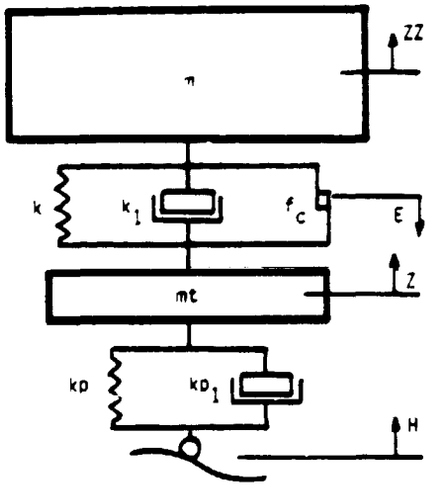
Le simulateur doit être portable, c'est-à-dire exécutable sur n'importe quel ordinateur possédant un minimum de ressources. Le simulateur doit pouvoir fonctionner en temps réel, en échangeant des données avec un processus physique.

L'ambition des objectifs, dont certains appartiennent encore au domaine de la recherche (la stabilité d'un modèle par exemple), pourrait compromettre leur mise en oeuvre si la stratégie suivante n'était adoptée. D'une part, la syntaxe et la sémantique des deux langages de NEPTUNIX sont définies en fonction des besoins de l'activité de modélisation, en les rendant les plus indépendantes possible du simulateur et en particulier de l'algorithme d'intégration et de l'ordinateur d'exécution. D'autre part, toute limitation au langage d'entrée peut être palliée par l'écriture d'une partie du modèle dans le langage procédural choisi pour le simulateur : le FORTRAN. De cette manière, toute avancée de la recherche se traduit par l'enrichissement des algorithmes de NEPTUNIX, mais n'affecte pas les langages. Ainsi, l'emploi du langage FORTRAN devient de moins en moins nécessaire, tandis que les diagnostics de NEPTUNIX 2 deviennent plus précis.

2 - LE LANGAGE DE DESCRIPTION DES MODELES MATHEMATIQUES (LANGAGE D'ENTREE)

La description complète d'un modèle simple en langage d'entrée va permettre de mettre l'accent sur les particularités du langage [NR 81] .

2.1 - Processus physique à modéliser



On se propose de faire rouler sur une piste sinusoïdale un avion dont le fuselage et le train d'atterrissage sont considérés comme des masses rigides. Le pneu en contact avec la piste et l'assemblage fuselage-train sont assimilés à la combinaison de liaisons élastiques et visqueuses. Pour l'assemblage fuselage-train, il s'y ajoute un frottement sec dont la présence introduit des discontinuités dans les forces. Toutes les forces sont verticales.

2.2 - Equations du modèle mathématique

```
!EQUATIONS
                                        COMMENT  FUSELAGE;

  0 = DZZ-ZZ';
  0 = M*DZZ'-F+FS;
                                        COMMENT  LIAISON FUSELAGE-TRAIN;
  0 = TRBLOQ*E'+(1-TRBLOQ)*(F-K*E-SGNDE*(K1*E'!Z+FC));
  0 = E-ES+ZZ-Z;
                                        COMMENT  TRAIN;
  0 = DZ-Z';
  0 = MT*DZ'-DFP+F-FS;
                                        COMMENT  LIAISON TRAIN-PISTE(PNEU);
  0 = DFP+KP*(Z-H)+KP1*(Z'-H');
                                        COMMENT  PISTE;
  0 = H-H0*(1-COS(FR*2*3.14159*T));
!FIN
```

Les équations sont écrites sous forme implicite dans un ou plusieurs blocs délimités chacun par !EQUATIONS et !FIN. Seules apparaissent les dérivées premières par rapport à la variable indépendante T. Un certain nombre de fonctions mathématiques sont disponibles (COS,ABS,SIGN,....). La variable logique TRBLOQ mémorise l'état du frottement sec : glissement (TRBLOQ=0) ou blocage (TRBLOQ=1). Suivant l'état, il existe deux équations possibles. Ce qui est remarquable, c'est que les deux expressions possibles sont rassemblées dans une seule et même équation, la troisième, dans laquelle la variable logique TRBLOQ a été "algébrisée" [B 76] . L'opérateur booléen "complément" est donc exprimé sous forme arithmétique : 1-TRBLOQ.

2.3 - Déclarations des variables et des paramètres

```
!DECLARATIONS

        VARIABLES CONTINUES:

  ZZ,    COMMENT DEPL. DU FUSELAGE / A SA POSITION STATIQUE      M    ;
  Z,     COMMENT DEPLACEMENT DU TRAIN / A SA POSITION STATIQUE   M    ;
  E,     COMMENT ENFONCEMENT DU TRAIN                            M    ;
  H,     COMMENT HAUTEUR DE LA PISTE                             M    ;
  DZZ,   COMMENT DERIVEE DE ZZ                                   M/S  ;
  DZ,    COMMENT DERIVEE DE Z                                    M/S  ;
  F,     COMMENT FORCE DE LIAISON ENTRE FUSELAGE ET TRAIN        SN   ;
  DFP;   COMMENT FORCE PNEU                                      SN   ;

        PARAMETRES:

  FC,    COMMENT FORCE DE COULOMB                                    ;
  M,MT,  COMMENT MASSES DU FUSELAGE ET DU TRAIN                      ;
  K,KP,  COMMENT COEFFICIENTS DE RAIDEUR                             ;
  K1,KP1,COMMENT COEFFICIENTS DE FROTTEMENT VISQUEUX                 ;
  FS,ES, COMMENT VALEURS STATIQUES DE F ET E                         ;
  FR,H0; COMMENT FREQUENCE ET AMPLITUDE                              ;

        VARIABLES LOGIQUES:

  TRBLOQ      ,     COMMENT 1:ETAT BLOQUE,0:ETAT DEBLOQUE;
  SGNDE SIGNE;      COMMENT 1:SIGNDE>0.,-1:SIGNDE<0;

        COMPARATEURS DERIV:

  SIGNDE SIGNE TBASC SEUIL=1.E-10,
  COMP   TBASC;

!FIN
```

Les déclarations figurent dans un ou plusieurs blocs délimités chacun par !DECLARATIONS et !FIN. Toute déclaration d'un identificateur doit précéder son utilisation. Le bloc ci-dessus doit donc être le premier dans le programme de description du modèle pris pour exemple. Le type des identificateurs est lié aux besoins de l'activité de modélisation et non aux nécessités informatiques. Outre les types VARIABLES CONTINUES, PARAMETRES, VARIABLES LOGIQUES et COMPARATEURS DERIV, il existe également les types VARIABLE INDEPENDANTE (T par défaut), PARAMETRE LOGIQUE, COMPARATEUR NODERIV et VARIABLE ANNEXE. A certaines variables peuvent être attachés des attributs. Ainsi, pour un comparateur, SEUIL quantifie la sensibilité et TBASC exige la connaissance exacte de l'instant de basculement. Pour un comparateur ou une variable logique, SIGNE impose que les deux valeurs soient -1 et 1 (au lieu de 0 et 1). Un paramètre logique peut prendre plus de deux valeurs discrètes. La déclaration X(3) est équivalente à la déclaration des identificateurs X01,X02,X03 (variables indicées).

2.4 - Evènements et actions associées

```
!INTERFACE

  COMP    =  ABS(FS*(M/(M+MT))-QFP-K*E) > Z*FC  ;
  SIGNDE  =                              E' > 0    ;

!FIN

!AUTOMATE

  SIGNDE'>0.
  SIGNDE'<0 : SGNDE  = SIGNDE     ;
              TRBLOQ = 1-COMP     ;
  COMP'  >0 : TRBLOQ = 0          ;
  TRBLOQ'<0 : SGNDE  = SIGN(F-K*E);
!FIN
```

L'évènement fondamental est le changement d'état d'un comparateur. Les comparateurs sont définis dans un ou plusieurs blocs délimités chacun par !INTERFACE et !FIN. Un comparateur a la valeur 1 si l'assertion qui le définit est vraie. Chacun des deux changements d'état possibles d'une variable C, déclarée COMPARATEUR DERIV ou VARIABLE LOGIQUE, est considéré comme un évènement. Les deux évènements sont notés C' > 0 et C' < 0. A chaque évènement, peut être associé un ensemble de mises à jour de variables figurant dans un bloc délimité par !AUTOMATE et !FIN. L'ensemble évènements-actions associées constitue les règles de fonctionnement d'un automate. Ainsi, le basculement d'un comparateur, qui est l'évènement fondamental, peut déclencher en cascade plusieurs évènements secondaires jusqu'à la stabilisation de l'automate sur un nouvel état.

Dans l'exemple proposé, les blocs INTERFACE et AUTOMATE décrivent le frottement sec. En situation de blocage, le déblocage ne peut se produire que si la force est supérieure à 2xFC (comparateur COMP). Si cette assertion est vraie, le glissement se produit avec une vitesse E' dans le sens de la force. Si cette vitesse passe par zéro (comparateur SIGNDE), le blocage ne peut à nouveau se produire que si l'assertion ci-dessus ne révèle pas un déblocage immédiat.

2.5 - Valeurs des paramètres et valeurs initiales des variables

```
!INITIALISATIONS

  M=45.87; MT=1.0193; K=2000; KI=10; KP=1000; KP1=70; ES=0.225; FS=450;
  E=0.225; F=450; TRBLOQ=1; SGNDE=1; SIGNDE=1; COMP=0;
  FR=3.; H0=0.05; FC=45;

!FIN
```

2.6 - Autres blocs du langage d'entrée

Un bloc spécial reçoit l'éventuelle partie procédurale du modèle en langage FORTRAN. Ce bloc est inutile dans le modèle pris pour exemple. D'autres blocs permettent d'agir sur le simulateur construit en indiquant son mode d'utilisation et l'ordinateur sur lequel il s'exécute.

3 - LE LANGAGE DE COMMANDE DU SIMULATEUR (LANGAGE D'EXECUTION)

Ce langage, entièrement distinct du précédent, est disponible quel que soit l'ordinateur-hôte du simulateur [R 81] .

3.1 - Exemple

Le simulateur du modèle pris comme exemple peut être commandé par le programme suivant :

```
    DEBUT;
      SAUVER (PERIODE=0,DISC,RAPPEL=(H0)):F,ZZ,E,Z,M,TRBLOQ;
        SIMUL H0=(0.02,0.05,0.1),HXTMIN=0,HXTMAX=2;
          JOURNAL M,Z,ZZ,E,TRBLOQ;
        EXEC;
      COURBE (FORMAT=2, PAGE=(5,10), SIMUL=(1,2,3),
              RAPPEL=(H0), FUSION,
              TITRE='TRAIN D''ATTERRISSAGE SIMPLIFIE',
              TITRE='INFLUENCE DE L''AMPLITUDE')        ZZ;
    FIN;
  FINPTX;
```

Chaque instruction (commande) commence par un mot-clé et se termine par un point virgule.

3.2 - Commandes de structuration

DEBUT et FIN délimitent un ensemble de simulations sur lesquelles portent les commandes de sortie de résultats telles que COURBE. Ces commandes de post-traitement ne peuvent donc intervenir qu'après un ou plusieurs couples de commandes SIMUL-EXEC définissant les simulations. Dans l'exemple ci-dessus, le couple SIMUL-EXEC définit trois simulations de 0 à 2 secondes , avec, pour chacune, une valeur distincte de l'amplitude H0 de la piste.

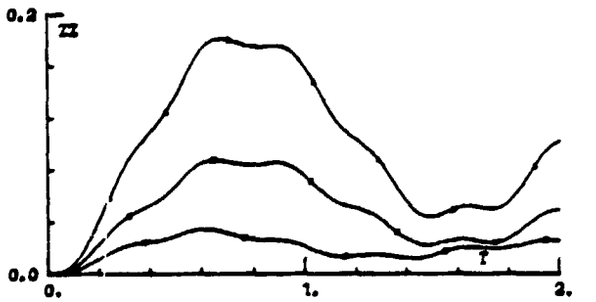
3.3 - Commandes de sortie de résultats

Les sorties de résultats peuvent s'effectuer durant la simulation elle-même (commande JOURNAL) ou en post-traitement (commande COURBE). Le post-traitement ne peut porter que sur les variables dont l'échantillonnage a été demandé par la commande SAUVER. Pour chaque commande, les choix standards du simulateur peuvent être modifiés par des sous-commandes. Par exemple, les sous-commandes de la commande COURBE ci-dessus permettant d'exiger que les évolutions de ZZ pour chacune des trois simulations figurent sur le même graphique (SIMUL,FUSION), de compléter la légende par deux titres et le rappel des valeurs de H0 (TITRE, RAPPEL) et de demander la sortie sur traceur, au lieu de l'imprimante, (FORMAT) avec des dimensions en centimètres qui permettent l'insertion qui suit (PAGE) :

NUMERO DE CAS 1

TRAIN D'ATTERRISSAGE SIMPLIFIE
INFLUENCE DE L'AMPLITUDE

NO SIMUL.	1 □	2 ⋈	3 △
H0 =	2.0000E-02	5.0000E-02	1.0000E-01



3.4 - Mise à jour d'un paramètre, d'une valeur initiale, d'un attribut

```
PARAM FR=4,KP=80;
INIT ZZ'=1E-5,Z=0.01,TRBLOQ=1;
ATTRIBUT COMP NOTBASC;
```

3.5 - Outils de mise au point

Une commande, DEBUG, permet de suivre finement le déroulement de l'intégration et, en particulier, de suivre le fonctionnement de l'automate.

4 - REALISATION

4.1 - Le simulateur

Le simulateur est l'assemblage d'une partie fixe entièrement portable et d'une partie engendrée automatiquement par NEPTUNIX. Celle-ci contient une image du modèle qui ne peut s'exécuter que sur l'ordinateur pour lequel le simulateur est prévu. Chacune des deux parties du simulateur est écrite dans un langage qui n'est qu'un sous-ensemble du FORTRAN 66, ce qui assure le maximum de portabilité.

L'algorithme d'intégration est une généralisation de l'algorithme de Gear [Ge71] [M 77] , qui a l'avantage d'être efficace face à des systèmes "raides". Un traitement centralisé des discontinuités est venu enrichir l'algorithme [NR 82] .

Le programme de commande du simulateur est analysé par un interpréteur piloté par une grammaire LL(1) du langage d'exécution [AU] [Ga 77] .

Le simulateur peut être amputé de l'interpréteur et des modules de sortie de résultats, ce qui permet de l'utiliser comme un simple sous-programme d'intégration.

4.2 - Le générateur de simulateur

Le compilateur NEPTUNIX traite les programmes en langage d'entrée par une analyse montante basée sur une formalisation du langage sous forme de grammaires LR(1)[AU] [Ga 77] hiérarchisées (grammaire de structure de blocs, grammaires spécifiques à chaque bloc, grammaire d'expressions arithmétiques).

La traduction du langage d'entrée, non procédural, en langage procédural FORTRAN impose au compilateur NEPTUNIX des activités très importantes, liées à l'algorithme d'intégration, qui viennent s'ajouter à la simple analyse du langage d'entrée :

- construction de la matrice jacobienne J du système $f(x,x',t)=0$ par dérivation formelle [N 79]
- qualification du modèle et élimination de ceux dont la simulation ne peut être qu'instable [K 81]
- élaboration du code de résolution de $J.\delta x=f$ qui soit le plus stable possible et le plus rapide possible.

Tout le code FORTRAN engendré qui est lié étroitement à l'algorithme d'intégration (calcul de f, calcul de J, calcul de $\delta x=J^{-1}f$, calcul de l'état des comparateurs, calcul de l'état de l'automate) est d'abord engendré sous forme de quadruplets. Outre son exécution possible sous le contrôle d'un interpréteur, cette forme du code permet également d'engendrer simplement le micro-code d'un éventuel micro-processeur spécialisé qui se chargerait ainsi de l'exécution de la partie la plus critique de l'algorithme d'intégration.

Le compilateur NEPTUNIX complète la simple recopie du programme en langage d'entrée par une documentation élaborée (numérotation des équations, références croisées, ...).

5 - APPLICATIONS

NEPTUNIX a été utilisé de façon extensive comme outil de modélisation de réacteurs nucléaires [B79-1] [B 80] aboutissant à la mise au point de modèles de plusieurs centaines d'équations. La caractéristique de NEPTUNIX la plus appréciée est l'aide fournie à l'activité de modélisation et, en particulier, l'introduction, sans modification, de formulations non linéaires ou implicites.

C'est cette même caractéristique qui a permis d'employer directement les équations de Lagrange pour décrire le modèle d'un véhicule automobile, en vue de simuler son comportement.

Par contre, c'est la portabilité qui a motivé le choix d'un simulateur NEPTUNIX pour traiter en temps réel des mesures prélevées sur un site nucléaire [B79-2] . L'ordinateur O installé sur le site n'est pas encore connu, néanmoins la mise au point du simulateur peut s'effectuer sur un ordinateur O_1. Le moment venu, NEPTUNIX sera capable d'engendrer automatiquement le même simulateur pour l'ordinateur O.

CONCLUSIONS

Le compilateur NEPTUNIX est écrit en langage PL/1, tandis que la partie fixe du simulateur est écrite en langage FORTRAN. Ces deux composants du système de simulation NEPTUNIX représentent environ 60.000 cartes en langage évolué et s'exécutent sur le matériel IBM de la série 370. La portabilité du simulateur lui-même a été établie sur plusieurs ordinateurs différents, y compris des mini-ordinateurs.

Pour répondre complètement à l'ensemble des objectifs et, en particulier, pour supprimer toute adhérence avec l'algorithme d'intégration du simulateur, un certain nombre de recherches doivent être poursuivies dans le domaine de la qualification de modèles et de la propagation d'erreur dans l'algorithme d'intégration.

Néanmoins, le caractère industriel du progiciel NEPTUNIX a justifié sa mise au catalogue d'une société de services en informatique.

La "portabilité" du langage de description des modèles et de la documentation fournie par le compilateur, la portabilité des simulateurs construits et le caractère "ouvert" du progiciel font de NEPTUNIX un outil :

- de transmission de connaissances dans le domaine de la modélisation. C'est en particulier un bon auxiliaire d'enseignement
- de construction de simulateurs d'entraînement
- d'exploitation décentralisée d'un simulateur
- de contrôle et de prévision par connexion en temps réel à un processus physique.

En outre, son langage d'entrée, non procédural, en fait un programme d'application facilement intégrable dans un atelier de génie logiciel dans le domaine de la modélisation des systèmes continus. Complété par des outils adéquats de gestion de bases de données et de connaissances, dans un environnement d'intelligence artificiel, il devient une pièce maîtresse dans la contruction de systèmes experts [L82] [NP82] .

REFERENCES

[AU][AHO] A.V., ULLMAN J.D. : The theory of parsing, translation and compiling. Volume 1 Parsing Prentice Hall 1972. Volume 2 Compiling Prentice Hall 1973

[B76] BONNEMAY A. : Description globale des Ensembles Hybrides. Rapport CEA-R-4751

[B79-1] BONNEMAY A., DELOURME D., DRUSCH P., TRAN TUC VI, SAMAK C. : Simulation tools in CEA BNES meeting Bournemouth U.K. 1979

[B79-2] BONNEMAY A., FONTANA X., LEDIEU DE VILLE A., NIMAL J.C., BERAUD : Estimation de la puissance locale dans le coeur d'un PWR à partir de mesures de thermométrie gamma IWG/NPPCI specialist meeting (AIEA) Munich 1979

[B80] BONNEMAY A. : Utilisation de systèmes algébro-différentiels implicites dans la modélisation et la simulation de processus de génération d'énergie IASTED Symposium. AMS 1980 - Interlaken Suisse

[C79] CELLIER F., BONGULIELMI A. : The COSY simulation language IMACS Congress 1979 - Sorrento Italy

[E78] ELMQVIST H. : A structured model language for large continuous systems LUTFD2/(TFRT-1015)/1-226/(1978) Ph. D. Thesis Lund Institute of Technology Sweden

[Ge71] GEAR C.W. : Numerical initial value problems in ordinary differential equations. Prentice-Hall series in Automatic computation

[Ga77] GALLAIRE H. : Techniques de compilation. Méthodes d'analyses syntaxiques CEPADUES-EDITIONS - Toulouse France

[K81] Etude de la stabilité des systèmes algébro-différentiels par les méthodes de Lyapounov. Thèse de Docteur-Ingénieur Université de Paris-Sud Orsay France

[L82] LAURIERE J.L. : Représentation et utilisation des connaissances. Les systèmes experts. RAIRO-TSI Vol 1-n°1 janvier-février 1982

[M77] MONSEF Y. : Contribution à la résolution de grands systèmes algébro-différentiels. Thèse de Docteur-ès-Sciences Université de Paris-Sud Orsay France

[N79] NAKHLE M. : Etude d'un langage et réalisation d'un système de simulation numérique muni d'une dérivation formelle. Thèse de Docteur-ès-Sciences Université de Paris-Sud Orsay F.

[NP82] NAKHLE M., PLANCON M.C. : Résultats d'une expérimentation de conception assistée par ordinateur de circuits électroniques dans un environnement SYSCAO. Rapport interne RTI/DES/ES.RP/82.017 CISI 1982

[NR79] NAKHLE M., ROUX P. : Large systems real time simulation on mini-computers. IMACS congress 1979. North Holland Publishing Company Amsterdam pp 299.305

[NR81] NAKHLE M., ROUX P. : NEPTUNIX 2 - Phase non numérique. Manuel de l'utilisateur. Note CEA-N-2246

[NR82] NAKHLE M., ROUX P. : Integration of nonlinear systems of stiff differential equations with discontinuities. IMACS Congress 1982 Montréal Canada

[R81] ROUX P. : NEPTUNIX 2. Phase numérique. Manuel de l'utilisateur Note CEA.N.2212.

SIR/82-23
A3
7520

Michel NAKHLE Pierre ROUX

NEPTUNIX : UN LANGAGE DE SIMULATION DE SYSTEMES CONTINUS

Le progiciel NEPTUNIX, à partir d'une description mathématique d'un système physique, construit le simulateur correspondant. Le système physique peut posséder des constantes de temps très réparties ("systèmes raides") et sa formulation mathématique (équations différentielles ordinaires et équations algébriques) peut être non linéaire, implicite ; elle peut même changer suivant le domaine de fonctionnement. Le langage de description du modèle mathématique, non procédural, est indépendant de l'algorithme d'intégration. Le simulateur construit par NEPTUNIX, portable sur un grand nombre d'ordinateurs, est doté d'un langage qui rend très aisée son exploitation. Ce langage jouit d'une grande indépendance vis-à-vis de l'ordinateur-hôte et de la méthode d'intégration. Un certain nombre d'avancées récentes dans les algorithmes numériques et non numériques ont présidé à la réalisation du progiciel. NEPTUNIX se révèle un outil puissant de modélisation, en particulier dans le domaine des réacteurs nucléaires.

"NEPTUNIX" : A CONTINUOUS SYSTEM SIMULATION LANGUAGE

From the mathematical description of a physical system, NEPTUNIX builds the corresponding simulator. Algebraic and ordinary differential equations describing a physical system may be "stiff", nonlinear, implicit and even dynamically variable. The non procedural language describing the mathematical model is independent from the integration algorithm. The NEPTUNIX built simulator, transportable on many computers, may be controlled by a userfriendly operating language, independent from host computer and integration method. Last years results about numerical and non-numerical algorithms were used for the package implementation. NEPTUNIX appears as a powerful modeling tool, specially in the field of nuclear reactors design.

Communication présentée au Second IASTED International Symposium à Paris,
29 juin/3 juillet 1982.
CEA-CONF-