Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Analyse de la dynamique dans le projet ISDOS

Monfort, Bernadette

Award date:
1979

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2026

nstitut d'Informatique
NAMUR

ANALYSE DE LA DYNAMIQUE

DANS LE PROJET ISDOS

LBS 6503784

6820.27012

Ce mémoire n'aurait pu être réalisé sans la collaboration de nombreuses personnes.

Ma gratitude va tout d'abord à Monsieur François BODART, professeur aux Facultés Universitaires Notre-Dame de la Paix, pour avoir accepté la direction de ce travail.
Je le remercie pour l'aide attentive qu'il m'a prodiguée tout au long de son élaboration.

Ma reconnaissance va également à Monsieur Yves PIGNEUR, assistant de Monsieur BODART pour sa collaboration durant mon travail.

Qu'il me soit permis également de remercier Madame Anne-Marie HENNEBERT et Monsieur Jean-Marie LEHEUREUX pour les nombreux conseils qu'ils m'ont donnés lors de l'application pratique de ce travail.

Je prie le Professeur D. TEICHROEW, de l'Université du Michigan, d'accepter ma plus vive reconnaissance pour m'avoir permis d'effectuer un stage au sein de son équipe. Je tiens particulièrement à remercier les membres du projet Isdos pour l'aide appréciable qu'ils m'ont apportée durant ce stage.

Enfin, que toutes les personnes que je ne peux citer et qui ont participé d'une quelconque façon à ce mémoire reçoivent ici l'expression de mes plus vifs remerciements.

°

° °

TABLE DES MATIERES

INTRODUCTION 1

PREMIERE PARTIE : ANALYSE DES SPECIFICATIONS DANS LE PROJET ISDOS IMPORTANCE DE LA DYNAMIQUE 4

CHAPITRE I.1 - LE PROJET ISDOS, OUTIL D'AIDE A L'ANALYSE D'UN SYSTEME D'INFORMATIONS 6

I.1.1. Historique 7
I.1.2. Structure générale de l'outil 8
I.1.3. PSL, langage relationnel 10

CHAPITRE I.2 - DECOMPOSITION METHODOLOGIQUE D'UN SYSTEME D'INFORMATIONS 14

I.2.1. Système et sous-systèmes 15
I.2.2. Sous-structure des données 15
I.2.3. Sous-structure des traitements 16
I.2.4. Sous-structure des ressources 16
I.2.5. Quantification 17

CHAPITRE I.3 - PRESENTATION GENERALE DE TROIS VERSIONS DU PROJET ISDOS 18

I.3.1. Analyse de la version 4.2 19

I.3.1.1. Objectifs globaux 19
I.3.1.2. Domaines de spécification 19
I.3.1.3. Types de rapports 24
I.3.1.4. Conclusion de l'analyse de la version 4.2 28

I.3.2. Analyse de la version 5.1 et comparaison avec la version 4.2 32

I.3.2.1. Objectifs globaux 32
I.3.2.2. Domaines de spécification 32
I.3.2.3. Types de rapports 36
I.3.2.4. Différences essentielles par rapport à la version 4.2 37

I.3.3. Analyse de la version NAMUR et comparaison avec les versions 4.2 et 5.1 40
I.3.3.1. Objectifs globaux 40
I.3.3.2. Domaines de spécification 40
I.3.3.3. Types de rapports 46
I.3.3.4. Comparaison de la version NAMUR et des versions 4.2 et 5.1 47

CHAPITRE I.4 - ANALYSE DES DIFFERENTES SOUS-STRUCTURES AU POINT DE VUE DYNAMIQUE 48
I.4.1. Version 4.2 49
I.4.1.1. Dynamique de la sous-structure des données 49
I.4.1.2. Dynamique de la sous-structure des traitements 52
I.4.1.3. Sous-structure des ressources 57
I.4.2. Version 5.1 58
I.4.2.1. Dynamique de la sous-structure des données 58
I.4.2.2. Dynamique de la sous-structure des traitements 60
I.4.2.3. Sous-structure des ressources 68
I.4.3. Version NAMUR 70
I.4.3.1. Dynamique de la sous-structure des données 70
I.4.3.2. Dynamique de la sous-structure des traitements 70
I.4.3.3. Sous-structure des ressources 92

CONCLUSION DE LA PREMIERE PARTIE 97

DEUXIEME PARTIE : REALISATION DE RAPPORTS - APPLICATION AU CAS DU RAPPORT DYNAMIC-INTERACTION 99

CHAPITRE II.1 - PRESENTATION DU METASYSTEM ET DU GENERALIZED ANALYZER 101
II.1.1. Le Meta-system 102
II.1.2. Le Generalized Analyzer 104

CHAPITRE II.2 - CARACTERISTIQUES DE LA PROGRAMMATION DES LOGICIELS ISDOS 107

CHAPITRE II.3 - CAS PARTICULIER DE RAPPORT - LE DYNAMIC INTERACTION 121

II.3.1. Objectifs du rapport 122
II.3.2. Format 123
II.3.3. Les options du DI 132
II.3.4. Analyse effectuée par le rapport DI 134
II.3.5. Limitations 138
II.3.6. Présentation du DI - version NAMUR 140
II.3.7. Extension et conclusion 141

CONCLUSION DE LA DEUXIEME PARTIE 142

TROISIEME PARTIE : SYNTHESE ET CONCLUSION 143

CHAPITRE III.1 - SYNTHESE D'ISDOS 144
CHAPITRE III.2 - APPRECIATION DU TRAVAIL EFFECTUE 146
CHAPITRE III.3 - EXTENSIONS 148

BIBLIOGRAPHIE

ANNEXES (2ème volume)

- Annexe I : Structuration des programmes
- Annexe 2 : Analyse des circuits
- Annexe 3 : Présentation du programme du Dynamic Interaction

INTRODUCTION

Les environnements industriels se caractérisent par une évolution de plus en plus rapide de la technologie et du marché et par des degrés d'incertitude et d'imprévision toujours accrus (1, 2, 3). Croissance et interactions de plus en plus nombreuses avec un environnement évolutif sont également le lot des organismes publics.

Face à ces changements, les organisations réagissent en élaborant des schémas de pensée et de comportement différenciés selon le type de problème appréhendé et selon les départements concernés. Cette différenciation pose des problèmes conflictuels au niveau de la collaboration interdépartementale.

Pour éviter ces conflits, l'organisation doit être considérée comme un tout, comme un système et il est indispensable de rechercher la cohérence au sein de l'organisation (notamment au niveau des objectifs).

Cette évolution des méthodes de management est résumée dans le concept de gestion intégrée : il s'agit de prendre en considération les interdépendances fondamentales dont l'organisation est le siège.

La gestion intégrée requiert la démarche de l'analyse des systèmes (1) :

elle décompose le système en sous-ensembles, identifie les éléments propres à chaque sous-ensemble ainsi que les interactions entre les différents sous-ensembles.

C'est dans cette optique d'analyse des systèmes qu'a été lancé le projet Isdos (Information System Development Operating System).

---

(1) BODART F. : "Problèmes d'Organisation et Méthodes d'Analyse Fonctionnelle"; cours FNDP, 1978-1979, Namur.
(2) LAWRENCE R. Paul and LORSCH Jay W. : "Organization and Environment", Harvard University, Boston, 1967, pp. 54-84.
(3) MARCH J.G. et SIMON H.A. : "Les Organisations", Dunod, 1969, pp. 110-131.

Isdos est le nom d'un système de gestion de l'information conçu au département d'Industrial and Operations Engineering de l'Université de Michigan sous la direction du Professeur D. Teichroew. Le but spécifique de ce système est d'aider les organisations à concevoir et à développer un système d'informations.

L'Institut d'Informatique de Namur a implémenté une version de ce système et travaille en collaboration avec l'Université du Michigan au développement des aspects dynamiques dans le but de pouvoir effectuer une simulation de système d'informations.

Les recherches menées dans ce sens ont abouti à la construction de la version NAMUR d'Isdos, version toujours en cours d'évolution.

Vis-à-vis de cette problématique, l'objet de ce mémoire est double :

- comparer trois versions d'Isdos (deux versions américaines et la version NAMUR) du point de vue des aspects dynamiques
- développer la mise en oeuvre d'un outil permettant l'analyse d'un aspect de la dynamique dans la version NAMUR.

La structure du mémoire

Comment se structure le mémoire dont nous venons de présenter l'objet ?

Le travail se divise essentiellement en deux parties.

Dans la première partie, nous présentons et comparons trois versions d'Isdos (deux versions développées aux USA et la version développée à NAMUR). Pour chacune de ces versions, nous mettons l'accent sur l'aspect dynamique.

Dans la deuxième partie, nous expliquons comment nous avons réalisé un outil d'aide à l'analyse de la dynamique pour la version NAMUR. Cette réalisation est basée sur les éléments d'un logiciel d'analyse de la dynamique développé à l'Université du Michigan.

P R E M I E R E P A R T I E

ANALYSE DES SPECIFICATIONS DANS LE PROJET ISDOS - IMPORTANCE DE LA DYNAMIQUE

Cette première partie se décompose en quatre chapitres :

Dans un premier chapitre, nous montrons comment Isdos constitue un outil d'aide à l'analyse des systèmes d'informations.

Au cours du second chapitre, nous proposons un schéma d'analyse des systèmes d'informations.

Conformément à ce schéma d'analyse, le troisième chapitre présente différentes versions du projet Isdos (deux versions développées aux USA et la version développée à NAMUR).

Le quatrième chapitre analyse plus particulièrement ces versions au point de vue dynamique.

CHAPITRE I.1 - LE PROJET ISDOS, OUTIL D'AIDE A L'ANALYSE D'UN SYSTEME D'INFORMATIONS

L'introduction a mis en valeur l'importance de la gestion intégrée et donc de l'analyse des systèmes. Ce premier chapitre, après un bref historique du projet Isdos, nous montre quelle est la structure générale de l'outil que ce projet a développé.

Il classe ensuite le langage utilisé par Isdos parmi les langages "relationnels".

I.1.1. Historique

Le projet Isdos a vu le jour à l'Université du Michigan en 1968, à l'initiative du Professeur D. Teichroew.

Ce projet avait pour but initial de résoudre un problème de conception de système d'informations selon la philosophie reprise dans l'introduction.

L'utilisateur donnerait une description de son problème au niveau fonctionnel et Isdos lui proposerait un éventail de solutions pour résoudre ce problème. Ces solutions concerneraient la configuration à adopter, la gestion de fichiers, l'implémentation du problème ainsi que l'automatisation de la documentation.

Afin de permettre les spécifications au niveau fonctionnel, Isdos a donc développé un langage de description de problèmes de systèmes d'informations, qui allait sans cesse évoluer, le langage devenant plus complexe et plus précis.

C'est ainsi que les versions 3.X furent développées pour la Royal Navy. A l'Université du Michigan furent successivement élaborées les versions 2.1, 4.2, 4.3, 5.0 et 5.1 et ce dans le cadre de programmes de recherche mais également dans une optique commerciale.

Le projet évolue actuellement dans trois directions :

- il développe les spécifications fonctionnelles au point de vue dynamique dans le but de pouvoir simuler un système à partir de sa description fonctionnelle.
- il développe un outil appelé METASYSTEME permettant aux utilisateurs de concevoir leur propre langage de spécifications.
- il développe des spécifications complémentaires adaptées à des problèmes particuliers comme, par exemple, la description d'un software.

Le présent travail concerne l'évolution des spécifications au point de vue dynamique.

I.1.2. Structure générale de l'outil

L'historique a mis en évidence l'orientation de la recherche dans le projet Isdos : développer un meilleur outil d'aide à l'analyse de systèmes d'informations.

Langefors définit les notions de "système" et de "système d'informations" comme suit (4) :

> system : = a set of objects with a set of relations
> information system : = any system in which the objects and relations are information sets and information handling objects.

Dans la perspective décrite dans l'introduction, Isdos est un outil d'aide à l'analyse d'un système d'informations. Il procède à cette analyse en deux phases (graphique 1 p. 9).

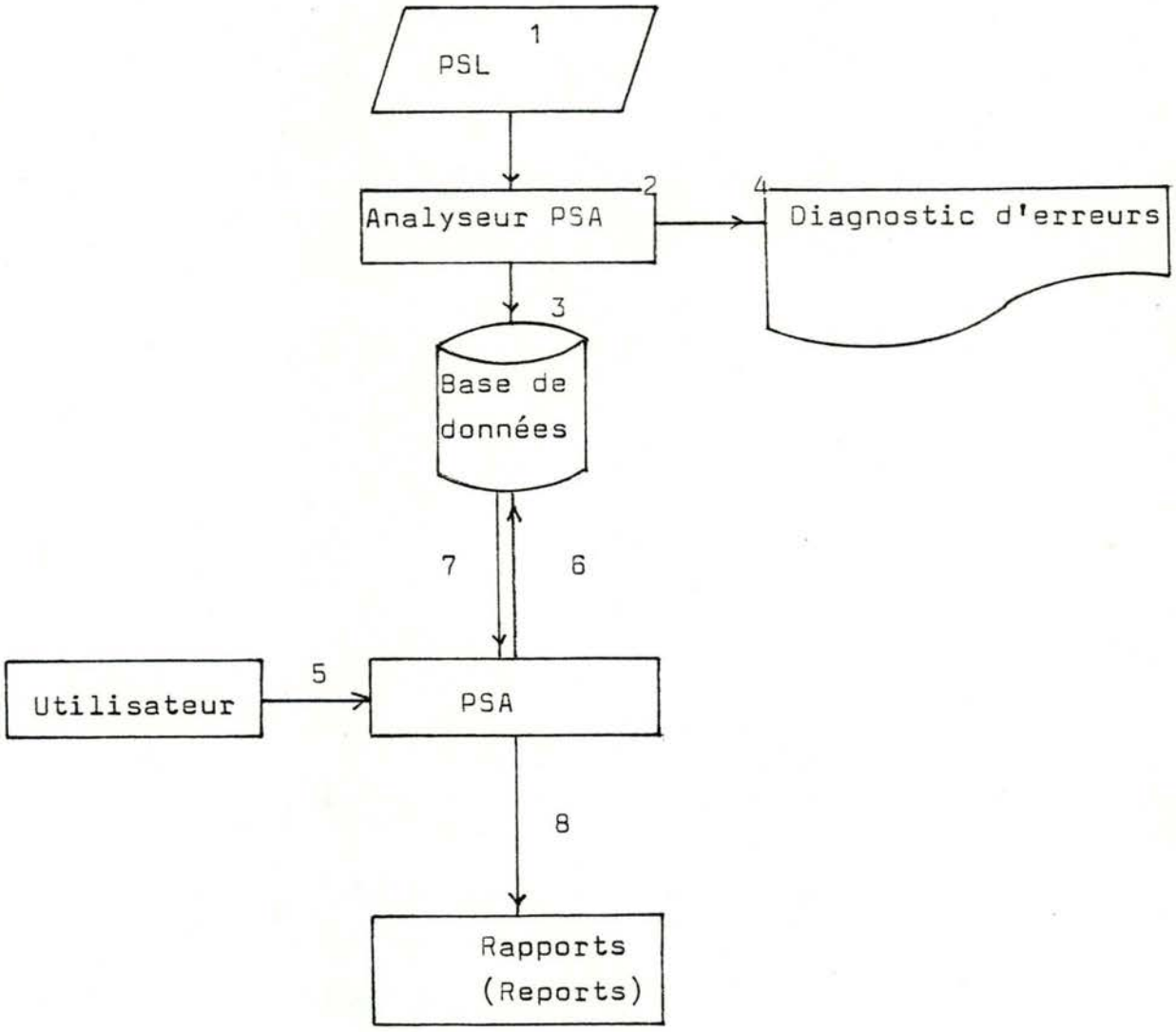
Dans une première phase, l'utilisateur décrit son système à l'aide des objets et relations faisant partie d'un langage appelé Problem Statement Language (PSL) (1 dans le graphique 1).

Dans une deuxième phase, l'utilisateur fait appel à un autre langage appelé Problem Statement Analyzer (PSA) 2. Ce langage vérifie la syntaxe de PSL et se charge d'enregistrer les informations décrites par l'utilisateur dans une base de données 3, en veillant à ce que cette base reste cohérente. Il signale les erreurs de syntaxe et les incohérences à l'utilisateur 4.
Celui-ci peut toujours retrouver les informations enregistrées dans la base de données en faisant appel au PSA 5. Il peut spécifier le type ou les caractéristiques de l'information qu'il veut retrouver. L'analyseur recherche les informations figurant dans la base 6 et opère une sélection

---

(4) LANGEFORS Borge : "Computer-Aided Information System Design" in J. Bubenko, B. Langefors and A. Solvbey (Eds), "Computer-Aided Information Systems Analysis and Design", p. 9.

Graphique 1 : Structure générale d'Isdos



selon les spécifications introduites par l'utilisateur 7 L'analyseur communique enfin les informations désirées par l'utilisateur 8 sous la forme que celui-ci a choisie.

C'est à l'utilisateur qu'incombe le choix de retrouver, avec l'aide de PSA, une information ou un ensemble d'informations qui lui permettent d'effectuer un contrôle du point de vue sémantique; ce contrôle l'amènera éventuellement à apporter les corrections nécessaires à la base de données.

Les deux phases, PSL-PSA, sont itératives. Dès qu'une information est décrite en PSL, PSA peut l'introduire dans la base de données et peut la retrouver. La description PSL d'un système d'informations peut être faite en autant de fois que l'utilisateur le désire.

I.1.3. PSL, langage "relationnel"

Nous venons de voir comment était structuré Isdos, outil permettant aux utilisateurs de concevoir un système d'informations. Les utilisateurs doivent décrire leur système à partir d'un langage appelé PSL. Dans ce paragraphe, nous analysons la structure de ce langage.

Les utilisateurs perçoivent un système à l'aide de modèles (mathématiques, économiques ...)

Leur perception de systèmes doit être formalisée de manière simple de façon à pouvoir être comprise par différents types d'utilisateurs ainsi que par les informaticiens chargés de gérer les informations ainsi décrites (5).

Le projet Isdos a développé un moyen permettant de décrire les systèmes, le langage PSL. Le langage PSL d'Isdos est un langage "relationnel" en ce sens que sa formalisation

---

(5) BENCI E., BODART F., BOGAERT H., CABANES A. : "Concepts for the Design of a Conceptual Schema", in G.M. Nyssen ed., "Modelling in Data Base Management Systems", North Holland Publishing Company, 1976.

est basée sur la description d'objets, des propriétés de ces objets et des relations existant entre ces objets (6).
Un objet représente un segment physique ou conceptuel du système à décrire. Il est décrit à l'aide d'une section PSL.
Une propriété d'un objet décrit cet objet en détails à l'aide d'un statement PSL.
Une relation décrit une association entre objets à l'aide d'un statement PSL.

Tout objet du système, défini par l'utilisateur, est accompagné de la définition de son type (classe d'objets à laquelle l'objet défini appartient). Ce type est appelé OBJECT-TYPE. Les "OBJECT-TYPE" sont en nombre limité.

Au niveau de la syntaxe, la définition d'un objet et de son type se fait dans une SECTION qui se présente de la façon suivante :

```
        OBJECT-TYPE     object-type-name (user defined name)
ex.     PROCESS         prc-facturation
```

Au stade de l'exécution du système, "prc-facturation" constitue un type d'objet. Il y aura autant d'occurrences de ce type que d'exécutions de la procédure de facturation.
Une SECTION définit donc un objet ou fait référence à un objet déjà défini. Elle spécifie les propriétés et relations relatives à cet objet à l'aide de STATEMENTS ou CLAUSES.

Les STATEMENTS sont composés d'un nom de propriété ou d'un nom de relation (ou association), suivi du ou des objets liés à l'objet défini dans la SECTION. Ils peuvent, dans certains cas, comprendre une partie optionnelle composée elle aussi d'un nom de relation suivi du ou des objets liés à l'objet défini dans la SECTION.

---

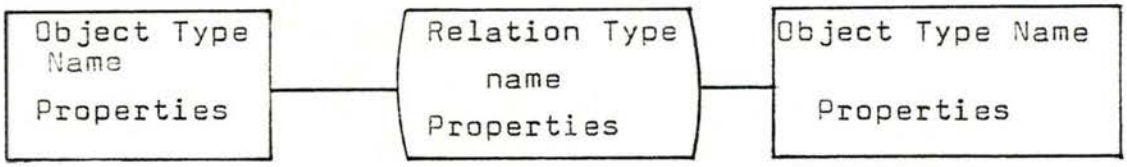
(6) Eldon D. WIG : "PSL/PSA Primer (version 4.2)", Department of Computational Science, University of Saskatchewan, Saskatoon, Canada, August 1978.

La syntaxe d'un STATEMENT est donc la suivante :

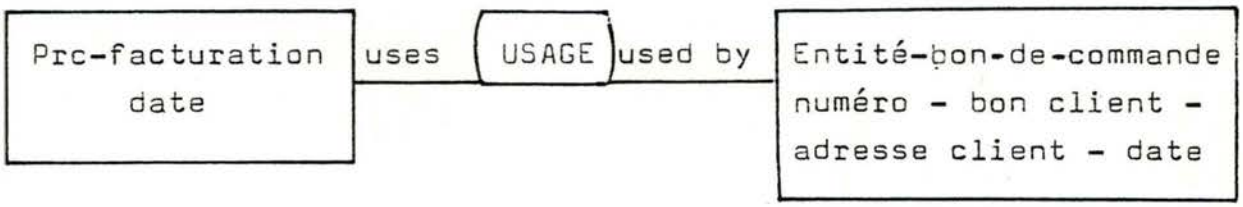
RELATION-NAME object-type-name(s) (RELATION-NAME object-type-name(s))

ex, USES entité-bon-de-commande option TO DERIVE entité-facture

On pourrait représenter certains aspects de la sémantique du langage PSL à l'aide d'un modèle de structure de données.
Ainsi, en utilisant un modèle "entité-association" (7), la représentation d'une relation de PSL serait :



Par exemple :



Une particularité très intéressante du langage PSL est d'être un langage non procédural c'est-à-dire (8)

qu'il n'existe pas de variables, d'instructions de branchement et que les instructions peuvent être mises dans n'importe quel ordre. Ceci est vrai au niveau des sections, mais également au niveau des statements qui composent ces relations.

---

(7) BENCI, BODART, BOGAERT, CABANES : op. cit. en (5) p. 10.
(8) DURIGNEUX Francis : "Mise en Oeuvre d'une Méthode d'Analyse Fonctionnelle à l'Aide du Logiciel Isdos". Mémoire, Facultés Universitaires Notre-Dame de la Paix à Namur, Institut d'Informatique, 1978.

C'est cette indépendance des statements qui permet à l'utilisateur d'introduire les informations dans la base de données en autant d'étapes qu'il le désire. La décomposition du travail d'analyse est donc très souple.

CHAPITRE I.2 - DECOMPOSITION METHODOLOGIQUE D'UN SYSTEME D'INFORMATIONS

Dans le chapitre précédent (I.1), nous avons étudié la structure générale d'Isdos, outil d'aide à l'analyse de systèmes d'informations.

Le chapitre I.3 compare trois versions différentes d'Isdos.

Dans le présent chapitre, nous explicitons une méthode de décomposition de systèmes d'informations. Cette méthode permet de mettre en évidence tous les éléments intervenant dans le système. Elle est la base de l'analyse des trois versions d'Isdos car elle nous permet de mettre en valeur les possibilités et de dégager les limites du langage associé à ces versions.

Reprenons les points essentiels de cette méthode de décomposition des systèmes :

Le principe de quasi-décomposabilité des systèmes permet de décomposer un système d'informations en sous-systèmes.

Chacun de ces sous-systèmes peut être analysé en termes de sous-structure des données et de sous-structure des traitements (9).

Une analyse supplémentaire en termes de sous-structure des ressources et une étape de quantification de chaque sous-système permettraient de définir une méthodologie d'analyse applicable à la structure du langage PSL d'Isdos.

Voyons plus en détails chaque élément de cette méthodologie.

---

(9) BODART : op. cit. en (1) p. 1.

I.2.1. Système et sous-systèmes

Nous avons déjà défini la notion de système (p. 8).

Dans le chapitre 2, nous appliquons l'analyse de décomposition aux différentes versions d'Isdos.
Pour chacune des versions, à ce paragraphe correspondra la définition du système lui-même, des sous-systèmes qui le composent et de certaines caractéristiques propres à chaque sous-système.

Nous avons vu que chaque sous-système peut être analysé en termes de sous-structure des données et de sous-structure des traitements et en sous-structure des ressources.
Explicitons successivement ces sous-structures.

I.2.2. Sous-structure des données

Chaque sous-système agit sur un ensemble d'informations. Dès lors, deux niveaux sont à distinguer :

- le niveau statique qui décrit les informations
- le niveau dynamique qui indique les actions du sous-système sur les informations décrites.

D'un point de vue statique, la sous-structure des données a pour but de définir l'ensemble d'informations du sous-système, de façon non redondante et exhaustive. Pour ce faire, elle décompose l'ensemble d'informations en différents niveaux d'agrégation et associe à chaque niveau sa description, les contraintes d'intégrité éventuelles et toute autre caractéristique qui lui est propre. Elle établit également les liens existant entre les différentes informations aux différents niveaux.

D'un point de vue dynamique, la sous-structure des données a pour but de définir la manière dont les différents systèmes exploitent l'ensemble de ces informations. Celles-

ci peuvent être créées, consultées, mises à jour et détruites. Elles peuvent être classées en :

- données exploitées en tant que moyen de communication entre le système et son environnement.
- données exploitées au sein du système ou du sous-système.

I.2.3. Sous-structure des traitements

Chaque sous-système agit sur un ensemble d'informations par l'intermédiaire de traitements.
La sous-structure des traitements peut être examinée de manière parallèle à la structure des données.

D'un point de vue statique, la sous-structure des traitements a pour but de décrire l'ensemble des actions accomplies par le sous-système sur l'ensemble d'informations associé à ce sous-système. A chaque action d'un niveau sont associés sa description, les caractéristiques qui lui sont propres, l'ensemble des actions d'un niveau plus agrégé dont elle fait partie, l'ensemble des actions d'un niveau moins agrégé qui la composent.

D'un point de vue dynamique, les différents traitements appartenant à un sous-système sont déclenchés par des événements survenant au sein du système ou provoqués par son environnement. L'aspect dynamique de la sous-structure des traitements décrit les conséquences de ces événements et l'enchaînement dynamique des différents traitements.

I.2.4. Sous-structure des ressources

La sous-structure des ressources analyse les différents types de ressources dont les traitements ont besoin pour se dérouler; elle décrit également la façon dont les ressources

sont employées.

Elle est un complément à la description de la dynamique d'un système.
Dans une perspective d'évaluation des spécifications fonctionnelles, la prise en charge des ressources constitue un aspect nécessaire de la dynamique d'un système.

I.2.5. Quantification

La quantification du système décrit les différents paramètres qui caractérisent les sous-structures précédentes. La quantification est très importante. En effet, la description d'un système d'informations doit rendre possible une estimation de la charge de travail que ce système aura à supporter. Cette quantification devient primordiale lorsque le comportement du système doit être simulé à partir de la description de ces spécifications.

CHAPITRE I.3 - PRESENTATION GENERALE DE TROIS VERSIONS DU PROJET ISDOS

Comme nous l'avons vu dans le paragraphe "historique", le projet Isdos s'est développé suivant différentes versions.

Dans la présente partie, nous analysons trois versions du langage de définition (PSL) d'Isdos : les versions 4.2 et 5.1 et la version NAMUR.

- La version 4.2 permet une description d'un système du point de vue statique.
- La version 5.1 intègre les aspects dynamiques dans la description du système.
- La version "NAMUR" développe plus spécifiquement les aspects dynamiques des systèmes dans le but de pouvoir effectuer une simulation de systèmes à partir de leur description en PSL.

Dans le chapitre I.4, nous étudions de façon détaillée les différences au niveau dynamique.
Auparavant, dans le présent chapitre, nous présentons les caractéristiques générales liées à ces trois versions, soit :

- les objectifs globaux de chacune des versions
- les domaines de spécification du PSL structurés selon la décomposition méthodologique proposée au chapitre I.2, et ce pour chaque version.
- la comparaison des domaines de spécification communs à chaque version en fonction des buts que se sont fixés ces versions.
  A ce stade, nous n'analysons pas en détails les spécifications dynamiques.
- les types de rapport existants ou proposés pour chaque version.

I.3.1. Analyse de la version 4.2 (9)

I.3.1.1. Objectifs globaux

La version 4.2 constitue un outil d'aide à l'analyse conceptuelle et fonctionnelle d'un système d'informations : sa fonction essentielle est de faciliter la gestion de la documentation d'un tel système.
Elle fournit à l'utilisateur les moyens lui permettant de décrire les spécifications de son système. Ces spécifications permettent de décrire le système particulièrement au point de vue statique et peuvent servir de base à des analyses de cohérence. Elles permettent également d'établir un dictionnaire de données. Classons ces spécifications selon la décomposition méthodologique proposée au chapitre I.2.

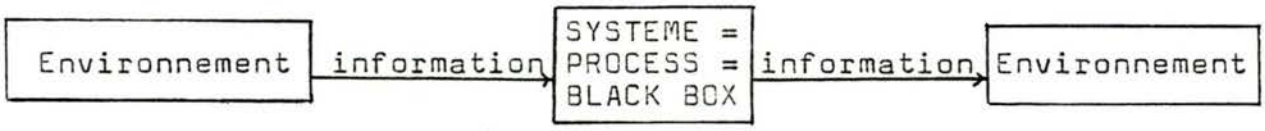
I.3.1.2. Domaines de spécification

I.3.1.2.1. Structure générale

La description du système (ensemble d'objets et de relations) est basée sur la section "PROCESS"; celle-ci définit l'objet associé au traitement d'une information.

Le système lui-même peut être considéré comme un PROCESS puisqu'il traite un ensemble d'informations qu'il reçoit de son environnement (graphique 2).

Graphique 2 : Schématisation d'un système d'informations



(9) Cfr. Bibliographie - Références à la version 4.2.

A la section "PROCESS" sont associées plusieurs clauses.

. L'une d'entre elles, la clause "SUBPARTS ARE", permet de partitionner un "PROCESS". Un système peut donc utiliser cette clause pour indiquer quels sont les sous-systèmes qui le composent.
. Il n'est pas rare qu'un système fasse appel à des processus appartenant à d'autres systèmes : il peut indiquer ses relations avec les processus de son environnement par l'intermédiaire de la clause "UTILIZES".
Par exemple, un constructeur d'ordinateurs peut utiliser les services d'une firme de software.
. Nous ne nous attardons pas sur les clauses de description proprement dite du "PROCESS" (clause DESCRIPTION), ni sur la description des caractéristiques des "PROCESS" (clauses ATTRIBUTES - RESPONSIBLE - PROBLEM - DEFINER - MAILBOX - ...)

Voyons maintenant comment peut être décrite la sous-structure des données associées à chaque sous-système.

I.3.1.2.2. Sous-structure des données

Dans la version 4.2, nous disposons d'une section "SET", définissant une collection logique d'informations. Un "SET" peut donc regrouper l'ensemble d'informations associé à un sous-système donné. Un "SET" peut être partitionné ou décomposé en structures moins agrégées à l'aide des différentes clauses qui lui sont associées :

. SUBSETS ARE : permet de partitionner le SET (graphique 3 : (1)) en sous-sets (graphique 3 : (2))
. CONSISTS OF : permet de décomposer le SET ou les sous-sets en niveaux d'agrégation inférieurs.

Voyons quels sont ces niveaux, du plus agrégé au moins agrégé en examinant le graphique 3.

Graphique 3 : Structure hiérarchique de la sous-structure des données

1. SET (1)
   (SUBSET) (2)
   2. ENTITY (3) RELATION BETWEEN entity
      AND entity (5)
   2. INPUT (4)
   2. OUTPUT (4)
      3. GROUP
         4. GROUP (6)
         4. ELEMENT
      3. ELEMENT

Les "ENTITY" (graphique 3 : (3)) peuvent être reliées deux à deux par l'intermédiaire de la section "RELATION".

La notion de "GROUP" est une notion récursive (graphique 3 : (6)).

Les sections "INPUT" et "OUTPUT" (graphique 3 : (4)) permettent de décrire les objets servant à la communication Environnement - Système (INPUT) et Système-Environnement (OUTPUT).

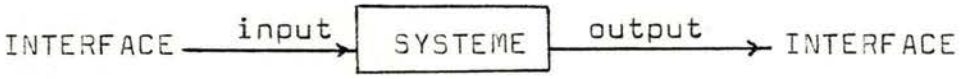
D'un point de vue dynamique, ce sont les différents objets que nous venons de décrire qui vont servir de moyens de communication avec l'extérieur du système ou à l'intérieur même de ce système.

Nous venons de définir les types d'objet "INPUT" et "OUTPUT" par rapport à leur rôle de communication avec l'environnement du système. Cet environnement peut être décrit à l'aide de la section INTERFACE. Un système peut donc être représenté selon le graphe suivant (graphique 4).

A l'intérieur du système ou des sous-systèmes, les différents niveaux de la structure de données servent de communication entre les "PROCESS". Ceux-ci peuvent créer de l'information, en transformer ou en utiliser à l'aide des relations : USES - UPDATES - DERIVES.

Quelle que soit l'action accomplie par un "PROCESS" et agissant sur le système d'informations, elle peut être décrite dans la clause "PROCEDURE".

Graphique 4 : Représentation d'un système à l'aide des concepts du PSL 4.2



I.3.1.2.3. Sous-structure des traitements

Nous avons vu que l'utilisateur peut décrire son système à l'aide de la section "PROCESS" (paragraphe I.3.1.2.1). Cette section définit l'objet associé au traitement d'une information. Tout traitement est donc décrit à l'aide de ce type d'objet "PROCESS".

a) point de vue statique

La relation "SUBPARTS" qui y est associée permet à l'utilisateur de décomposer un traitement en traitements moins agrégés (analyse top-down du système). La relation inverse "PART OF" lui permet de regrouper différents traitements en un traitement de niveau plus agrégé (analyse bottom-up du système).
Ces deux relations hiérarchisent le système et permettent de distinguer un "PROCESS" représentant le système global d'un autre "PROCESS" représentant un traitement élémentaire. Cette hiérarchisation n'apparaît pas toujours de façon très évidente dans les rapports fournis par l'analyseur (PSA) et l'utilisateur doit parfois faire des manipulations de fichiers d'informations de façon à limiter la documentation qu'il reçoit au niveau hiérarchique désiré. Cet inconvénient disparaît si l'utilisateur prend la précaution de caractériser les différents niveaux à l'aide de la clause "ATTRIBUTES".

b) point de vue dynamique

La sous-structure des traitements définit encore deux autres concepts : le concept d'EVENT et le concept de CONDITION.

Un EVENT, événement interne ou externe au système, ou le changement de valeur d'une CONDITION (passage à l'état vrai ou faux), ou le changement d'état d'un PROCESS (passage à l'état actif ou inactif) peuvent agir sur l'enchaînement dynamique des traitements par l'intermédiaire d'une seule action, l'action TRIGGERS (déclenche).

I.3.1.2.4. Sous-structure des ressources

Aucune relation dans la version 4.2 du PSL, ne permet d'associer à un traitement les ressources dont il a besoin pour se dérouler. Cet aspect ne peut donc pas être décrit dans la version 4.2 si ce n'est par l'intermédiaire de commentaires.

I.3.1.2.5. Aspect quantification

Dans la version 4.2, il existe une section "DEFINE" permettant, entre autres, de définir un paramètre représenté par le type d'objet "system-parameter". A cette section sont associées différentes clauses dont l'une permet de donner aux paramètres des valeurs plus précises, valeurs discrètes (entières) ou intervalles de valeurs (clause VALUE IS ...)

La clause "HAPPENS system-parameter TIMES-PER interval-name" associée aux sections PROCESS, EVENT, INPUT, OUTPUT rend possible la définition d'une fréquence d'occurrences, un

des facteurs-clés pour évaluer le volume d'informations que le système doit traiter. La fréquence est exprimée relativement à un intervalle décrit dans la section "INTERVAL".

I.3.1.3. Types de rapports

Nous avons analysé les concepts permettant à l'utilisateur de décrire son système. Cette description est enregistrée dans la base de données et toute information doit pouvoir être retrouvée : tel est le rôle des commandes de type "rapports" de l'analyseur (PSA) (Cfr. chap. I.1.2. Structure générale de l'outil).

La découpe méthodologique proposée au chapitre I.2 nous permet de classer les rapports du PSA version 4.2 d'après le type d'informations qu'ils permettent de retrouver, le principe général étant que la plupart des commandes nécessitent une opération préliminaire.

I.3.1.3.1. Principe général

Les commandes PSA ont pour but de retrouver les relations associées à un ensemble d'objets (object -name ou user-name). L'utilisateur peut sélectionner cet ensemble dans la base de données selon un critère qu'il définit à l'aide de la commande (PSA) NAME-SELECTION (NS). Les informations sélectionnées sont enregistrées dans un fichier qui sert d'input à la plupart des commandes (PSA) que nous analysons. Nous classifions ces commandes d'après le type d'informations qu'elles permettent de retrouver.

I.3.1.3.2. Classification des rapports suivant la découpe méthodologique adoptée

I.3.1.3.2.1. Structure générale du système

Toute la description du système est enregistrée dans la base de données et il est primordial de pouvoir connaître, à tout moment, le volume d'informations décrit.

PSA comprend une commande "Data - Base - Summary" indiquant pour chaque type d'objet (OBJECT-TYPE) :

- le nombre d'objets décrits par l'utilisateur (object-type-name)
- le nombre de synonymes définis pour chaque type d'objet
- le pourcentage que représente ce nombre de synonymes par rapport au nombre d'objets définis par l'utilisateur
- le nombre de clauses "DESCRIPTION" associées aux objets définis
- le pourcentage que représente ce nombre de descriptions par rapport au nombre d'objets définis par l'utilisateur.

D'autres rapports permettent de retrouver :

- la liste des noms définis par l'utilisateur (NAME-LIST)
- leur histoire, c'est-à-dire le nombre de modifications qu'ils ont subies et la date de la dernière modification (LIST-CHANGES).

Tout objet, quel qu'il soit, peut également faire l'objet d'un rapport "FORMATTED-PROBLEM-STATEMENT" qui indique :

- toutes les clauses que l'utilisateur a associées à cet objet
- toutes les clauses qu'il aurait pu y associer.

Ce rapport complété par l'utilisateur peut de nouveau être introduit dans la base de données et le PSA complètera la description existante.

La décomposition en niveaux du "PROCESS" peut être retrouvée à l'aide de rapports du genre STRUCTURE (présentation semblable à celle du graphique 3 p. 21) ou PICTURE (présentation graphique).

L'utilisation par le système de processus appartenant à son environnement peut être contrôlée avec le rapport UTILIZES-ANALYSIS.

Avant d'analyser les rapports propres à chaque sous-structure, examinons les rapports que ces sous-structures ont en commun.

I.3.1.3.2.2. Sous-structure des données et des traitements

La plupart des types d'objet PSL disposent de clauses leur associant un commentaire, une description ou des caractéristiques propres.
Pour retrouver cette information, divers rapports existent, rapports que l'utilisateur peut paramétrer selon son désir (DICTIONARY -KEYWORD-IN-CONTEXT - PUNCH-COMMENT-ENTRY - ATTRIBUTES ...)

Voyons maintenant les rapports propres à la sous-structure des données.

I.3.1.3.2.2.1. Sous-structure des données

La hiérarchisation de la sous-structure des données fait l'objet de rapports tels CONTENTS (analysant les relations CONSISTS - CONTAINED IN), CONTENTS-ANALYSIS, IDENTIFIER-ANALYSIS (relation IDENTIFIES), SUBSET-ANALYSIS (relation SUBSETS ARE-SUBSET OF).

Les relations unissant deux "ENTITY" peuvent également être retrouvées et analysées (RELATION - STRUCTURE).

L'aspect dynamique de la sous-structure des données fait également l'objet de rapports. Ceux-ci présentent les actions des traitements sur les données à différents niveaux, présentation matricielle ou graphique suivie selon le désir de l'utilisateur d'une analyse.

ex. DATA-PROCESS-INTERACTION , ELEMENT-PROCESS-ANALYSIS, ELEMENT-PROCESS-USAGE, EXTENDED-PICTURE.

Voyons maintenant les rapports spécifiques à la sous-structure des traitements.

I.3.1.3.2.2.2. Sous-structure des traitements

Un "PROCESS" peut être décomposé en niveaux. Cette décomposition peut être retrouvée à l'aide des rapports STRUCTURE et PICTURE (cfr. structure générale du système).

L'enchaînement dynamique de ces différents "PROCESS" peut être suivi sur un rapport graphique, le PROCESS-CHAIN.

I.3.1.3.2.3. Quantification

Etant donné que la sous-structure des ressources n'est pas décrite dans la version 4.2, nous analysons directement l'aspect quantification.

Le seul rapport analysant la quantification du système est le rapport FREQUENCY, qui analyse la clause "HAPPENS system-parameter TIMES-PER interval-name".

I.3.1.4. Conclusion de l'analyse de la version 4.2

Nous allons voir . dans quelle mesure la version 4.2 répond aux objectifs qu'elle s'est fixés
. quelles sont les limites tant au point de vue de la description du système (PSL) qu'au point de vue de l'analyseur (PSA).

I.3.1.4.1. Objectifs

La version 4.2 d'Isdos a pour objectif d'aider l'utilisateur à décrire ou à documenter un système d'informations en insistant sur les aspects statiques.

Cette description du système se fait à l'aide de concepts fort généraux. Si l'utilisateur désire décrire certains points de manière spécifique, c'est à lui de caractériser ces points, par exemple, au moyen d'attributs. Ces attributs permettent d'effectuer une description PSL en fonction de l'information que l'utilisateur veut retrouver par la suite à l'aide de l'analyseur (PSA) dans le but de documenter ou d'analyser le système qu'il a décrit.

Cet outil de description et de documentation présente, cependant, un inconvénient pour le responsable qui doit se mettre au courant de la manipulation de la version 4.2.
Ce responsable se trouve confronté à une documentation volumineuse, trop sommaire dans certains cas (résumés syntaxiques) ou trop complexe dans d'autres cas (exemples et schémas d'utilisation).
Seule une certaine pratique de PSL et PSA peut aider l'utilisateur à maîtriser la version 4.2.

I.3.1.4.2. Limites de PSL

PSL permet de décrire les spécifications du système. Nous avons vu qu'il rend possible la documentation des différents aspects de ce système. Mettons en évidence un des principaux avantages du langage PSL : il est construit sur base de termes faisant partie de l'anglais courant. Il est donc proche du langage utilisateur.

Les concepts utilisés par la version 4.2 ne sont pas très nombreux. Etant donné la simplicité de ses concepts, la version 4.2 est un excellent moyen de communication au sein d'un système car elle ne nécessite pas un apprentissage très long.

Bien que ses concepts ne soient pas tellement nombreux, la version 4.2 permet cependant la description de tout système, quel qu'il soit. Citons, par exemple, la description de la structuration de systèmes en Temps Réel (10) réalisée à l'aide de la version 4.2. Cette structuration emploie les objets (PSL) de la version 4.2 et pour chacun de ces objets, les auteurs proposent une liste d'attributs et de valeurs d'attributs ainsi qu'une liste de conventions de description. Pour chacune de ces listes, elles expliquent la signification des termes utilisés et leur utilité dans la description du système en temps réel. Signalons encore que cette description est faite non seulement d'un point de vue fonctionnel mais également d'un point de vue organique.
Cet exemple nous montre donc que toute description est possible mais au prix de conventions pouvant parfois être assez lourdes.

Certaines limites sont également apparues lors de notre analyse et notamment le fait qu'il n'est pas possible de décrire la sous-structure des ressources : il n'y a pas de

---

(10) D. DEVORKIN - C. HARTROUGH - R. OBENDORF - "Using PSL/PSA (version A4.2) for Real-Time Systems", Naval Weapons Center, June 1978, Isdos Ref # 79000-0249-0.

section prévue à cet effet, ce qui rend difficile un contrôle ultérieur de cet aspect du système. Ceci représente une limite dans le cas où l'utilisateur veut avoir une description précise des ressources, intervenant dans la dynamique des traitements. Dans d'autres cas, l'utilisateur peut toujours décrire les ressources à l'aide d'un type d'objet mis à sa disposition.

Par exemple ENTITY ressource - machine - x;
ATTRIBUTES ARE ressource réutilisable.

I.3.1.4.3. Limites du PSA

PSA travaille sur les informations décrites en PSL. Il a donc des possibilités et des limites parallèles à celles de PSL.

Il est une aide précieuse à l'utilisateur car il permet une visualisation schématique ou graphique de l'information enregistrée dans la base de données.

Les rapports présentant cette information sont généralement assez complets. De plus, l'utilisateur peut positionner un certain nombre de paramètres pour sélectionner l'information qui lui sera présentée ou la présentation même de cette information.

Certains rapports effectuent une analyse de l'information qu'ils ont présentées. Cette analyse est, cependant, limitée car seul l'utilisateur peut juger de la valeur sémantique de l'information qu'il a enregistrée dans la base de données.

Emettons encore une critique quant au rapport "NAME-SELECTION" : le critère de sélection n'offre pas assez de possibilités et l'utilisateur doit parfois retravailler le fichier à l'aide de l'éditeur pour y sélectionner les éléments à propos desquels il veut retrouver de l'information.

Prenons l'exemple d'un utilisateur qui décrit les différents niveaux des PROCESS en apposant à chaque process qu'il définit un suffixe représentant son numéro de niveau dans la hiérarchie des process.
S'il veut retrouver les process de niveau trois (qu'il a décrits user-name-3), il doit sélectionner tous les PROCESS de la base de données et avec l'éditeur, rechercher dans le fichier output du NAME-SELECTION les process dont il a besoin.
S'il ne veut pas passer par l'éditeur, il faut qu'il définisse des attributs indiquant pour tout process le niveau auquel il est situé.

La version 4.2 est donc une version qui permet de décrire les problèmes essentiellement au point de vue statique.

I.3.2. Analyse de la version 5.1 et comparaison avec la version 4.2 (11)

I.3.2.1. Objectifs globaux

L'objectif principal de la version 5.1 diffère de celui de la version 4.2 : la version 5.1 veut fournir un outil d'aide à l'analyse et à la conception de systèmes d'informations non seulement au point de vue statique mais surtout au point de vue dynamique.

Les spécifications du système peuvent être décrites de façon beaucoup plus détaillée que dans la version 4.2 du fait que le nombre de clauses est plus important. Elles permettent notamment la description du sous-système des ressources.

I.3.2.2. Domaines de spécification

I.3.2.2.1. Structure générale du système

La description de la structure du système est identique à celle de la version 4.2.

I.3.2.2.2. Sous-structure des données

La découpe hiérarchique de la sous-structure des données est pratiquement identique à celle de la version 4.2.

Signalons un apport très intéressant de la version 5.1 :

à la section RELATION est associée une clause dont la syntaxe est la suivante : "COMPOSED OF relation-name".

---

(11) Cfr. Bibliographie - Références à la version 5.1.

Cette association "COMPOSED" permet de décomposer des relations, de créer des relations à partir de relations existantes, de dégager les transitivités ou les inclusions entre relations.

Toutes les relations intervenant dans le système peuvent être analysées; le passage de la description conceptuelle à la description logique des accès, ensuite aux techniques d'accès elles-mêmes en est d'autant plus aisé et permet de choisir l'implémentation des relations réalisant les accès les plus efficaces.
Par exemple, si on a la relation

LIGNE-DE-COMMANDE (NØ CLIENT, NOM-CLIENT, ADRESSE-CLIENT, NØ PRODUIT, QUANTITE-COMMANDEE, DATE),

on pourrait dire que cette relation est composée des deux relations suivantes

CLIENT (NØ CLIENT, NOM-CLIENT, ADRESSE-CLIENT)

et

COMMANDE-CLIENT (NØ CLIENT, NØ PRODUIT, QUANTITE-COMMANDEE, DATE).

Si ces deux relations servent par ailleurs, il se peut qu'il soit très intéressant d'implémenter ces relations de façon indépendante. Si l'on peut accéder au nom et à l'adresse du client, sans devoir accéder à la commande (ou aux commandes) de ce client, on pourra, par exemple, mettre le fichier clients à jour plus facilement.

L'important dans cette relation de composition est que l'utilisateur sache bien ce qu'il entend par ce terme de COMPOSITION (s'agit-il de COMPOSITION de relations ayant des domaines communs représentés de façon identique ou de façon différente, etc.)

D'un point de vue dynamique, la communication Système - Environnement est exprimée de la même manière que dans la version 4.2.
En ce qui concerne les communications possibles à l'intérieur

du système, nous avons vu que les processus de la version 4.2 peuvent agir sur les différents objets qui permettent de définir la sous-structure des données : objets SET - SUBSET - ENTITY - INPUT - OUTPUT - GROUP - ELEMENT.
Les processus de la version 5.1 peuvent agir sur ces mêmes objets. Ils peuvent en plus agir sur les objets définis comme "RELATION" par l'intermédiaire des actions USES-DERIVES-UPDATES; cette possibilité complète le modèle de communication.

I.3.2.2.3. Sous-structure des traitements

La hiérarchisation des traitements est faite de manière identique à celle de la version 4.2.
La version 4.2 définit trois événements pouvant agir sur l'enchaînement dynamique des traitements :

- l'objet EVENT
- le changement de valeur d'une CONDITION
- le changement d'état d'un PROCESS

Dans la version 5.1, dont l'objectif est d'inclure la description de la dynamique dans le langage PSL, deux types d'objet, en plus des trois événements de la version 4.2, peuvent agir sur la dynamique; il s'agit des objets

- PROCESS
- INPUT

Les actions que peuvent accomplir ces trois événéments sont en plus de l'action TRIGGERS possible dans la version 4.2 :

- l'action INTERRUPTS (interrompre)
- l'action TERMINATES (achever)

I.3.2.2.4. Sous-structure des ressources

Un traitement, pour s'exécuter, a besoin d'un processus et de ressources.

En définissant les types d'objet "PROCESSOR" et "RESOURCE", la version 5.1 introduit la définition d'une sous-structure des ressources, que la version 4.2 ne prend pas en considération.

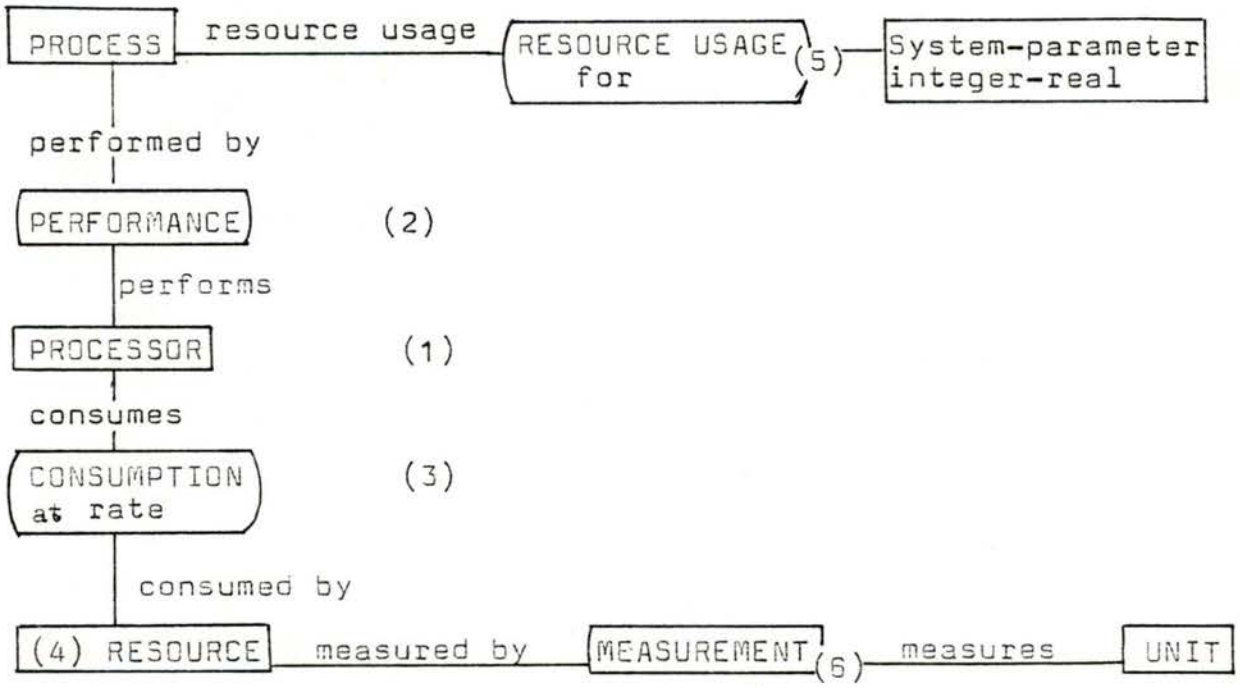
Les objets caractérisant la sous-structure des ressources ainsi que les clauses qui y sont associées sont schématisés dans le graphique 7

(1) PROCESSOR : objet servant à exécuter un "PROCESS" : pour réaliser cette exécution, le processeur peut consommer (3) des ressources

(4) RESSOURCE : objet employé par un "PROCESSOR" pour exécuter un "PROCESS".

Les relations (3), (5) et (6) permettent de quantifier la sous-structure des ressources.

Graphique 7 : Schématisation de la sous-structure des ressources (version 5.1)



I.3.2.2.5. Quantification

La version 4.2 décrit un paramètre de quantification à l'aide de la section "DEFINE".

La version 5.1 fait précéder la définition de tout type d'objet du mot "DEFINE" en sorte que la physionomie d'une section de la version 5.1 est :

DEFINE OBJECT-TYPE object-type-name;

Dans la version 5.1, le mot DEFINE permet, en fait, à l'analyseur PSA de distinguer une section d'une clause.

Au "system-parameter" défini dans la section "DEFINE" de la version 4.2, correspond, dans la version 5.1, un type d'objet ayant un certain nombre de clauses qui lui sont propres. Les possibilités offertes par ces clauses ne diffèrent cependant pas beaucoup par rapport à ce que permet la version 4.2 Seule la clause "DISTRIBUTION" pourrait permettre des nuances assez importantes dans la quantification. Mais, étant donné le fait qu'elle est considérée uniquement comme commentaire et qu'elle n'a donc pas de syntaxe spécifique, son apport est négligeable.

La fréquence d'occurrence des objets PROCESS, EVENT, INPUT et OUTPUT peut être exprimée de façon pratiquement identique dans la version 5.1 et dans la version 4.2.

Enfin, nous avons vu que la sous-structure des ressources de la version 5.1 peut être quantifiée (relations (3), (5), (6) du graphique 7 p. 35).

I.3.2.3. Types de rapports

Les rapports existant dans la version 5.1 suivent le même principe général que ceux existant dans la version 4.2. Ils peuvent être structurés de la même façon.
Seuls quelques rapports supplémentaires permettent de retrou-

ver et/ou d'analyser l'information concernant les clauses nouvellement introduites. Les autres ont seulement été adaptés à la version 5.1 et donnent parfois à l'utilisateur la possibilité de choisir une présentation un peu différente ou de demander une analyse complémentaire.

I.3.2.4. Différences essentielles par rapport à la version 4.2

Comme nous l'avons souligné, la version 5.1 poursuit un objectif très différent de celui de la version 4.2. La version 4.2 a, en effet, pour objectif de décrire un système d'informations plus spécifiquement au point de vue statique, à l'aide de concepts généraux.

La version 5.1, en plus des aspects statiques et dynamiques de la version 4.2, s'est fixé comme objectif de décrire plus particulièrement les spécifications dynamiques. C'est seulement au terme d'une étude plus détaillée de la dynamique (cfr. chapitre I.4) que nous pourrons tirer des conclusions générales vis-à-vis de la version 5.1.

Nous pouvons cependant, dès à présent, mettre en évidence quelques différences et points communs entre les versions 4.2 et 5.1.

I.3.2.4.1. Niveau des spécifications PSL

Nous avons vu que la version 4.2 décrit un système d'informations à l'aide de quelques concepts assez simples et assez généraux.

La version 5.1 est un outil dont le principe est identique à celui de la version 4.2. Le nombre de clauses associé à chaque type d'objet est cependant plus important, particulièrement au point de vue dynamique. Il permet de ce fait une description beaucoup plus raffinée du système d'in-

formations.
L'utilisateur a la possibilité de décrire des situations très complexes en les nuançant à souhait sans devoir recourir, comme dans la version 4.2, à la définition de nombreux attributs.
Le fait que ces clauses soient plus nombreuses et plus sophistiquées pourrait, cependant, limiter un des rôles essentiels joué par le PSL de la version 4.2 : le rôle de communication.

I.3.2.4.2. Niveau des rapports PSA

Tout comme dans la version 4.2, le PSA version 5.1 travaillant sur les informations décrites en PSL présente des possibilités et des limites parallèles à celles de PSL version 5.1.

Le langage PSL étant beaucoup plus étendu et beaucoup plus riche, PSA fournit donc des rapports plus détaillés. Ces rapports présentent un caractère très complet et tout comme dans la version 4.2, l'utilisateur peut positionner différents paramètres suivant l'information qu'il désire ou la présentation de cette information.

Le rapport "NAME-SELECTION" de la version 5.1 offre beaucoup plus de possibilités que celui de la version 4.2 quant aux critères de sélection, notamment, au point de vue de la manipulation des strings de caractère.

I.3.2.4.3. Niveau de la documentation

Le problème de documentation posé par la version 4.2 n'a pas été résolu dans la version 5.1.
On peut même dire qu'il s'est aggravé : en effet, la version PSL 5.1 présente plus de clauses. Certaines de ces clauses peuvent être difficiles à nuancer, ce qui pose un problème

à l'utilisateur qui ne dispose pas d'une documentation adéquate pour l'aider à distinguer les nuances de ces différentes clauses.

La version 5.1 permet donc une description de système plus détaillée que la version 4.2, avec les avantages et les inconvénients que comporte un accroissement de la spécificité des moyens mis à la disposition de l'utilisateur pour décrire un système d'informations.

I.3.3. Analyse de la version NAMUR et comparaison avec les versions 4.2 et 5.1 (12)

I.3.3.1. Objectifs globaux

La version NAMUR s'insère dans le projet Isdos. Elle poursuit donc comme objectif principal d'apporter aux utilisateurs un outil d'aide à l'analyse et à la conception d'un système d'informations.
Cependant, outre les spécifications des objets du système sur base des concepts de la version 4.2, elle s'attache tout particulièrement à une description aussi complète que possible de la dynamique du système, dans le but de pouvoir effectuer directement une simulation à partir des éléments spécifiés avec éventuellement introduction de paramètres en interactif (cfr. chap. I.4 p. 48).

I.3.3.2. Domaines de spécification

I.3.3.2.1. Structure générale du système

La structure générale du système est identique à celle de la version 4.2 et, par conséquent, à celle de la version 5.1.

I.3.3.2.2. Sous-structure des données

La découpe hiérarchique de la sous-structure des données diffère de celle des versions 4.2 et 5.1 dans la distinction des niveaux et dans la définition des concepts utilisés.

---

(12) BODART F. et PIGNEUR Y. : "A Model and a Language for Functional Specifications and Evaluation of Information System Dynamics", Working Conference, Oxford, 1979.

Niveaux et concepts des versions 4.2 et 5.1

1. SET (SUBSET)
   2. ENTITY 2. RELATION entre ENTITY
   2. INPUT
   2. OUTPUT
      3. GROUP 4. GROUP
                4. ELEMENT
      3. ELEMENT

Niveaux et concepts de la version NAMUR

1. SET
1. MESSAGE
   2. ENTITY
      3. GROUP
      3. ELEMENT
   2.GROUP 3. GROUP
            3. ELEMENT
   2. ELEMENT
   2. RELATION

La version NAMUR spécifie deux types d'objet en premier niveau :

- le type d'objet "SET" utilisé pour spécifier une collection d'informations et repris des versions 4.2 et 5.1
- le type d'objet "MESSAGE" utilisé pour représenter un ensemble d'informations INPUT et/ou OUTPUT et également pour décrire une transaction "voyageant" au travers du système.

Rappelons que les versions 4.2 et 5.1 réservent les types d'objet "INPUT" et "OUTPUT" uniquement à la communication avec l'environnement : une information entrant dans le système et utilisée à diverses étapes du traitement doit donc appartenir à un "SET".

Le graphique 4 (p. 22) associé à la représentation d'un système pour les versions 4.2 et 5.1 devient donc le graphique 8 pour la version NAMUR.

Graphique 8 : Représentation d'un système à l'aide des concepts de PSL (NAMUR)

Les autres différences dans les niveaux spécifiés dans les versions 4.2 et 5.1 et NAMUR ne portent pas à conséquence : elles donnent seulement un moyen différent d'exprimer la même information.

En ce qui concerne l'aspect dynamique de la sous-structure des données, la version NAMUR n'a conservé que quelques relations élémentaires : elle a repris les relations permettant aux processus d'utiliser (USES) des informations. Dans une perspective de simulation, il est indispensable que les processus puissent créer, modifier, supprimer des informations. La version NAMUR pourra compléter l'aspect dynamique de la sous-structure des données dans une version ultérieure, en s'inspirant par exemple de la dynamique des versions 4.2 et 5.1.

I.3.3.2.3. Sous-structure des traitements

La décomposition hiérarchique des traitements est identique à celle des versions 4.2 et 5.1.

Les événements et les actions intervenant dans l'enchaînement dynamique des traitements ont une importance capitale dans la version NAMUR vu l'objectif de simulation que cette version s'est fixé.

La dynamique des traitements est basée sur quatre catégories d'événements :

- les "EVENT" (tout comme dans les versions 5.1 et 4.2)
- les changements d'état de processus (cinq transitions sont possibles alors que deux seulement étaient possibles dans la version 5.1)
- les événements provoqués par des objets du système. Dans la version NAMUR, ces objets sont les "PROCESS" et les "MESSAGE" (Dans la version 5.1, il s'agit des "PROCESS" et des "INPUT")

. les événements complexes, combinaison d'événements simples, appelés "SYNCHRONIZATION-POINT". Ces événements constituent une originalité par rapport aux versions 4.2 et 5.1.

Pour chacune de ces catégories d'événements, quatre actions sont possibles (alors qu'une seule action est possible dans la version 4.2 et que trois actions sont possibles dans la version 5.1). Il s'agit des actions :

. TRIGGERS (déclencher)
. INTERRUPTS (interrompre)
. RESUMES (reprendre)
. ABORTS (achever)

Ces quatre actions peuvent être conditionnelles.

Nous expliciterons cet aspect dynamique dans le chapitre I.4.

I.3.3.2.4. Sous-structure des ressources

La sous-structure des ressources est un complément indispensable à la dynamique des "PROCESS".
Dans une optique de simulation, un "PROCESS" ne peut être exécuté que si les ressources qu'il réclame sont disponibles pour lui, au moment où il est prêt à être activé.

Vu l'importance de cette sous-structure dans la version NAMUR, il est compréhensible qu'elle soit décrite de façon beaucoup plus complète dans le PSL "NAMUR" que dans le PSL "5.1".

Elle définit, en plus du "PROCESS" et du "PROCESSOR" qui gardent la même signification que dans la version 5.1 - les concepts de "CONSUMABLE-RESOURCE" et de "REUSABLE-RESOURCE"

. CONSUMABLE-RESOURCE est une ressource consommable, c'est-à-dire une ressource qui ne peut être utilisée qu'une fois et qui disparaît après utilisation
. REUSABLE-RESOURCE est une ressource réutilisable, c'est-à-

dire une ressource qui peut être utilisée plusieurs fois.

Les clauses attachées à ces quatre types d'objet sont très nombreuses et ne sont pas comparables à celles de la version 5.1, tant elles affinent la description de la sous-structure des ressources.

Nous analyserons en détails dans le chapitre I.4 les principales clauses syntaxiques attachées à la section "PROCESSOR", point central de notre analyse. Nous les expliquerons sur base d'un exemple mais voyons dès à présent les informations qu'elles permettent de décrire :

Syntaxe	Signification
DEFINE PROCESSOR processor-name;	
. UNIT-OF-MEASURE IS interval-name	. reference à une unité de temps
. CAPACITY IS system-parameter SHARABLE AMONG system-parameter	. indication de la capacité du processeur
. ACTIVE FOLLOWING calendar-name (FOR process-name)	. indication du moment d'activité
. UNIT-PRICE IS system-parameter unit-name	. référence à une unité de prix
. USES (WITH EXCLUSION) reusable-resource name processor-name (AT RATE OF system-parameter) (TO PERFORM process-name)	. consommation d'un autre processeur ou d'une ressource réutilisable
. CONSUMES consumable-resource (AT FIXED RATE OF system-parameter VARIABLE (TO PERFORM process-name)	. consommation d'une ressource consommable
. PERFORMS process-name (DURING system-parameter) (PREEMPTED BY)	. processus exécuté par le processeur

La description de la sous-structure des ressources permet de connaître à tout moment les informations relatives à leur disponibilité. Ces informations sont très intéressantes pour l'utilisateur qui doit dimensionner un système qu'il a conçu.

I.3.3.2.5. Quantification

L'aspect quantification est primordial puisque la version NAMUR projette de simuler le comportement du système.

Comme dans les versions 4.2 et 5.1, c'est le concept de "system-parameter" qui est employé pour quantifier le système. Plusieurs remarques doivent être faites quant à la syntaxe attachée à ce type d'objet.

1) Dans la version "NAMUR", les valeurs prises par le "system-parameter" sont ponctuelles, alors que dans les versions 4.2 et 5.1, il est possible de décrire des intervalles de valeurs.
   La description d'intervalles représente un avantage considérable pour le responsable de la définition d'un système : en effet, ce responsable dispose la plupart du temps des valeurs minimum et maximum.
2) La version NAMUR permet à un paramètre d'être défini comme un multiple d'un autre paramètre par la clause :
   MULTIPLIED BY system-parameter;
   Cette clause permet l'expression d'un cas particulier d'une fonction exprimée sous la forme "ax". Etant donné la fréquence de ce genre de cas, on peut comprendre qu'il fasse l'objet d'une clause particulière. La question se pose alors de savoir si le cas d'une fonction de forme "ax + b" n'est pas plus fréquent.
   De plus, la consommation de ressources consommables par un processeur s'exprime sous la forme :

```
PROCESSOR  CONSUMES consumable-resource-name
  AT FIXED  RATE OF system-parameter
     VARIABLE
```

   S'il était possible de quantifier un paramètre sous forme "ax + b", l'option FIXED ou VARIABLE (RATE) se retrouverait dans la description même des paramètres, ce qui serait beaucoup plus avantageux en cas de simulation et permettrait de modifier des taux fixes ou variables sans avoir à modifier

la description d'une ressource ou d'un processeur. Une consommation ax + b pourrait être aisément modifiée en a'x + b' ou même en a"x.

3) la version 5.1 avait déjà introduit des unités de temps durant lesquelles des événements se produisaient avec un taux de fréquence différent. Le type d'objet "UNIT" comprend, en effet, une clause :

PERIOD IN WHICH event name HAPPENS system-par. TIMES
process integer
input
output

La notion de CALENDAR, dans la version NAMUR, généralise cette clause puisqu'il est permis non seulement de définir des moments d'activité mais encore des périodes. Il est possible d'indiquer un moment ponctuel mais aussi des intervalles de temps entre deux événements, des intervalles relatifs (après un événement), des périodes ou des intervalles de période relatifs.
Signalons, cependant, que les périodes relatives aux moments d'activité doivent obligatoirement apparaître après la définition de ces moments. Ces deux clauses (période et activité) auraient pu, en fait, être regroupées de façon à ce que le langage reste procédural (1).

I.3.3.3. Types de rapports

Depuis le mois de mai 1978, l'Institut d'Informatique travaille à la réalisation de rapports. Cette réalisation est basée sur les rapports existant pour la version 5.1.

Etant donné l'objectif de simulation de la version NAMUR, les rapports proposés, dans un premier temps du moins, sont ceux qui aident l'utilisateur à effectuer un contrôle sémantique sur la sous-structure des traitements qu'il a défi-

(1) Cfr. chapitre II.1, p. 101.

nie.

D'autres rapports devront encore être réalisés, rapports qui analyseront la sous-structure des ressources.

La plupart des rapports proposés sont actuellement en cours de développement.

I.3.3.4. Comparaison de la version NAMUR et des versions 4.2 et 5.1

La version NAMUR est une version toujours en cours d'évolution. Elle s'est fixé pour objectif de simuler un système à partir de sa description et a donc développé un langage de spécifications qui permet de décrire de façon détaillée les aspects dynamiques.

Il est donc difficile de comparer la version NAMUR avec la version 4.2 étant donné que la version 4.2 développe surtout les spécifications à un niveau statique.

La version 5.1, quant à elle, intègre les spécifications dynamiques et nous comparerons donc la version 5.1 et la version NAMUR dans le chapitre I.4 qui analyse les aspects dynamiques des trois versions présentés de façon plus détaillée.

Signalons dès à présent que les éléments intervenant dans la version NAMUR permettent de nuancer la dynamique des traitements et la sous-structure des ressources de façon beaucoup plus détaillée que dans la version 5.1. La signification attachée aux différents éléments n'est pas toujours évidente et c'est pourquoi la version NAMUR devra faire l'objet d'une documentation permettant à l'utilisateur de saisir les différents concepts utilisés.

CHAPITRE I.4 - ANALYSE DES DIFFERENTES SOUS-STRUCTURES AU POINT DE VUE DYNAMIQUE

La première partie de ce travail avait pour objet de réaliser une présentation générale du projet Isdos. Nous avons présenté dans les trois chapitres précédents, le projet, une méthodologie d'analyse et l'application de cette méthodologie aux différents domaines de spécification de trois versions du projet. Nous avons analysé et critiqué les sous-structures des données, des traitements et des ressources et l'aspect quantification pour chacune de ces versions.

Le présent chapitre approfondit la présentation des aspects dynamiques de chacune des sous-structures :

- il analyse les différentes façons dont les traitements peuvent agir sur la sous-structure des données
- il détaille l'enchaînement dynamique des événements et des traitements ainsi que les conditions auxquelles est soumise l'exécution des traitements
- il étudie l'aspect dynamique de la sous-structure des ressources, c'est-à-dire la manière dont les ressources sont exploitées par les traitements.

I.4.1. Version 4.2

I.4.1.1. Dynamique de la sous-structure des données - version 4.2

Les différents niveaux de la sous-structure des données servent de communication entre les "PROCESS" à l'intérieur du système ou servent de communication entre le système et son environnement. Rappelons que les différents éléments de la sous-structure des données sont pour la version 4.2 : SET - SUBSET - ENTITY - INPUT - OUTPUT - GROUP - ELEMENT - RELATION.

Les processus agissent sur la sous-structure des données essentiellement par l'intermédiaire de trois actions :

. USES : un processus peut utiliser certains éléments de la sous-structure des données. Le terme "utiliser" est assez général : il ne spécifie pas si l'objet utilisé existe toujours après utilisation.
Deux options sont attachées à la relation "USES" :
  . l'option "TO DERIVE" indique quels objets sont dérivés à partir des objets utilisés
  . l'option "TO UPDATE" indique quels objets sont mis à jour sur base des objets utilisés.

. DERIVES : un processus peut dériver des objets à partir d'autres objets.
L'option "USING" attachée à la relation "DERIVES" permet d'indiquer quels objets sont utilisés pour dériver d'autres objets.

. UPDATES : un processus peut mettre de l'information à jour.
L'option "USING" attachée à la relation "DERIVES" permet d'indiquer sur base de quels objets l'information est mise à jour.

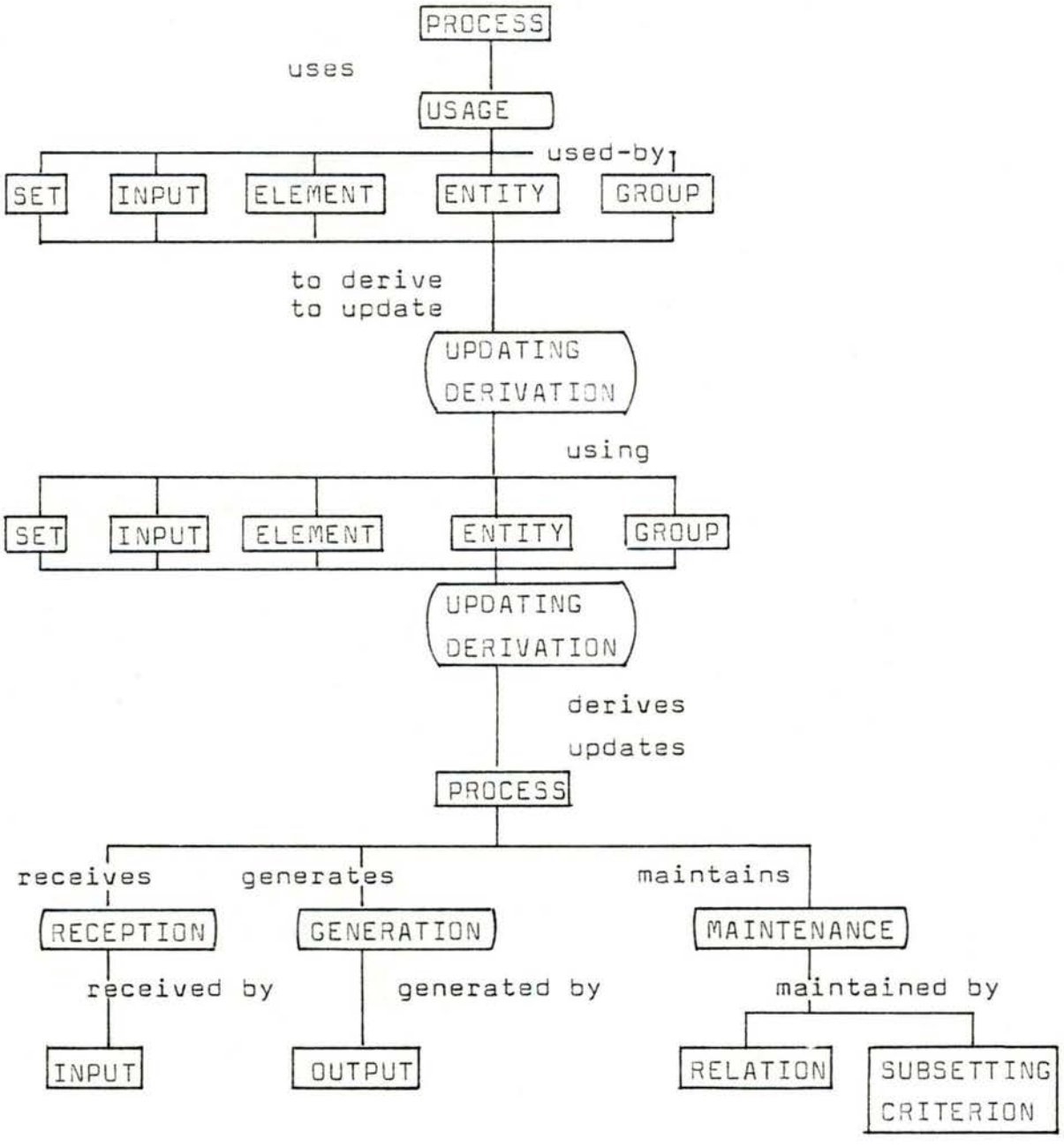
Les deux dernières relations "DERIVES ... (USING)" et "UPDATES ... (USING)" sont, en fait, une autre formalisation de la première relation "USES ... (TO DERIVE ...)"
(TO UPDATE ...)

Outre ces trois actions spécifiques, un processus peut encore agir sur une "RELATION" ou sur un critère de décomposition de set en sous-sets (SUBSETTING-CRITERION) par l'intermédiaire de la relation "MAINTAINS" : ce terme n'a pas de signification très spécifique; il peut aussi bien signifier créer, utiliser, mettre à jour.

En ce qui concerne plus spécifiquement la communication du système avec l'environnement, les processus peuvent générer (GENERATES) des "OUTPUT" ou recevoir (RECEIVES) des "INPUT".

Toutes ces relations sont illustrées dans le graphique 12 (p. 51) qui nous permet de voir combien cet aspect dynamique est développé dans une version qui a pour but principal la définition des aspects statiques d'un système.

Graphique 12 : Dynamique de la sous-structure des données
Version 4.2



I.4.1.2. Dynamique de la sous-structure des traitements version 4.2

La sous-structure des traitements est articulée autour de deux types d'objet : "PROCESS" et "EVENT". Nous avons vu qu'un "PROCESS" est l'objet associé au traitement d'une information. En terme de dynamique, lorsqu'on parle de "PROCESS", on fait allusion à une ou plusieurs occurrences de la "PROCEDURE" attachée à ce "PROCESS".

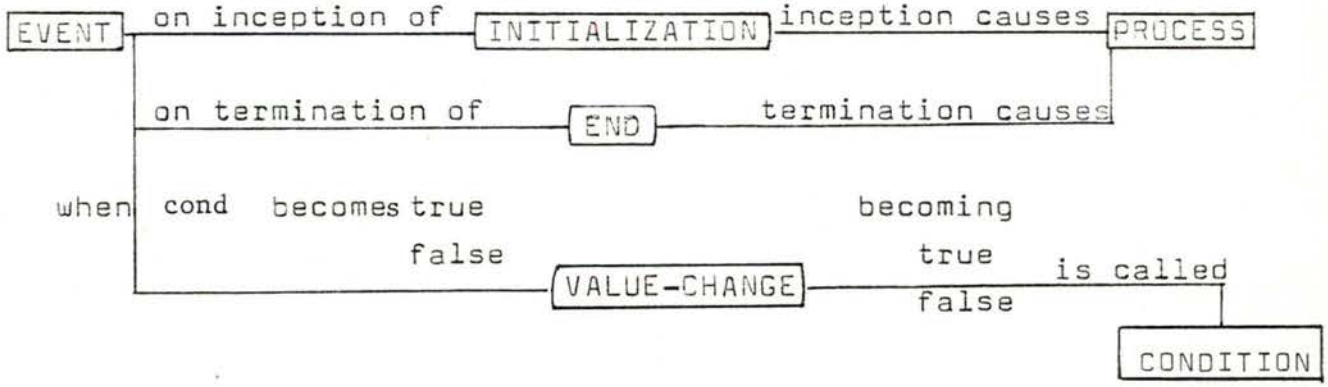
Un "EVENT" peut être défini comme un changement d'état du système. Ce changement peut survenir à l'intérieur du système ou être un effet de l'environnement du système.

Les événements (EVENT) de la version 4.2 peuvent être classés en trois catégories sur base de leur signification.

1. événements définis comme tels par l'utilisateur et indépendants du fonctionnement du système : ils sont définis à l'aide de la section EVENT qui syntaxiquement, se présente comme suit : EVENT event-name ((1) dans le graphique 13 p. 53).
2. événements caractérisés par le changement d'état d'un processus (début ou fin de processus) et dont la syntaxe est la suivante : EVENT event-name;
   ON INCEPTION OF process-name;
   ON TERMINATION OF process-name; (2)
3. événements caractérisés par le changement de valeur d'un élément conditionnel. Cet élément conditionnel est défini dans la section CONDITION. Les événements de cette troisième catégorie sont définis comme suit :
   EVENT event-name;
   WHEW condition-name BECOMES TRUE
   FALSE (3)

Dans chacune des catégories précitées, les relations inverses de celles que nous avons illustrées, existent. Nous les indiquons sur le graphique 13 (p. 53).

Graphique 13 : Graphe des événements - version 4.2



Ces trois catégories de signification sont reprises dans l'exemple qui suit (p. 54). Les numéros présentés dans cet exemple correspondent aux numéros que nous avons donné à chaque catégorie d'événements. La référence (4) apparaîtra dans la suite du texte.

Exemple de dynamique : Fonctionnement d'un système d'urgence (900)

N.B. La syntaxe du langage PSL précise que les noms définis par l'utilisateur ne peuvent dépasser plus de 30 caractères. Nous ne respectons pas cette règle dans les exemples afin de ne pas nuire à leur lisibilité. D'autre part, nous respectons la convention proposée par F. Durigneux (13) et qui consiste à préfixer chaque nom d'utilisateur par une abréviation du type d'objet.

Enoncé : comment réagit un système d'urgence lors d'un appel téléphonique ?

* EVENT ev-déclenchement-d'un-incendie (1) . Lorsqu'un incendie se déclenche, on appelle le "900".
TRIGGERS prc-appel-900 (4)

(13) DURIGNEUX F. : op. cit. en (8) p. 12.

* PROCESS prc-appel-900 INCEPTION-CAUSES ev-déclen- chement-alerte (2)	. l'appel "900" provoque la mise en état d'alerte du service
* EVENT ev-declenchement-alerte ON INCEPTION OF prc-appel-900(2) TRIGGERS prc-extinction-incendie	. une fois l'alerte déclenchée, le "900" commence à exécuter la procédure d'extinction d'incendie
* CONDITION cond-existence-de- blessés BECOMING TRUE IS CALLED ev- appel-ambulance (3)	. dès qu'on signale qu'il y a des blessés provoqués par cet incendie, le "900" envoie une ambulance
* EVENT ev-appel-ambulance TRIGGERS prc-hospitalisation (4)	. l'ambulance appelée déclenche la procédure d'hospitalisation des blessés
* PROCESS prc-extinction-incendie TERMINATION-CAUSES ev-fin- alerte	. une fois l'incendie éteint, le service quitte l'état d'alerte
* EVENT ev-fin-alerte ON TERMINATION OF prc-extinction-incendie (2)	

Nous venons d'examiner les différentes catégories d'événements; voyons maintenant quelles peuvent être les conséquences de l'occurrence d'un événement.

Tout événement peut déclencher un ou plusieurs processus par l'intermédiaire de la relation TRIGGERS. La syntaxe est la suivante :

EVENT event-name;
TRIGGERS process-name(s); voir exemple (4)

Il est important de spécifier la portée de l'utilisation d'une relation TRIGGERS. La syntaxe, en effet, considère "process-name" comme un object-type name.
Reprenons l'exemple décrit en (4) :

```
EVENT ev-appel-ambulance;
   TRIGGERS prc-hospitalisation;
```

L'appel de l'ambulance déclenche l'exécution d'une occurrence de la procédure d'hospitalisation.

On pourrait décrire les étapes de la procédure comme suit :

1. trajet en ambulance
2. entrée à l'hopital
3. procédure d'urgence

On pourrait également définir ces étapes comme autant de process faisant partie du processus hospitalisation. On aurait alors :

```
PROCESS prc-hospitalisation;
        SUBPARTS  prc-trajet-ambulance;
                  prc-entrée-hopital;
                  prc-procédure-d'urgence.
```

Si on imagine qu'une ambulance transporte trois blessés, l'événement terminant le prc-trajet-ambulance déclenche trois occurrences du prc-entrée-hopital.

La description de la dynamique des traitements se fait en termes de types d'objet, mais sous-entend une ou des occurrences des procédures associées aux traitements au stade de la réalisation du système.

Le schéma de la dynamique de la version 4.2 se résume donc au graphique 14.

Graphique 14 : Graphe de la dynamique - version 4.2

Comme nous l'avons vu lors de la présentation des spécifications, la version 4.2 est une version assez générale. La dynamique des traitements est limitée à l'essentiel.

De plus, il manque un lien : les "EVENT" et les "PROCESS" ne peuvent agir sur les "CONDITION". Seule la description de ces "CONDITION" permet de faire le lien, ce qui rend ce lien difficilement contrôlable.
Prenons l'exemple d'un contrôle de qualité situé entre la phase A et la phase B d'un processus de fabrication. PSL décrit cette situation de la façon suivante :

```
* EVENT ev-fin-phase-A;
    ON TERMINATION OF prc-phase-A;      . fin de la phase A du
                                          processus de fabrica-
                                          tion

* CONDITION cond-contrôle-de-
  qualité;
    BECOMING TRUE IS CALLED ev-         . déclenchement de la
    initialisation-phase-B;               phase B
    BECOMING FALSE IS CALLED ev-        . déclenchement d'une
    recyclage;                            procédure de recycla-
    TRUE WHILE;                           ge du produit.
      le contrôle de qualité est
      vrai aussi longtemps que le
      produit fabriqué est dans
      les normes;
    FALSE WHILE;
      le contrôle de qualité est
      faux dès qu'une des normes
      de fabrication n'est pas respectée;
* PROCESS prc-phase-B;
    TRIGGERED BY ev-initialisation-phase-B;
```

Cet exemple nous montre l'absence de lien entre les "EVENT" et les "CONDITION".
Rien ne permet, en effet, de signaler qu'après la phase A, c'est le contrôle de qualité qui intervient pour réaliser l'enchaînement EVENT-PROCESS.

I.4.1.3. Sous-structure des ressources -version 4.2

Dans le chapitre précédent (I.3 p. 23), nous avons vu qu'il n'existe aucune relation dans le PSL de la version 4.2 permettant de décrire la sous-structure des ressources à l'aide de relations spécifiques.

Dans la version 4.2, c'est dans la sous-structure des données que l'aspect dynamique est le plus développé, étant donné que cet aspect permet de faire le lien entre les différents éléments statiques utilisés. Les deux autres sous-structures sont réduites au minimum quant à leur aspect dynamique, ce qui est compréhensible étant donné l'orientation essentiellement statique de la version 4.2.

I.4.2. Version 5.1

Nous venons d'analyser les spécifications permettant la description des aspects dynamiques dans la version 4.2. Nous analysons de la même façon, dans ce paragraphe, les spécifications exprimant les aspects dynamiques de la version 5.1, aspects que la version 5.1 s'est donné pour objectif de développer.

I.4.2.1. Dynamique de la sous-structure des données - version 5.1

La dynamique de la sous-structure des données de la version 5.1 reprend les concepts dynamiques de la version 4.2.

De plus, elle ajoute certains autres concepts qui raffinent l'expression de la dynamique et que nous présentons, regroupés en trois classes :

- classe 1 : la classe 1 permet à l'objet "RELATION" d'être associé à un "PROCESS" par l'intermédiaire des relations valables pour tout type d'objet appartenant à la sous-structure des données de la version 4.2, relations reprises dans la version 5.1, à savoir :

  . USES ... (TO DERIVE ...)
  (TO UPDATE ...)
  . DERIVES ... (USING)
  . UPDATES ... (USING)

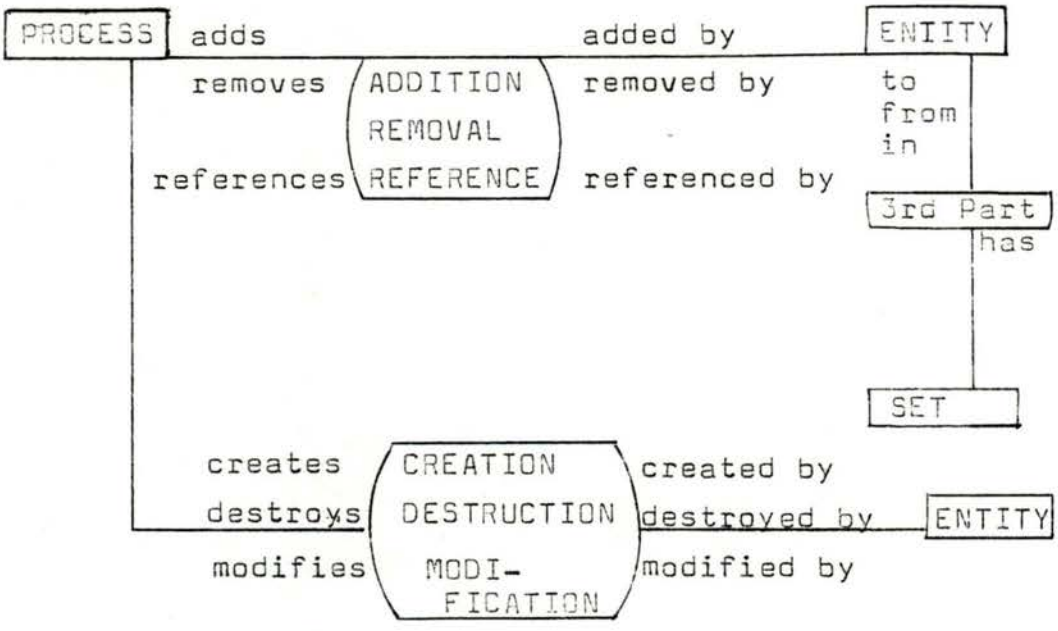
- classe 2 : la classe 2 regroupe les actions permettant à un "PROCESS" d'agir spécifiquement sur des occurrences du type d'objet "ENTITY".
  Ces actions sont la création (CREATES), la destruction (DESTROYS), et la modification (MODIFIES) d'entités.
  Les processus agissent sur une ou plusieurs occurrences d'entités, ce nombre d'occurrences étant spécifié soit à

l'aide d'un "integer" soit à l'aide d'un "system-parameter". Notons qu'il n'est pas possible de décrire quelles occurrences spécifiques sont détruites ou modifiées, si ce n'est à l'aide d'un commentaire.

- classe 3 : la classe 3 regroupe les opérations qu'il est possible d'effectuer sur les sets. Il s'agit d'un apport intéressant lors de l'utilisation d'Isdos en vue de la conception d'un système. La description des opérations sur les sets permet, en effet, de détecter les informations qui risquent de devenir plus ou moins rapidement volumineuses mais également de savoir que la place occupée par telle ou telle information ne s'accroît pas lors de la mise en oeuvre du système. Les clauses permettant la description des opérations sur les sets sont très utiles pour la gestion de fichiers.

Les clauses de la classe 1 ont une schématisation à peu près identique à la schématisation de la dynamique des données - version 4.2 (graphique 12 p. 51).
Les clauses de la classe 2 et 3 sont reprises dans le graphique 15.

Graphique 15 : Dynamique de la sous-structure des données version 5.1



I.4.2.2. Dynamique de la sous-structure des traitements version 5.1

Tout comme nous venons de le faire pour la version 4.2, examinons quels sont les différents événements qui déclenchent les processus avant d'analyser les conséquences d'occurrences de ces événements. Nous donnerons ensuite différents exemples qui nous permettront de critiquer les spécifications dynamiques de la version 5.1.

a) Evénements

Nous retrouvons dans la version 5.1 les mêmes catégories d'événements que dans la version 4.2 :

1. événements définis comme tels par l'utilisateur
2. événements caractérisés par le changement d'état d'un processus
3. événements caractérisés par le changement de valeur d'une condition

Nous devons ajouter une quatrième catégorie qui regroupe les objets PSL pouvant être la source d'une action dynamique :

4. objets PSL, sources d'actions dynamiques.

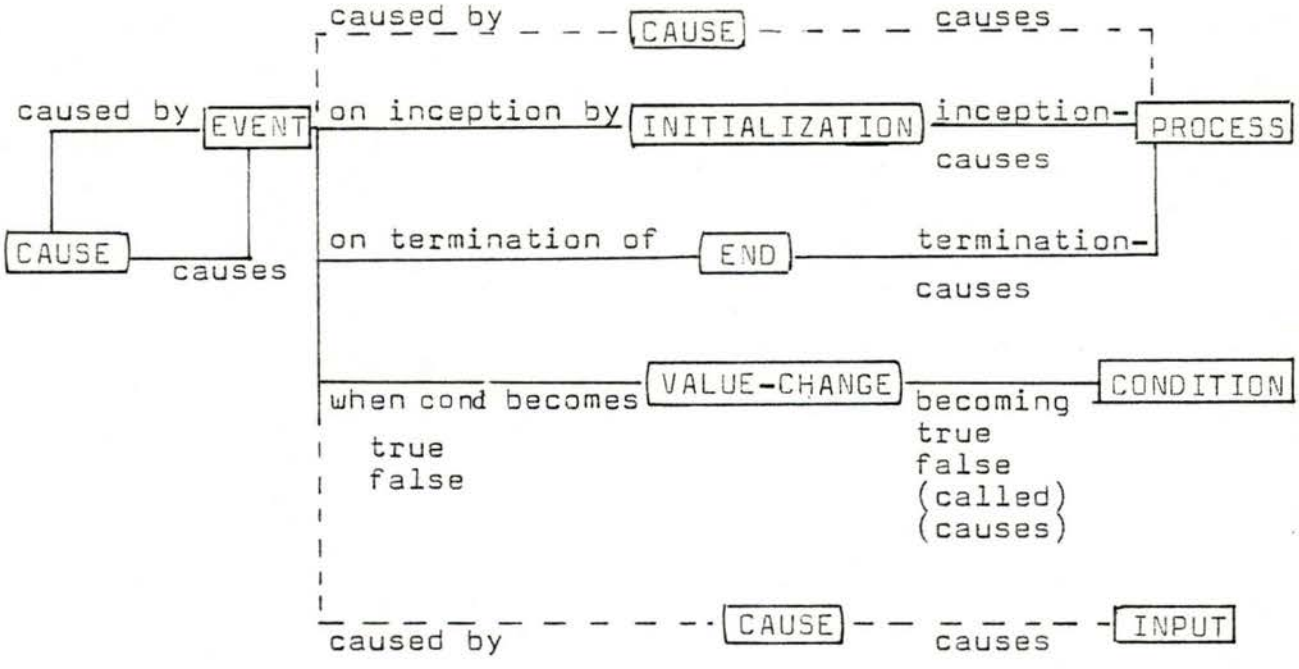
La syntaxe de la version 5.1 associe, en effet, à l'occurrence d'objets PSL tels que event, input et process un nom d'événement.
L'expression syntaxique est la suivante :

```
EVENT event-name;
   CAUSED BY  event   name(s);
              input
              process
```

Les quatre catégories que nous venons de définir font partie de l'enchaînement dynamique de la version 5.1.

Le graphe des événements de la version 4.2 (graphique 13 p. 53) est donc complété et se présente selon le graphique 16 pour la version 5.1.

Graphique 16 : Graphe des événements - version 5.1



N.B. les relations ajoutées par la version 5.1 à la version 4.2 sont représentées en pointillés.

b) Actions

Nous venons de voir quelles étaient les différentes catégories d'événements. Voyons maintenant comment ces événements agissent sur le système.

La dynamique associée à la version 4.2 est pour le moins élémentaire. La version 5.1, quant à elle, multiplie le nombre de clauses possibles pour chacune des catégories de l'enchaînement dynamique.

En résumé, nous pouvons classer ces clauses en trois catégories, suivant la portée de leur action :

- les actions visant à modifier l'état d'une condition
- les actions visant à modifier l'état d'un processus
- les actions d'ordonnancement des processus.

Nous reprenons dans le tableau 1 (p. 63) :

- ces différentes actions
- les catégories de l'enchaînement dynamique qui peuvent les provoquer
- la syntaxe PSL
- la signification de la syntaxe.

Tableau 1 : Classification des actions de la version 5.1

Type d'action	Catégorie de l'enchaî-	Syntaxe	Signification
1) actions visant à modifier la valeur d'une "CONDITION"	catégorie 4 : PROCESS	DEFINE CONDITION condition-name;	. définition d'une condition
		CHANGED BY process	. changement de valeur provoqué par un processus
	catégorie 1 à 4	MADE TRUE BY process FALSE event input	. influence d'un événement sur la valeur d'une condition
2) actions visant à modifier l'état d'un "PROCESS"	catégorie 1 à 4 OR WHEN condition-name BECOMES TRUE FALSE	DEFINE PROCESS process-name;	. définition d'un processus
		TRIGGERED BY event process input	. déclenchement du processus
		INTERRUPTED	interruption du processus
		TERMINATED	fin du processus
		WHEN cond...	
3) actions d'ordonnancement d'un "PROCESS"	catégorie 4 : PROCESS	ON TERMINATION OF process;	. à la fin du processus
		ON INCEPTION OF process;	. au début du processus
		COMES AFTER process	. se passe après le processus
		COMES BEFORE process	. se passe avant le processus

c) Exemples

Pour illustrer la dynamique des traitements dans la version 5.1, nous reprenons l'exemple décrit pour la version 4.2 (Fonctionnement d'un système d'urgence p. 53).

Nous mettons l'accent sur trois différences entre ces deux versions (4.2 et 5.1) :

1. les types d'objet dans la version 5.1 doivent être précédés du mot "DEFINE"
2. Excepté les conséquences de la remarque 1, la description de l'exemple de la version 4.2 pourrait très bien être analysée par un analyseur PSA de la version 5.1
3. la syntaxe de la version 5.1 permet cependant d'éviter la description d'étapes intermédiaires. Voyons-en un exemple :

a) description 4.2

* CONDITION cond-existence-de-blessés; BECOMING TRUE IS CALLED ev-appel-ambulance;	. quand il y a des blessés, on appelle l'ambulance
* EVENT ev-appel-ambulance; TRIGGERS prc-hospitalisation;	. l'ambulance appelée déclenche la procédure d'hospitalisation

b) description possible en 5.1 sous une forme abrégée

* DEFINE CONDITION cond-existence-de-blessés; BECOMING TRUE TRIGGERS prc-hospitalisation;	. quand il y a des blessés, on déclenche la procédure d'hospitalisation

La version 5.1 évite donc la description de l'événement faisant le lien entre CONDITION et EVENT.

Le nombre de clauses mis à la disposition de l'utilisateur dans la version 5.1 rend donc possible l'expression de situations complexes.

Voyons l'exemple d'une ménagère qui doit décider quel dessert elle va cuisiner. Cet exemple emploie certaines clauses et met notamment en évidence les rôles joués par les types d'objet :

Exemple (1)

```
* DEFINE PROCESS prc-preparation-
     du-dessert;                          . la ménagère doit pré-
                                            parer un dessert
  ON INCEPTION TRIGGERS prc-choix-        . la ménagère doit choi-
     du-dessert;                            sir le dessert qu'elle
                                            va préparer
* DEFINE PROCESS prc-choix-du-dessert     . prenons l'hypothèse que
                                            la ménagère fixe son
* DEFINE CONDITION cond-prep-milk-          choix sur la prépara-
     shake-fraise;                          tion d'un milk-shake
  BECOMING TRUE TRIGGERS prc-prep-          à la fraise
     ingredients;
  TRUE WHILE;
  cette condition est vraie lorsque
  la ménagère a fixé son choix sur
  la préparation d'un milk-shake à la
  fraise, quelles que soient les rai-
  sons qui aient influencé ce choix;
* DEFINE PROCESS prc-prep-ingredients;   . la ménagère doit pré-
                                            parer les ingrédients :
                                            lait + glace à la
                                            fraise
  ON TERMINATION TRIGGERS prc-            . une fois les ingré-
     mixage;                                dients prêts, la
                                            ménagère peut les mixer.
```

---

(1) FEMMES D'AUJOURD'HUI : "Savoir Cuisiner : le lait glacé", p. 2, 17-7-1979.

```
* DEFINE PROCESS prc-mixage;
    ON TERMINATION TRIGGERS
    prc-presentation-dessert;

* DEFINE PROCESS prc-presentation-
    dessert;
  ON TERMINATION OF prc-
    preparation-du-dessert;
```

. une fois le mixage terminé, la ménagère doit verser le mélange dans un grand verre et décorer le résultat de quelques morceaux de fraises. La préparation du dessert est achevée.

Nous venons de voir que de nombreuses situations peuvent être décrites à l'aide de la version 5.1.

Cependant, nous allons maintenant prendre deux exemples qui vont chacun nous faire découvrir certaines lacunes de la version 5.1.

Exemple 1 Le premier exemple est la description d'un problème d'ordonnancement. Nous reprenons un problème décrit dans le cours de graphes (14)
Voici l'énoncé de deux étapes de ce problème :

étape 1 : il faut obtenir un permis d'exploitation pour pouvoir commencer certains travaux

étape 2 : un de ces travaux est la construction d'une route; cette construction ne pourra commencer que cinq semaines après l'obtention du permis d'exploitation.

Si on veut employer les éléments de la dynamique et non les éléments descriptifs (ATTRIBUTES ...), la description de ce problème est la suivante :

```
* DEFINE CONDITION cond-permis-d'exploitation;
  BECOMING TRUE CAUSES ev-attente-construction-route;
```

. quand on a obtenu le permis d'exploitation, il faut attendre avant de pouvoir construire la route

---

(14) FICHEFET Jean : "Théorie des graphes", cours FNDP, 1977-1978, Namur.

* DEFINE EVENT ev-attente-construction-route;
MAKES cond-attente-délais TRUE;
. l'événement déclenche le compte-à-rebours.
* DEFINE CONDITION cond-attente-délais;
TRUE WHILE;
la condition reste vraie aussi longtemps que le délai de cinq semaines n'est pas écoulé;
BECOMING FALSE TRIGGERS prc-construction-route;
. la construction de la route peut commencer dès que le délai est écoulé.

Les étapes par lesquelles il a fallu passer pour décrire ce problème sont nombreuses et ont pour conséquence de rendre complexe une situation qui peut s'énoncer de façon simple dans le langage courant.

Exemple 2 : Le second exemple met en évidence un autre problème qui tient au fait que l'expression de la "CONDITION" a une signification presqu'exclusivement dynamique. L'exécution d'une action ne peut pas dépendre d'une condition.

Reprenons l'éxemple des blessés hospitalisés (p. 64) et ajoutons-y une hypothèse :

. dans les cas graves, les blessés sont hospitalisés
. sinon, ils reçoivent des soins

Or, l'événement déclenchant le processus d'hospitalisation ne peut soumettre le déclenchement du processus à l'état d'une condition : la syntaxe ne prévoit pas d'action conditionnelle. Nous entendons par action "conditionnelle", une action dont l'exécution dépend de la valeur d'une condition.

Dans ce cas-ci, il est cependant possible d'inclure l'hypothèse en exprimant l'exemple de la façon suivante qui consiste à inclure l'hypothèse dans la définition même de la condition.

```
* DEFINE CONDITION cond-existence-de-blessés-graves;
      BECOMING TRUE TRIGGERS prc-hospitalisation;
      . dans les cas graves, le processus d'hospitalisation
        est déclenché

* DEFINE CONDITION cond-existence-de-blessés-bénins;
      BECOMING TRUE TRIGGERS prc-soins;
      . dans les cas bénins, seuls des soins sont donnés.
```

Il n'est malheureusement pas toujours possible de résoudre le problème de cette façon et les artifices par lesquels on doit passer sont lourds à manipuler dans une description de système.

La version 5.1 est certainement beaucoup plus complète que la version 4.2. Elle permet de décrire des situations très complexes mais oblige parfois l'utilisateur à passer par des étapes intermédiaires qui ne sont pas toujours très évidentes et qui compliquent parfois des situations simples.

I.4.2.3. Sous-structure des ressources -version 5.1

La version 5.1 introduit dans la description du système la sous-structure des ressources, complément indispensable aux spécifications dynamiques. Le graphique 7 (p. 35) nous montre quelles sont les relations possibles entre les objets définis "PROCESSOR" et "RESOURCE" et le "PROCESS" qui utilise ces deux objets.

Un "PROCESS" est accompli (PERFORMED BY) par un "PROCESSOR" qui consomme des ressources à un certain taux (CONSUMES AT RATE). D'autre part, un "PROCESS" peut consommer des ressources sans passer par l'intermédiaire du "PROCESSOR" à l'aide de la clause "RESOURCE USAGE".
Il est donc possible de décrire comment les processus consomment des ressources et de la capacité "processeur". Il

n'existe cependant pas de clauses décrivant la disponibilité tant du processeur que des ressources. D'autre part, les ressources se voient associées à une unité de mesure, tandis que le processeur n'a pas d'unité de référence explicite.

Un utilisateur peut donc décrire quelle est la sous-structure des ressources de son système mais au niveau de l'exécution de ce système, les clauses de la version 5.1 ne permettent pas l'attribution dynamique du processeur ou des ressources à un processus du fait qu'il n'est pas possible de contrôler dynamiquement leur disponibilité.

Dans la version 5.1, il est non seulement possible d'effectuer une description détaillée au point de vue statique (concepts repris de la version 4.2), mais il est également possible de décrire les aspects dynamiques de façon complète. La version 5.1 répond donc à son objectif.

I.4.3. Version NAMUR

Etant donné l'objectif de simulation que s'est proposé la version NAMUR, elle a développé les aspects dynamiques du langage PSL de façon plus détaillée encore que ce que ne l'a fait la version 5.1.

I.4.3.1. Dynamique de la sous-structure des données version NAMUR

Nous avons analysé les différences existant dans l'aspect statique de la sous-structure des données de la version NAMUR par rapport aux versions 4.2 et 5.1, au chapitre I.3 (p. 18).

Nous avons également souligné à ce moment que la version NAMUR n'a repris comme relation dynamique que la relation permettant aux processus d'utiliser (USES) des éléments statiques de la sous-structure des données.

Etant donné le caractère complet de l'expression de la dynamique dans la sous-structure des données de la version 5.1, il est possible que dans une version ultérieure et plus complète, la version NAMUR s'inspire de cet aspect de la version 5.1. Rappelons, en effet, que la version NAMUR est toujours en cours de développement.

I.4.3.2. Dynamique de la sous-structure des traitements version NAMUR

Nous avons analysé en détails les spécifications de la dynamique des versions 4.2 et 5.1. Dans la version NAMUR, ces spécifications sont particulièrement importantes. En effet, c'est à partir de celles-ci que s'effectuera la simulation.

Nous suivons dans ce paragraphe la même démarche que pour les versions 4.2 et 5.1 :

a) analyse des types d'événements
b) analyse des actions

a) Evénements

Nous avons classé les événements de la version 5.1 en quatre catégories :

1. événements définis comme tels par l'utilisateur
2. événements caractérisés par le changement d'état d'un processus
3. événements caractérisés par le changement de valeur d'une condition
4. objets du système, sources d'actions dynamiques

Nous ajoutons une cinquième catégorie constituée d'événements complexes :

5. événements complexes.

Détaillons les éléments de ces différentes catégories.

1. la catégorie 1 ne subit pas de modification par rapport à la version 5.1 et contient toujours le type d'objet EVENT.
2. la catégorie 2 comprend les événements caractérisés par le changement d'état d'un processus. Ces changements d'état ne sont plus appelés explicitement EVENT.
   Il s'agit des événements : INCEPTION : début de l'exécution d'un processus
   INTERRUPTION : interruption d'un processus
   RESUMPTION : reprise d'activité d'un processus
   ABORTION : fin "anormale" d'un processus
   TERMINATION : fin d'un processus.
   Ces changements d'état permettent une description très détaillée du problème.

Leur formalisation présente des avantages et des inconvénients

. Le fait de ne pas devoir décrire d'événements intermédiaires entre PROCESS est avantageux lors de la description de problèmes dynamiques et présente un raccourci d'écriture non négligeable.
Le nombre de clauses syntaxiques PSL employées par l'utilisateur pour décrire un problème est donc moins élevé par rapport à la situation où l'utilisateur devrait passer par le concept d'EVENT.

. Il existe cependant un désavantage qui réside dans le fait que le nombre de clauses syntaxiquement possibles est plus élevé.
En effet, on a actuellement les possibilités syntaxiques suivantes :
chaque changement d'état (ON INCEPTION ... ON TERMINATION) peut se voir associer quatre actions; nous avons donc 5 x 4 = 20 clauses possibles.
Si on associait explicitement à chaque changement d'état le concept d'EVENT nous aurions cinq clauses libellées de la façon suivante :

```
ON INCEPTION
ON INTERRUPTION
ON RESUMPTION          IS CALLED event-name    (1)
ON ABORTION
ON TERMINATION
```

Pour rendre cette proposition réalisable, il faudrait que la syntaxe de EVENT permette les mêmes actions que celles permises pour les différents changements d'état d'un processus.
Par exemple, seul le changement d'état "TERMINATION" d'un PROCESS peut générer des messages.
Sa traduction à l'aide de la solution (1) serait :

```
DEFINE EVENT event-name;
   GENERATES message-name (IF condition-name IS TRUE )
                                                FALSE
```

Pour que cette traduction soit tout à fait correcte, il faudrait soumettre cette dernière clause (GENERATES) à une restriction éventuelle sur le type d'événement pouvant l'utiliser. Or, il est très difficile d'introduire des restrictions syntaxiques de ce genre dans le langage PSL du fait que de telles restrictions nuiraient à la facilité d'utilisation du langage.

D'autre part, le passage par la notion d'EVENT (1) rend nécessaire une étape supplémentaire au point de vue écriture mais peut aider à mieux localiser les changements à apporter en cas de modification.
La syntaxe actuellement d'application :

```
DEFINE PROCESS prc-p;
ON INCEPTION TRIGGERS prc-x;
ON INCEPTION TRIGGERS prc-y;
```

deviendrait :

```
DEFINE PROCESS prc-p;
   ON INCEPTION IS CALLED ev-inc-p;
DEFINE EVENT ev-inc-p;
   TRIGGERS prc-x;
   INTERRUPTS prc-y;
```

. introduction d'un nouveau type d'objet = début du process prc-p

. les actions déclenchées par le début du processus sont centralisées.

Etant donné l'importance de l'avantage de la solution actuelle - raccourci d'écriture - et les avantages que pourrait procurer la solution proposée, il serait intéressant de pouvoir combiner ces deux solutions.
On pourrait, par exemple, prévoir qu'un rapport (par exemple, le FORMATTED-PROBLEM-STATEMENT) accomplisse le passage de la solution actuelle à la solution préconisée. L'utilisateur ne devrait qu'écrire l'étape 1 et le passage à l'étape 2 serait fait automatiquement.

3. La troisième catégorie regroupant les événements caractérisés par le changement de valeur d'une condition disparaît dans la version NAMUR.
Signalons dès à présent qu'une CONDITION n'a plus la même signification que dans la version 5.1.

Nous avons vu dans la version 5.1 que le changement d'état d'une condition est appelé explicitement EVENT. La sémantique d'une condition est donc essentiellement dynamique.

Dans la version NAMUR, par contre, la notion de condition est statique et permet l'expression d'actions conditionnelles c'est-à-dire dont l'exécution dépend de la valeur d'une condition. La "CONDITION" de la version NAMUR a donc une valeur consultative plus proche de la notion de condition ou test en programmation : selon son état au moment du test, telle ou telle action sera ou non accomplie.

Par ailleurs, la version NAMUR a introduit un objet appelé "SYNCHRONIZATION-POINT" qui généralise, entre autres, la notion dynamique de la "CONDITION" de la version 5.1. Cet objet constitue en fait, la cinquième catégorie d'événements (cfr. p. 71).

4. La quatrième catégorie reprend les objets du système, sources d'actions dynamiques. Nous venons d'analyser l'objet "EVENT" et nous analyserons dans la cinquième catégorie l'objet "SYNCHRONIZATION-POINT". Cette catégorie reprend donc le seul type d'objet "PROCESS".

Nous avons vu dans la catégorie 3 que chaque changement d'état de ce "PROCESS" est un événement pouvant agir sur le système. Un "PROCESS" peut également agir en cours d'exécution en générant des informations appelées "MESSAGE". Les "MESSAGE" sont générés lorsque les processus ont une durée d'exécution assez longue. Ils rendent compte au système de la progression de cette exécution et ne dépendent donc pas d'un changement d'état de processus.
D'autre part, à chaque terminaison de processus est associée implicitement la génération d'un "MESSAGE". Cette asso-

ciation peut également être explicite.

La génération d'un "MESSAGE" peut elle aussi agir sur le système et peut, à ce titre, être reprise dans les événements sources d'actions dynamiques.

Nous venons donc d'examiner les quatre catégories d'événements qui étaient définies pour la version 5.1 et reprises partiellement dans NAMUR.

Avant d'analyser les actions possibles pour ces différents événements, il nous reste un type d'objet à définir. Ce type appelé "SYNCHRONIZATION-POINT" constitue la cinquième catégorie d'événements. Il est introduit dans la version NAMUR et permet de synchroniser dynamiquement des processus. Il a donc toute son importance dans une optique de simulation (15).

5. Catégorie des événements complexes

Un point de synchronisation "SYNCHRONIZATION-POINT" est un événement complexe c'est-à-dire une combinaison d'événements simples, agissant sur le système. Est considéré comme événement simple un événement appartenant à une des catégories que nous venons de définir. Un point de synchronisation est réalisé lorsque tous les événements qui le composent sont réalisés.
Il mémorise les différents événements réalisés, pendant un certain temps défini pour chaque événement : il constitue donc un point d'attente. Il agit sur la dynamique du système lorsque ses conditions de réalisation sont remplies.

La participation des événements à un point de synchronisation se fait par l'intermédiaire de la clause "CONTRIBUTES". Cette clause est attachée à chacune des catégories d'événement comme l'indique le graphique 17 (p. 77).
Elle comprend deux options :

. la première option soumet la participation d'un événement à la valeur d'une condition. La "CONTRIBUTION"

---

(15) BODART F. et PIGNEUR Y. : op. cit. en (12) p. 40.

est donc une action conditionnelle.

. la deuxième option permet à un événement de participer à un point de synchronisation après un certain intervalle de temps, ce qui facilite la description de problèmes d'ordonnancement.

Le graphique 17 (p. 77) nous montre quels événements peuvent participer à un point de synchronisation. La façon dont ils y participent est décrite dans une clause "PREDICATE", considérée jusqu'à présent comme un commentaire. Les auteurs de la version NAMUR proposent toutefois de décrire le "PREDICATE" comme une expression booléenne d'événements simples.

Le point de synchronisation est donc un événement complexe qui, lors de sa réalisation, peut provoquer différents changements d'état du système. Dans cette perspective, est-il possible de le comparer avec la notion de "CONDITION" de la version 5.1 ?

La combinaison d'événements (clause PREDICATE de la version NAMUR) caractérisant le point de synchronisation peut être reprise dans les clauses de commentaires (DESCRIPTION, TRUE WHILE, FALSE WHILE) de la "CONDITION", version 5.1.

La différence essentielle entre ces deux concepts réside dans la notion d'attente qui est incluse dans la sémantique du point de synchronisation : il mémorise les différents événements et une fois qu'une combinaison d'événements simples le définissant est réalisée, il agit.

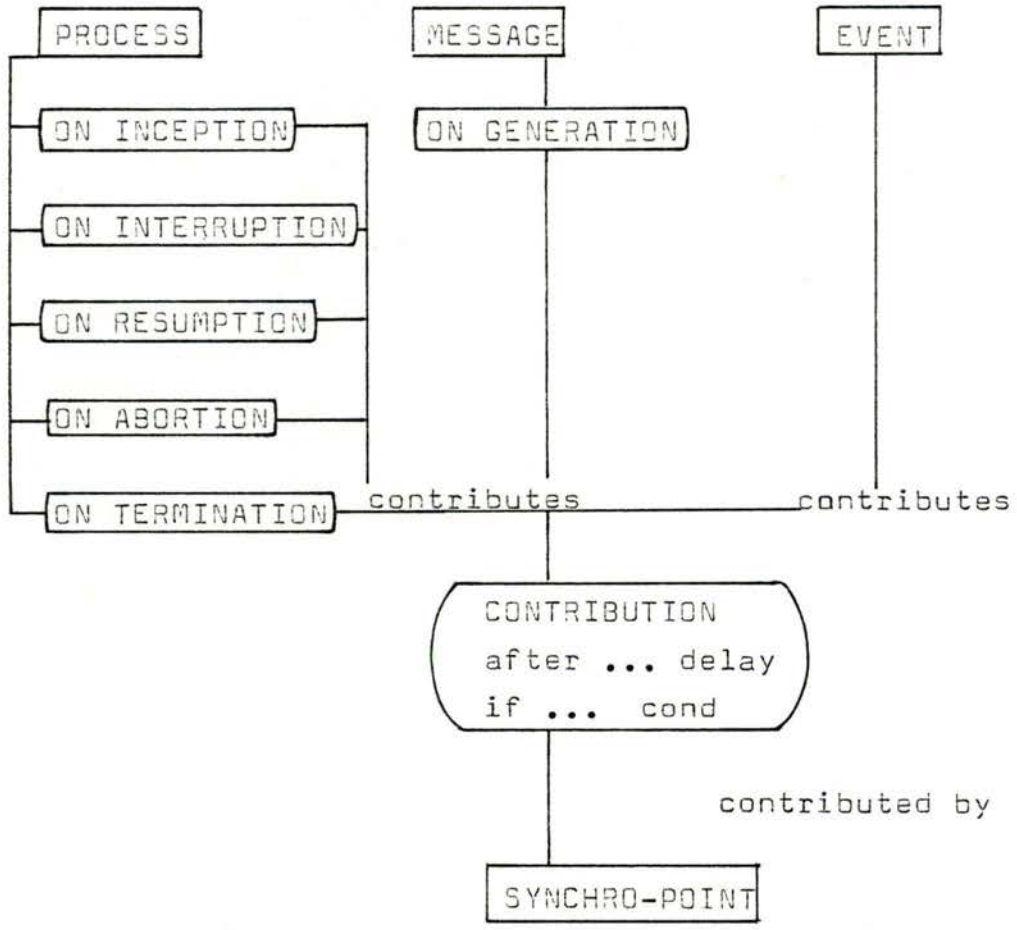
Signalons qu'un événement contribue à un point de synchronisation soit . pendant une durée illimitée

soit . jusqu'à ce que le point de synchronisation soit réalisé

soit . jusqu'à ce qu'un événement, indépendant de la réalisation du point de synchronisation, se soit passé dans le système.

L'exemple suivant nous montre combien peut être utile l'indication d'une durée : soit une fabrique de produits pharmaceutiques ou chimiques chargée de la fabrication d'un produit C, composé du produit A et du produit B de durée de validité respective 3 minutes et 10 minutes. Le mélange A + B ne peut être effectué que lorsque les deux produits sont valides. "A + B" peut être défini comme un point de synchronisation auquel le produit A, dès sa fabrication, participe pour une durée de 3 minutes et auquel le produit B, dès sa fabrication, participe pour une durée de 10 minutes.

Graphique 17 : Graphe du point de synchronisation



b) Actions

Analysons maintenant les actions attachées aux catégories d'événements que nous venons de définir. Rappelons que ce sont ces actions qui modifient l'état d'un processus.

La version NAMUR a essayé d'introduire une certaine standardisation dans les spécifications PSL et pour ce faire, a donné à toutes les sources d'actions le même statut. Ainsi quatre actions peuvent être effectuées par chaque élément appartenant à une des catégories précitées, à savoir :

- TRIGGERS = déclenche
- INTERRUPTS = interrompt
- RESUMES = reprend
- ABORTS = achève

Si un événement déclenche (TRIGGERS) un processus prc-p, cette action a pour effet de créer une nouvelle occurrence du type d'objet prc-p et de l'activer. Si l'action "INTERRUPTS" est déclenchée vis-à-vis d'un processus, l'occurrence de ce processus retourne momentanément à l'état inactif, état qui reprend une forme active lorsque l'action "RESUMES" est mise en oeuvre. L'occurrence disparaît lorsqu'un événement l'achève "ABORTS" ou lorsqu'elle arrive à la fin d'exécution de la "PROCEDURE" (TERMINATION d'un processus).

L'action TRIGGERS a donc pour effet d'introduire une nouvelle occurrence dans le système. Les trois actions, "INTERRUPTS", "RESUMES", "ABORTS", changent l'état dans lequel se trouvent des occurrences qui ont été créées préalablement.

Cette différence dans les fonctions des actions explique que des options différentes soient attachées à ces actions :

- TRIGGERS dispose des options . FOR EACH set-name
  system-parameter
  . IF condition-name IS TRUE
  FALSE

- INTERRUPTS
  RESUMES
  ABORTS

disposent des options

. WHERE condition-name
. IF condition-name IS
  TRUE
  FALSE

Expliquons la signification de ces différentes options.

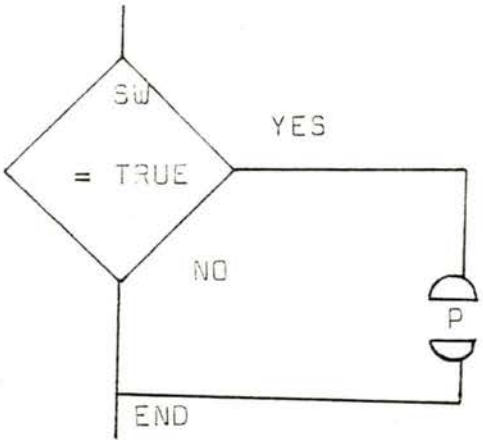
1. Evenement-e : TRIGGERS prc-p (FOR EACH set-name)
                                    system-parameter

L'événement "Evenement-e" crée un nombre d'occurrences de processus prc-p égal au nombre d'éléments dans le set-name ou à la valeur spécifiée par le system-parameter.
Par exemple, la fin du mois déclenche le payement des salaires pour chaque employé d'une entreprise. Cet exemple peut être décrit de la façon suivante :

```
DEFINE EVENT  ev-fin-du-mois;
   TRIGGERS prc-payment-salaire
            FOR EACH set-signalétique-employé
         ou FOR EACH sys-p-nombre-employés
```

2. Evenement-e TRIGGERS
               INTERRUPTS
               RESUMES
               ABORTS
   prc-p (IF condition-name IS (TRUE)
                               (FALSE)

Nous avons signalé que la "CONDITION" n'a pas du tout la même signification dans la version NAMUR et dans les versions 4.2 et 5.1. Dans la version NAMUR, en effet, elle représente un concept statique. Elle est plus proche de la notion de "condition" employée généralement en programmation : une condition possède une valeur à un moment déterminé, elle est testée et suivant le résultat du test, telle ou telle action est déclenchée.
Démontrons-le en transposant un algorithme dans le langage Isdos : soit dans le courant d'un programme, un switch SW prenant une valeur FALSE lorsqu'une erreur grave a été détectée, erreur qui empêche le déroulement normal de la procédure P.



Isdos décrit cette situation de la façon suivante :

```
DEFINE EVENT ev-exécution programme;
     TRIGGERS prc-appel-procedure-p IF cond-SW IS TRUE;
     ABORTS prc-execution-programme IF cond-SW IS FALSE.
```

3. Evenement-e INTERRUPTS prc-p (WHERE condition-name)
   RESUMES
   ABORTS

Nous avons vu que ces trois actions agissent sur des occurrences de processus. Comme il peut y avoir plusieurs occurrences d'un processus à un moment donné dans le système, au cours de la simulation, il s'agira de préciser sur quelle(s) occurrence(s) de ce processus l'action INTERRUPTS, RESUMES ou ABORTS va porter.

La solution à ce problème est apportée par l'option "WHERE condition-name". L'action va porter sur les occurrences de prc-p pour lesquelles apparaît la condition spécifiée dans l'option. Cette action peut elle-même être conditionnelle.

Au cas où l'option "WHERE condition-name" n'est pas spécifiée, il a été pris comme convention que l'Evenement-e agit sur toutes les occurrences de prc-p.

Voyons, par exemple, comment peuvent être décrites les conséquences d'une grève dans le traffic ferroviaire de marchandises :

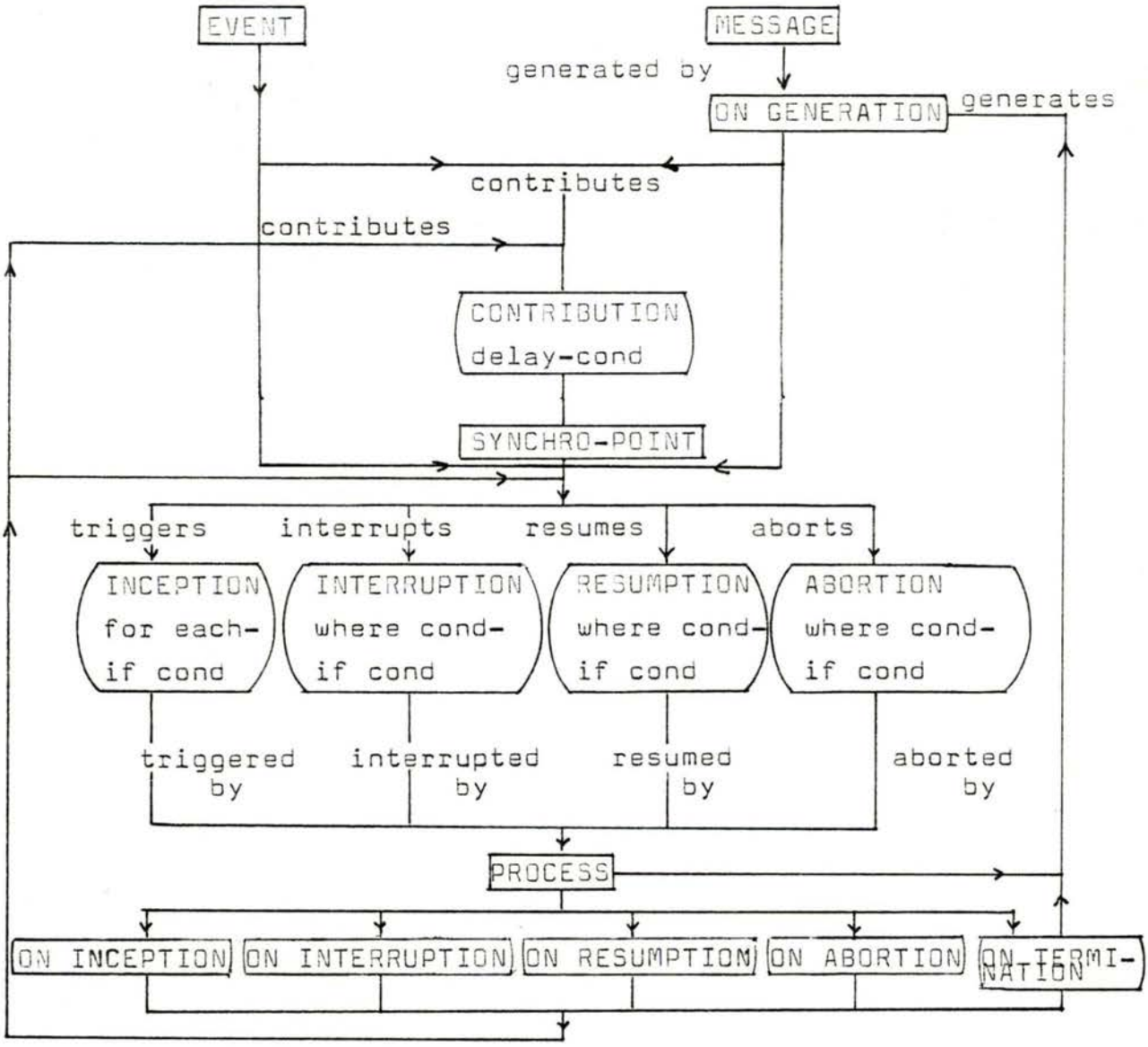
- le transport de toute marchandise non périssable est interrompu pendant la durée de la grève
- le transport par train sera définitivement arrêté pour les marchandises périssables si leurs délais de validité sont inférieurs au temps pendant lequel la grève se poursuit.

La syntaxe décrit ce problème comme suit :

```
DEFINE CONDITION cond-péremption;
    PREDICATE;
        condition vraie quand la date de péremption de la
        marchandise est antérieure à la date prévue de
        fin de grève;
DEFINE EVENT  ev-greve-traffic-ferroviaire;
        INTERRUPTS prc-transport-par-train WHERE cond-non-
                                        perissable;
        ABORTS prc-transport-par-train WHERE cond-perissable
                                IF cond-peremption IS TRUE;
```

Les actions et options que nous venons d'analyser figurent sur le graphique 18 (p. 82) qui reprend la dynamique des actions de la version NAMUR.

Graphique 18 : Dynamique de la version NAMUR



c) Inconsistance dynamique

Nous venons d'analyser quels sont les événements et actions intervenant pour décrire la dynamique de la sous-structure des traitements dans la version NAMUR.

Nous proposons de terminer cette analyse de la dynamique par l'étude d'un problème que nous n'avons pas soulevé pour les versions 4.2 et 5.1 : le problème d'inconsistance dynamique.

L'objet de ce paragraphe est

. de définir ce problème
. d'examiner les raisons de son analyse
. de regrouper les différents cas où il risque d'apparaître.

L'inconsistance dynamique concerne les erreurs que l'utilisateur peut commettre en décrivant la dynamique de son système.
Reprenons le graphique 18 (p. 82) de la dynamique de la version NAMUR.
Un certain danger d'incohérence apparaît dans la possibilité qu'il existe des circuits.

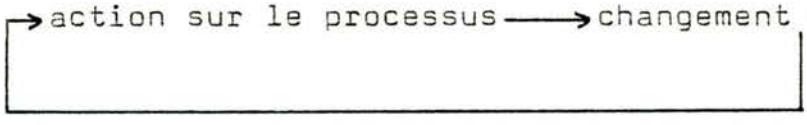
Ce problème, important pour les versions 4.2 et 5.1 acquiert dans la version NAMUR une importance capitale : en effet, rappelons que l'objectif de cette version est de simuler le comportement du système à partir de sa description. Celle-ci doit donc être contrôlée par l'utilisateur afin que la simulation porte sur un système cohérent et donc que les résultats soient exploitables.

Détaillons un peu ces cas d'incohérence.

Les boucles ou circuits peuvent exister à plusieurs niveaux :

- soit boucle directe : événement-a action événement-a
- soit boucle indirecte : événement-a action événement-b
  événement-b action événement-a
  . l'événement-b pouvant, en fait, être constitué d'une chaîne d'événements.

Voyons quelles boucles sont possibles dans la version NAMUR :

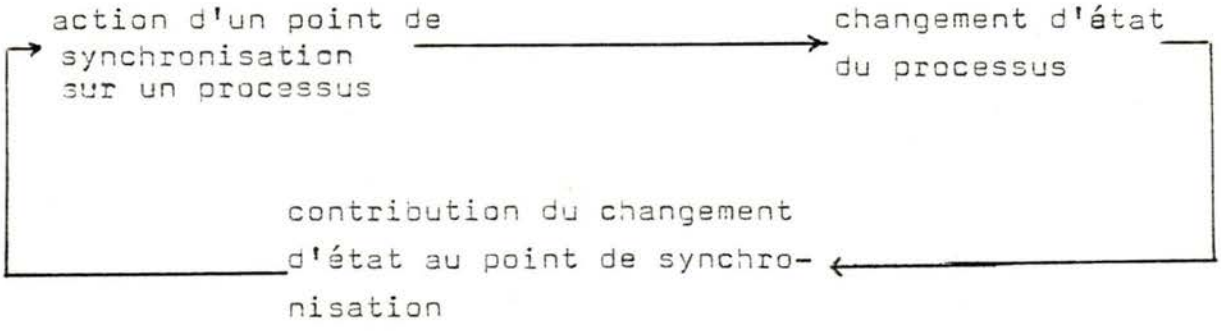
1. 

Cette boucle peut être . directe ex resumption of process a
triggers process a
. indirecte ex inception of process a

triggers process b
process-b interrupts process a

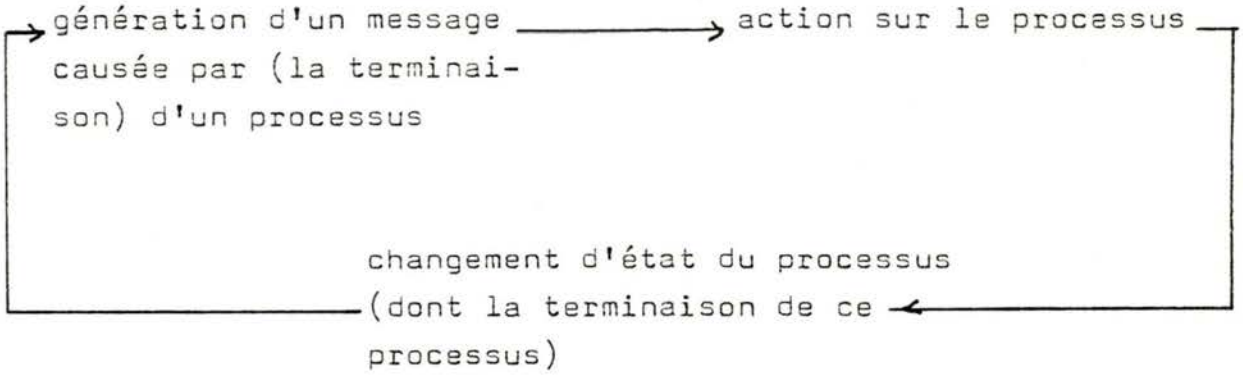
Certaines situations peuvent être cohérentes, d'autres non. Nous examinons deux cas de boucle cohérente après notre examen des cas possibles.

2.



Ce cas-ci est <u>incohérent par définition</u> même du point de synchronisation.

3.



Tout comme dans le premier cas, il est impossible de juger de la cohérence de cette situation <u>en-dehors d'une</u> situation particulière.
Nous revenons sur ces problèmes dans la deuxième partie de ce travail.

<u>La syntaxe ne prévoit aucune restriction</u> concernant ces boucles.

Nous pensons qu'une <u>seule règle</u> générale permet d'accepter la cohérence d'une boucle (en-dehors de la signi-

fication attachée aux événements particuliers intervenant dans la boucle) :

"au moins une des actions de la boucle

. doit être conditionnelle (IF condition-name-...)

ou

. doit être achevée par l'occurrence d'un événement (ABORTED BY)"

Si cette règle n'est pas respectée, la boucle est sans fin et il n'est pas possible d'effectuer une simulation cohérente du système.

Nous proposons donc de signaler ces boucles à l'utilisateur dans un rapport PSA, DYNAMIC INTERACTION (cfr 2ème partie). Il appartiendra dès lors à l'utilisateur de vérifier la cohérence de la description dans le cas particulier qu'il étudie.

Voyons quelques exemples de circuits qui nous aident à imaginer pourquoi la syntaxe ne peut logiquement pas imposer à un utilisateur des contraintes l'empêchant de décrire des circuits dans son système.

Considérons par exemple les systèmes auto-régulateurs : les conséquences d'un changement dans une des variables tendent à contrecarrer la variation initiale (16).

Prenons donc l'exemple d'une forêt.

En l'absence d'intervention extérieure, la densité des arbres tend vers un certain optimisme, fonction de l'ensoleillement et de l'espace vital. Si, pour une raison quelconque, la densité diminue, les graines tombées des arbres voisins ont plus de chance de trouver humidité, lumière et espace vital, et donc de germer. La boucle tend alors à augmenter. Mais, si elle augmente trop, les graines ne trouvent plus assez de ressources vitales et meurent sans germer, ce qui associé au dépérissement naturel de la forêt, tend à diminuer sa densité.

---

(16) POPPER Jacques : "La Dynamique des Systèmes", Les Editions d'Organisation", Eyrolles éditeur - Paris 1973 - p. 79.

Cet exemple peut être décrit à l'aide de la version NAMUR, de la façon suivante :

```
* DEFINE PROCESS prc-diminution-densité-des-arbres;
      ON INCEPTION TRIGGERS prc-accroissement-conditions-
      vitales;
      . lorsque la densité des arbres diminue, humidité,
        lumière et espace vital s'accroissent.
* DEFINE PROCESS prc-accroissement -conditions-vitales;
  ON INCEPTION TRIGGERS prc-croissance-nouvelles-pousses;
  ABORTED BY TERMINATION OF prc-diminution-densité-des-arbres;
  . lorsque les conditions vitales s'accroissent, de nouvelles
    pousses peuvent germer.
    Les conditions vitales cessent de s'accroître lorsque la
    densité des arbres cesse de diminuer.
* DEFINE PROCESS prc-croissance-nouvelles-pousses;
      ON INCEPTION TRIGGERS prc-accroissement-nombre-d'arbres;
      . Les nouvelles pousses provoquent un accroissement du
        nombre d'arbres.
* DEFINE PROCESS prc-accroissement-nombre-d'arbres;
      ON INCEPTION TRIGGERS prc-augmentation-densité-des
      arbres;
      ON TERMINATION ABORTS prc-augmentation-densité-des
      arbres;
      . l'accroissement du nombre d'arbres provoque une aug-
        mentation de la densité des arbres.
        La densité des arbres cesse d'augmenter lorsque le
        nombre d'arbres cesse d'augmenter.
* DEFINE PROCESS prc-augmentation-densité-d'arbres;
      ON INCEPTION TRIGGERS prc-diminution-conditions-vitales;
      . lorsque la densité des arbres augmente, humidité,
        lumière et espace vital diminuent.
* DEFINE PROCESS prc-diminution-conditions-vitales;
      ON INCEPTION TRIGGERS prc-diminution-nouvelles-pousses;
      ABORTED BY TERMINATION OF prc-augmentation-densité-
      d'arbres;
      . lorsque les conditions vitales diminuent, les
```

graines ont moins de chance de germer.
Les conditions vitales cessent de diminuer lorsque la densité des arbres cesse de s'accroître.

* DEFINE PROCESS prc-diminution-nouvelles-pousses;
ON INCEPTION TRIGGERS prc-diminution-nombre-d'arbres;
. le fait que les graines ont des difficultés à germer a pour conséquence que le nombre d'arbres diminue.

* DEFINE PROCESS prc-diminution-nombre-d'arbres;
ON INCEPTION TRIGGERS prc-diminution-densité-des-arbres;
ON TERMINATION ABORTS prc-diminution-densité-des-arbres;
. la diminution du nombre d'arbres provoque une diminution de la densité des arbres.
La densité des arbres cesse de diminuer lorsque le nombre d'arbres cesse de diminuer.

Voyons un deuxième exemple, beaucoup plus direct, proposé par Jay W. Forrester (17) : il s'agit des boucles de population.

Si la population augmente, le taux B.R (Birth Rate c'est-à-dire le nombre d'individus nés dans le système par année) augmente.
L'accroissement de ce taux provoque un accroissement de population.

Si la population augmente, le taux D.C (Death Rate c'est-à-dire le nombre d'individus qui meurent dans le système par année) augmente. L'accroissement de ce taux provoque une diminution de population.

Si la population diminue, le taux BR diminue, ce qui provoque une diminution de population.

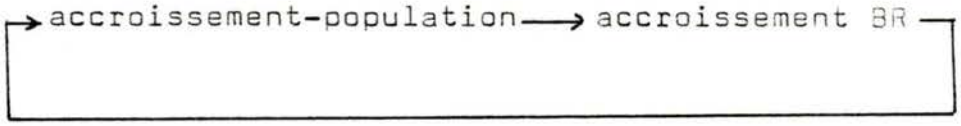
Si la population diminue, le taux DR diminue, ce qui provoque un accroissement de population.

---

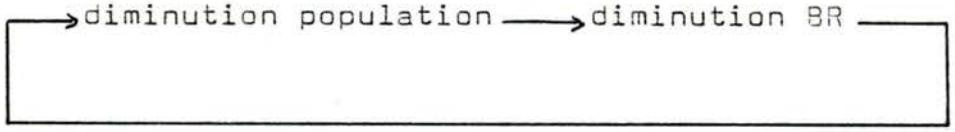
(17) FORRESTER W. Jay : "World Dynamics" Wright-Allen-Press, 1971, pp. 20-23.

Les boucles de la population peuvent se traduire ainsi dans la version NAMUR :

```
* DEFINE PROCESS prc-accroissement-population;
     ON INCEPTION TRIGGERS prc-accroissement BR;
     ON INCEPTION TRIGGERS prc-accroissement DR;
     . si la population augmente, les taux BR et DR s'accroissent également.
* DEFINE PROCESS prc-accroissement BR;
     ON INCEPTION TRIGGERS prc-accroissement-population;
     . si le taux BR augmente, la population augmente.
     Il s'agit donc d'une boucle directe :
```

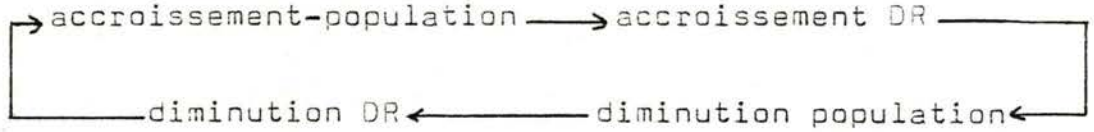



```
* DEFINE PROCESS prc-accroissement DR;
     ON INCEPTION TRIGGERS prc-diminution-population;
     . si le taux DR augmente, la population diminue.
* DEFINE PROCESS prc-diminution-population;
     ON INCEPTION TRIGGERS prc-diminution BR;
     ON INCEPTION TRIGGERS prc-diminution DR;
     . si la population diminue, les taux BR et DR diminuent également.
* DEFINE PROCESS prc-diminution BR;
     ON INCEPTION TRIGGERS prc-diminution-population;
     . si le taux BR diminue, la population diminue.
     Il s'agit d'une deuxième boucle directe :
```




```
* DEFINE PROCESS prc-diminution DR;
     ON INCEPTION TRIGGERS prc-accroissement-population;
     . si le taux DR diminue, la population augmente.
```

Il s'agit maintenant d'une boucle indirecte :



d) Comparaison entre les différentes versions - Exemples

Comparons maintenant l'expression de la dynamique dans la version NAMUR et dans les versions 4.2 et 5.1. Dégageons-en les différences essentielles sur base d'exemples.

La sous-structure des traitements de la version NAMUR présente deux avantages qui font que, pour la description d'un problème assez dynamique, elle est plus attrayante que les versions 4.2 et 5.1.

Ces avantages se résument ainsi :

- la standardisation des clauses facilite l'utilisation du langage
- l'écriture du problème est brève et ne nécessite pas de longs artifices.

Nous avons insisté sur l'aspect standardisation dans la classification des événements.

Examinons un peu plus en détails la formalisation dans la version NAMUR.

1) Formalisation

Reprenons l'exemple du fonctionnement d'un service d'urgence (p. 53) que nous avons décrit à l'aide de la version 4.2 puis de la version 5.1. En utilisant la version NAMUR, l'expression du problème est la suivante :

* DEFINE EVENT ev-déclenchement-d'un-incendie;
TRIGGERS prc-appel-900;

. le formalisme est identique à ceux des versions 4.2 et 5.1 (mis à part la convention du mot "DEFINE")

* DEFINE PROCESS prc-appel-900;
ON INCEPTION TRIGGERS prc-extinction-incendie;

. la formalisation est plus courte que dans les versions 4.2 et 5.1 du fait qu'on ne décrit pas d'événement intermédiaire.

* DEFINE PROCESS prc-extinction-incendie;
ON INCEPTION TRIGGERS prc-hospitalisation FOR EACH nombre-de-blessés IF cond-existence-de-blessés IS TRUE;
ON TERMINATION GENERATES msg-fin-alerte;

. la condition fait entièrement partie de l'enchaînement dynamique. Tous les liens entre les différentes actions sont faits et la présentation est extrêmement synthétique par rapport à celle des versions 4.2 et 5.1.

Nous voyons dans cet exemple combien un utilisateur peut apprécier la version NAMUR pour la description de problèmes complexes :

. la présentation synthétique est très claire
. le fait de pouvoir décrire des actions conditionnelles représente un réel avantage.

2) Problèmes d'ordonnancement

Nous avons émis une critique vis-à-vis de la version 5.1 qui oblige l'utilisateur à passer par de nombreux intermédiaires pour décrire un problème d'ordonnancement. Voyons donc si la version NAMUR est adaptée à la description de tels problèmes, ces problèmes étant susceptibles d'être soumis à une simulation.

Reprenons les deux étapes du problème déjà présenté p. 58 (examen de la version 5.1) :

étape 1 : il faut obtenir un permis d'exploitation pour pouvoir commencer certains travaux

étape 2 : un de ces travaux est la construction d'une route; cette construction ne pourra commencer que cinq semaines après l'otention du permis d'exploitation.

Voici comment la version NAMUR décrit ce problème :

```
* DEFINE MESSAGE msg-obtention-permis-d'exploitation;
    ON GENERATION CONTRIBUTES TO syn-construction-route
       IF cond-obtention IS TRUE
       AFTER int-5-semaines DELAY;
```

. lorsque la demande de permis d'exploitation est renvoyée (GENERATION), si l'autorisation est accordée (IF cond IS TRUE), on pourra commencer la construction de la route cinq semaines plus tard (AFTER ... DELAY)

```
* DEFINE SYNCHRONIZATION-POINT syn-construction-route;
    TRIGGERS prc-construction-route;
```

. une fois réalisé, c'est-à-dire une fois le délai de 5 semaines écoulé, le point de synchronisation va déclencher le processus de construction de la route.

Deux sections et deux clauses suffisent pour décrire les deux étapes du problème. Le passage par le point de synchronisation est plus simple que tous les intermédiaires nécessités dans la version 5.1 pour décrire le même problème.

Le point de synchronisation est indispensable pour la description des problèmes d'ordonnancement; il ne complique cependant pas le problème. Cette description est, en effet, aisément transposable en un graphe PERT ou MPM.

I.4.3.3. Sous-structure des ressources - version NAMUR

Nous avons vu que la sous-structure des ressources de la version NAMUR s'articule autour des quatre concepts suivants :

PROCESS - PROCESSOR - CONSUMABLE-RESOURCE - REUSABLE-RESOURCE

Nous avons analysé la syntaxe attachée au concept principal, la syntaxe du "PROCESSOR". La sémantique attachée aux différentes clauses est loin d'être évidente, c'est pourquoi nous nous proposons de l'expliquer à partir d'un exemple.

Voici l'énoncé de cet exemple :

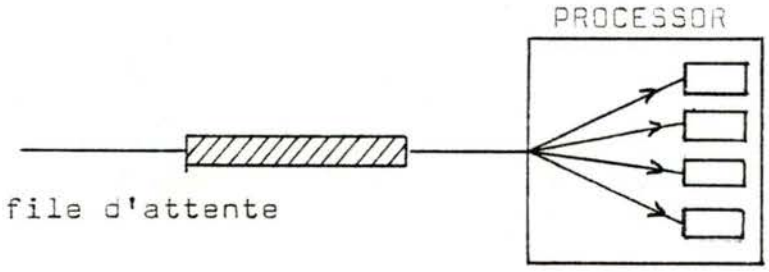
Soit, dans une firme commerciale, le poste chargé de contrôler la validité des bons de commande reçus. Il est composé de quatre opératrices contrôlant les différentes parties de la commande : identification du client, lignes de commande, prix. Elles enregistrent les commandes valides.

Cet énoncé est traduit de la façon suivante dans la version NAMUR :

* DEFINE PROCESSOR proc-poste-contrôle-enregistrement;
  - le poste de contrôle et d'enregistrement est chargé d'effectuer divers traitements; il est donc défini comme un "PROCESSOR"
* UNIT-OF-MEASURE IS interval-minute;
  - l'activité d'un "PROCESSOR" est mesurée par rapport à une unité de temps que nous choisissons ici égale à la minute.
* CAPACITY IS 4
  - le niveau d'activité du processus est égal à 4. Ceci signifie que si le processus est actif durant 1 minute, le travail qu'il accomplit durant cette minute est équivalent à 4 minutes de travail. Ceci correspond à la situation où un processus (ici, le processeur-poste-contrôle-enregistrement) est constitué de 4 postes de travail (dans notre exemple, chaque opératrice représente un poste de travail).

* SHARABLE AMONG 4;
  . cette clause signifie qu'il y a 4 points d'entrée dans le processeur; ceci équivaut à dire que le poste de travail peut exécuter au maximum quatre processus simultanément.
  Si un cinquième processus veut saisir le processeur, il sera mis en file d'attente (graphique 19) jusqu'à ce qu'un processus ait quitté le poste.

Graphique 19 : Points d'entrée dans le processeur



La valeur associée au SHARABLE n'est pas toujours identique à celle associée à la CAPACITY :

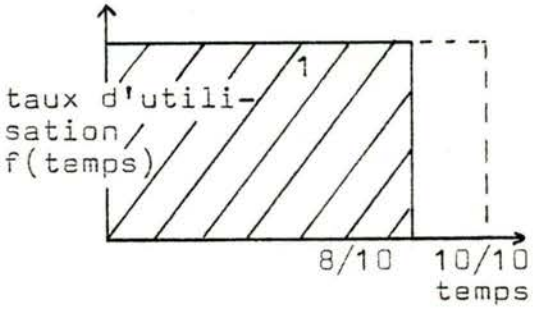
- elle peut être inférieure : ex : SHARABLE AMONG 2 signifie que deux processus seulement peuvent se trouver dans le processeur à un moment donné
- elle peut être supérieure : ex : SHARABLE AMONG 5 signifie que cinq processus, au maximum, peuvent être admis et traités simultanément par le processeur.

* ACTIVE FOLLOWING calendrier-hebdomadaire;
  . la section CALENDAR définira calendrier-hebdomadaire, par ailleurs. Ce calendrier est constitué des intervalles de temps pendant lesquels le processeur est actif.
* USES rr-operatrice AT RATE 1-unité-de-mesure-opératrice TO PERFORM prc-contrôle, prc-enregistrement;
  . cette clause signifie que la ressource est utilisée à 100 % de sa capacité pendant le temps durant lequel

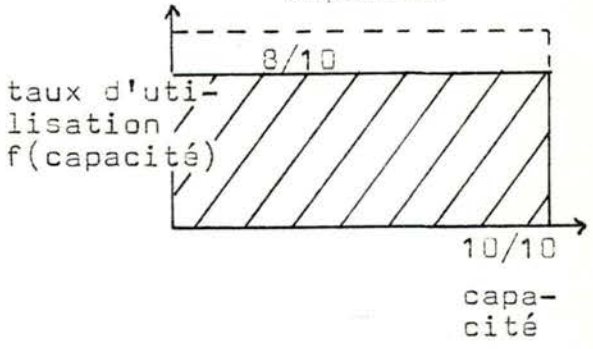
elle est requise par le processeur. La valeur associée à la clause RATE indique quelle portion de la capacité du processeur est utilisée lors du traitement d'un processus.

- Le taux (rate) peut être <u>égal</u> à l'unité comme dans l'exemple
- il peut également être <u>inférieur</u> à l'unité : <u>ex</u> :
  AT RATE 0.8;
  ceci signifie que le processeur va utiliser entièrement la ressource pendant 8/10 du temps qu'il la réclame (graphique 20) ou utilise les 8/10 de sa capacité pendant tout le temps qu'il la réclame (graphique 21)

<u>Graphique 20</u> : taux d'utilisation fonction du temps



<u>Graphique 21</u> : taux d'utilisation fonction de la capacité



L'optique (graphique 20 ou graphique 21) qui sera choisie lors de la simulation influencera la disponibilité des ressources. Dans le cas du graphique 20, la ressource sera utilisée pendant huit minutes sur dix et pourra encore être utilisée pendant les deux minutes restantes. Ceci est valable dans le cas où on n'a pas pris l'hypothèse d'allocation de ressources <u>pour toute la durée d'exécution</u> du processus quel que soit le temps effectif durant lequel la ressource est utilisée par le processeur.

Dans le cas du graphique 21, la ressource sera toujours sous-utilisée si on prend l'hypothèse que l'allocation de ressources

se fait en nombres entiers.

Les possibilités au niveau de la simulation dépendent des choix qui seront faits tant au point de vue de l'interprétation qu'au point de vue des hypothèses de simulation.

. Le taux peut également être supérieur à l'unité : ex :
AT RATE 1.2;
ceci signifie que pour accomplir un processus, une opératrice (utilisée AT RATE 1-unité-de-mesure-opératrice) aura besoin de l'assistance de 2/10 de la capacité de travail d'une autre opératrice (utilisée AT RATE 0.2-unité-de-mesure-opératrice) ou de toute sa capacité pendant 2/10e du temps de traitement.

* PERFORMS prc-contrôle DURING 2 1/2 minutes; (1)
PERFORMS prc-enregistrement DURING 1/2 minute;

. ces clauses indiquent le temps pendant lequel le processus requiert l'usage d'un processeur à un taux équivalent à $\frac{\text{CAPACITY}}{\text{SHARABLE}}$ pour être exécuté. Si on a
CAPACITY = 4 et SHARABLE = 4, la forme (1) est donc équivalente à :
PERFORMS prc-contrôle DURING 2 1/2 minutes AT RATE 1;
"AT RATE 1" de sa capacité, signifie bien que si l'on a 4 processus, la capacité de travail 4 x 1 = 4 sera atteinte et on ne pourra plus accepter qu'un processus rentre dans le processeur avant qu'un autre processus en soit sorti.
Cette diminution de capacité (CAPACITY = 4 x 1 - 1 x 1)
1 processus
AT RATE 1
est valable durant 2 1/2 minutes pour le prc-contrôle.
Elle est valable durant 1/2 minute pour le prc-enregistrement.

La description de la ressource "opératrice" employée par le processeur se fait de la façon suivante :

* DEFINE REUSABLE-RESOURCE rr-operatrice;
  . la ressource utilisée est le personnel "opératrice".
* UNIT-OF-MEASURE IS unité-de-mesure-opératrice;
  . on pourrait définir ici comme unité de mesure le temps ou une unité "opératrice" permettant par la suite d'indiquer le nombre d'opératrices employées ou disponibles. Pour d'autres ressources, d'autres unités peuvent être choisies :
    le litre d'eau, la tonne de métal, etc.
* AVAILABILITY IS 5;
  . cette clause indique la disponibilité de la ressource.

Cette description permet de savoir, à tout moment, quelle est la capacité utilisée, quelle est la capacité restant disponible, quel est le nombre de processus se déroulant dans le processeur, etc.
On pourra également organiser le déroulement du processus étant donné la disponibilité des ressources. Par exemple, si un processus A requiert l'utilisation de 4 unités-de-mesure-opératrice, il ne peut être exécuté en même temps qu'un processus demandant l'utilisation de 3 unités-de-mesure-opératrice puisque la disponibilité de la ressource "opératrice" a été définie égale à 5.

Toutes les informations que l'on peut connaître dynamiquement par l'intermédiaire de la description de la sous-structure des ressources sont très intéressantes pour l'utilisateur qui doit dimensionner un système qu'il a conçu.

La version NAMUR s'est donc efforcée de développer les spécifications dynamiques d'après l'utilisation qu'elle veut en faire par la suite.

La standardisation des clauses dynamiques de la

sous-structure des traitements la rend aisée à manipuler pour la description de problèmes dynamiques.

La dynamique de la sous-structure des ressources met à la disposition de l'utilisateur des informations précieuses quant à la disponibilité des éléments du système. On peut cependant regretter que les clauses attachées à cet aspect de la description ne soient pas plus évidentes du point de vue de leur utilisation.

Conclusion de la première partie

Dans cette Première Partie, nous avons analysé, comparé et critiqué le langage de description (PSL) mis à la disposition d'un utilisateur par les versions 4.2, 5.1 et NAMUR en insistant sur les aspects dynamiques.

Au terme de cette analyse, nous pouvons conclure que chaque version répond aux objectifs qu'elle s'est fixés. Un utilisateur qui doit choisir un langage PSL doit donc le faire en fonction de ses objectifs.

Chacune des versions permet de décrire un système tant au point de vue statique qu'au point de vue dynamique.

- La version 4.2 développe surtout les spécifications permettant la description des aspects statiques. Le vocabulaire utilisé regroupe des concepts assez étendus. L'utilisateur qui voudrait décrire des caractéristiques plus spécifiques devrait définir ces caractéristiques lui-même, par exemple sous forme d'attributs.
- La version 5.1 développe, en plus des aspects statiques de la version 4.2, les spécifications permettant la description détaillée des aspects dynamiques d'un système.
- La version NAMUR est toujours en cours de développement. Elle insiste sur la description des aspects dynamiques du système. Elle détaille les spécifications de la dynamique de la sous-structure des traitements ainsi que celle de la

sous-structure des ressources du fait qu'elle a pour objectif de simuler le système à partir de sa description.

Ces trois versions se sont ainsi fixées leurs objectifs respectifs et ont donc développé chacune leur langage de description (PSL) et d'analyse (PSA).

Elles présentent cependant le désavantage de ne pas être accompagnées d'une documentation à la portée de l'utilisateur.

DEUXIEME PARTIE

R E A L I S A T I O N D E R A P P O R T S -

A P P L I C A T I O N A U C A S D U R A P P O R T

DYNAMIC-INTERACTION

Dans la Première Partie, nous avons analysé la syntaxe de différentes versions PSL du projet Isdos.

La deuxième partie a pour but d'analyser la conception d'un rapport (PSA) destiné à la version NAMUR. Cette deuxième partie se décompose en trois chapitres :

Dans un premier chapitre, nous présentons le contexte dans lequel est programmé et implémenté le rapport, c'est-à-dire le META-SYSTEM et le GENERALIZED-ANALYZER.

La deuxième chapitre présente l'environnement de ce rapport au point de vue programmation.

Le troisième chapitre développe la conception même du rapport.

CHAPITRE II.1 - PRESENTATION DU META-SYSTEM ET DU GENERALIZED-ANALYZER

Avant de présenter un rapport spécifique, nous allons définir sur quelle base nous pouvons concevoir des rapports adaptés à la version NAMUR d'Isdos.

Ce chapitre a pour but :

- de définir les outils employés pour la construction du langage PSL (version NAMUR) dont nous avons analysé la syntaxe dans la première partie.
- de montrer la structure et le fonctionnement de ces outils.
- d'examiner la façon dont ils ont été appliqués à la version NAMUR.

Pour construire le langage PSL, deux outils principaux, liés l'un à l'autre, sont utilisés :

. le META-SYSTEM
et . le GENERALIZED-ANALYZER.

II.1.1. Le META-SYSTEM (18)

Le META-SYSTEM est un système défini par les trois éléments qui le composent : - le META-langage
- le META-generator
- la META-data-base.

Ces trois éléments apparaissent sur le graphique 22 (p. 103).

Le rôle du META-langage est essentiel; en effet, il est assez gênant d'imposer à l'utilisateur un langage rigide défini par des informaticiens. C'est pourquoi le projet Isdos a développé le META-langage, soit un langage permettant à l'utilisateur de définir le vocabulaire et la grammaire qu'il veut employer en vue de décrire son système (1).

Le langage META rend donc possible l'auto-extensibilité du langage PSL puisque tout utilisateur, employant le PSL du langage META peut développer son propre PSL.

Voyons comment concrètement le META-langage permet à l'utilisateur de définir son propre langage (appelé Target Language).
Le META-langage doit être employé par l'utilisateur pour définir les "OBJECT-TYPE" et les "ASSOCIATION" qu'il désire utiliser dans son PSL.

La structure d'une section META est la suivante :

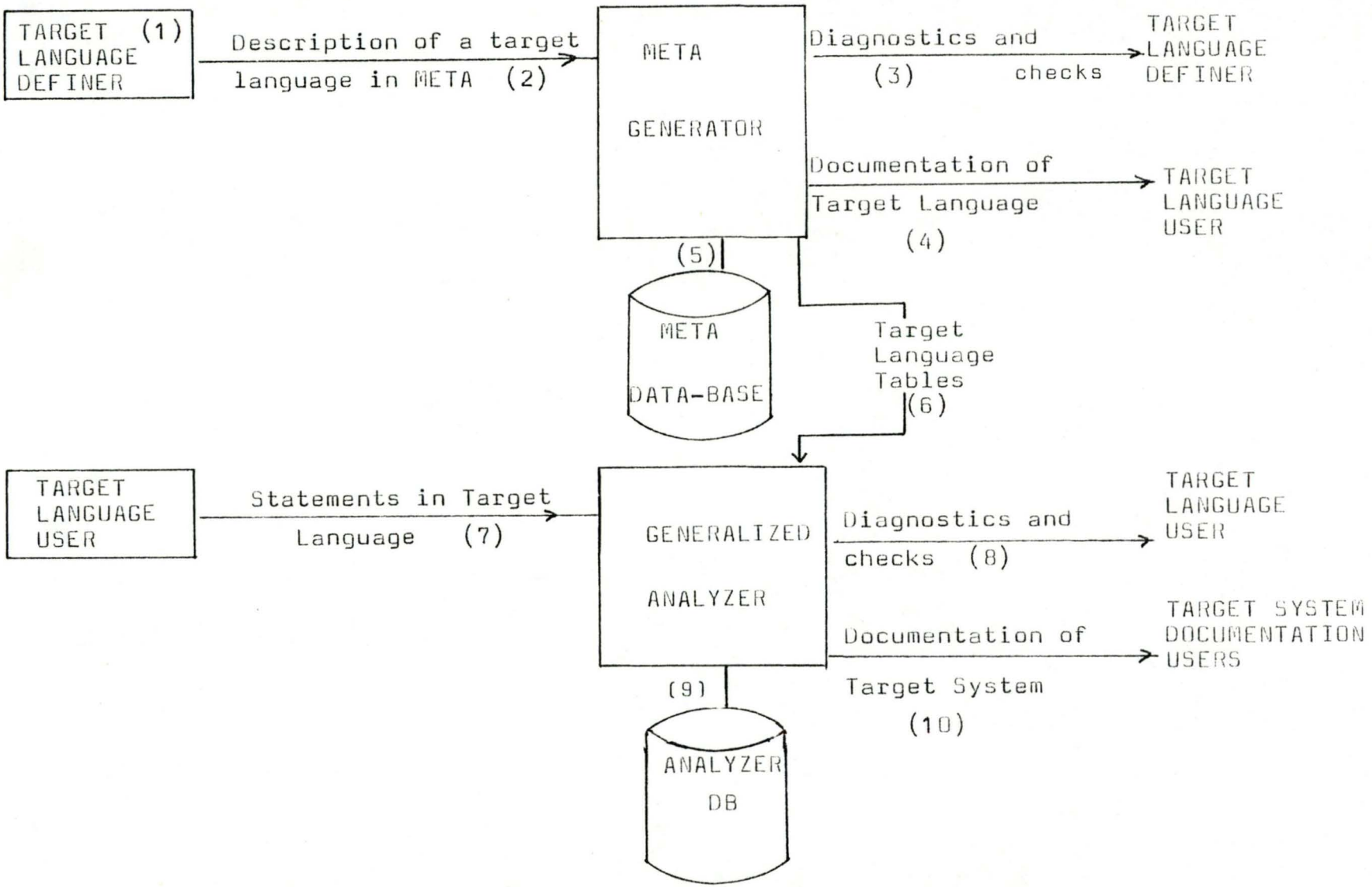
1. object type OBJECT OBJECT-TYPE (cet OBJECT TYPE est un user-name)

ex OBJECT PROCESS
ou OBJECT PROCESSUS (si l'utilisateur est francophone)

---

(18) D. TEICHROEW : "Isdos Meta System Memorandum META-1", Isdos Project. May 1977, revised February 1978.

Graphique 22 : Structure du Meta-System et du Generalized-Analyzer



2. association RELATION ASSOCIATION-NAME (défini par l'utilisateur)

ex RELATION consists-of-relation

. différentes clauses permettent à l'utilisateur de spécifier les "OBJECT-TYPE" concernés par cette "RELATION" et le formalisme selon lequel s'exprime cette "RELATION".

Une fois que l'utilisateur a défini un Target Language à l'aide du meta-langage, il va introduire cette définition dans le META-GENERATOR ((2) dans le graphique 22 p. 103.)

Le meta-générateur analyse la syntaxe du meta-langage et fournit à l'utilisateur

. les messages d'erreur (3)
. une documentation concernant le Target Language (4)

Il est chargé d'introduire les informations qu'il reçoit dans une base de données appelée META-DATA-BASE (5) et de traduire ces informations sous forme de tables appelées Target Language Tables (6).

II.1.2. Le GENERALIZED-ANALYZER (GA)

Pour l'utilisateur, il s'agit d'abord de définir les concepts dont est constitué le langage qu'il désire employer. Ensuite seulement, il pourra utiliser le langage spécifié. Cette deuxième étape passe par l'intermédiaire du GENERALIZED-ANALYZER (GA). Le schéma de cette deuxième étape apparaît également sur le graphique 22 p. 103.

L'utilisateur décrit son problème à l'aide du langage qu'il a élaboré (7). Le GA :

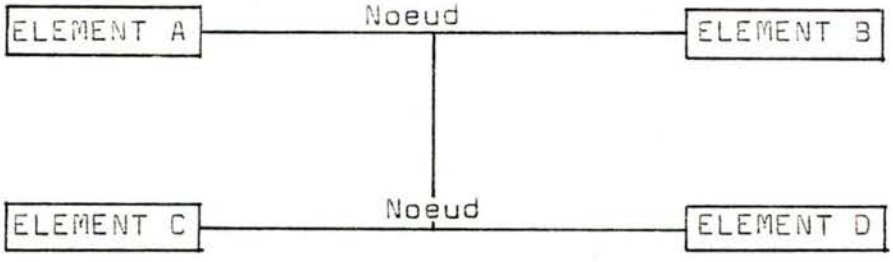
. examine cette description par rapport aux concepts définis par l'utilisateur, concepts qui sont mémorisés sous forme de tables auxquelles le GA a accès (6).

Notons que, dans certaines implémentations, c'est la META Database qui sert de tables pour le GA

- . fournit à l'utilisateur les diagnostics d'erreurs (8)
- . introduit la description dans une base de données appelée ANALYZER-DATA-BASE (9)

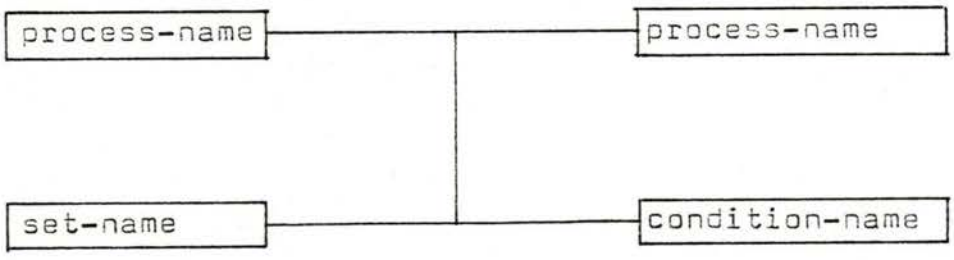
L'information ainsi introduite dans la base de données peut être retrouvée et présentée à l'utilisateur sous forme de rapports (10).

Signalons qu'il existe déjà deux versions du META-SYSTEM et donc du GA.
La première version du META autorise une relation à associer au maximum quatre types d'objets. C'est cette version que le langage défini à NAMUR emploie.
Dans une META-data-base construite par la première version du META, les relations sont enregistrées de la façon suivante :



Prenons, par exemple, la relation TRIGGERS qui associe PROCESS-PROCESS-CONDITION-SET et voyons comment elle est enregistrée dans la META-data-base.
La syntaxe de cette relation est

* DEFINE PROCESS process-name;
ON INC TRIGGERS process-name FOR EACH set-name IF
condition-name IS TRUE
FALSE



La deuxième version du META et du GA autorisent des associations entre dix éléments au maximum. La structure de la META-data-base peut être représentée par un polyèdre à dix sommets.

La version NAMUR est construite sur base de la première version (seule disponible au début de l'élaboration de la version NAMUR). L'usage de la deuxième version lui aurait ouvert d'autres possibilités du fait qu'elle n'aurait pas été limitée par la règle des quatre objets.

CHAPITRE II.2 - CARACTERISTIQUES DE LA PROGRAMMATION DES LOGICIELS ISDOS

L'objet de cette deuxième partie est d'analyser la conception d'un rapport tel qu'il puisse se baser sur des informations décrites à l'aide de la version NAMUR.

Le premier chapitre de cette partie a présenté les deux outils qui ont permis d'élaborer la version NAMUR et a introduit un des caractères principaux du développement du projet Isdos : l'auto-extensibilité du langage PSL.*

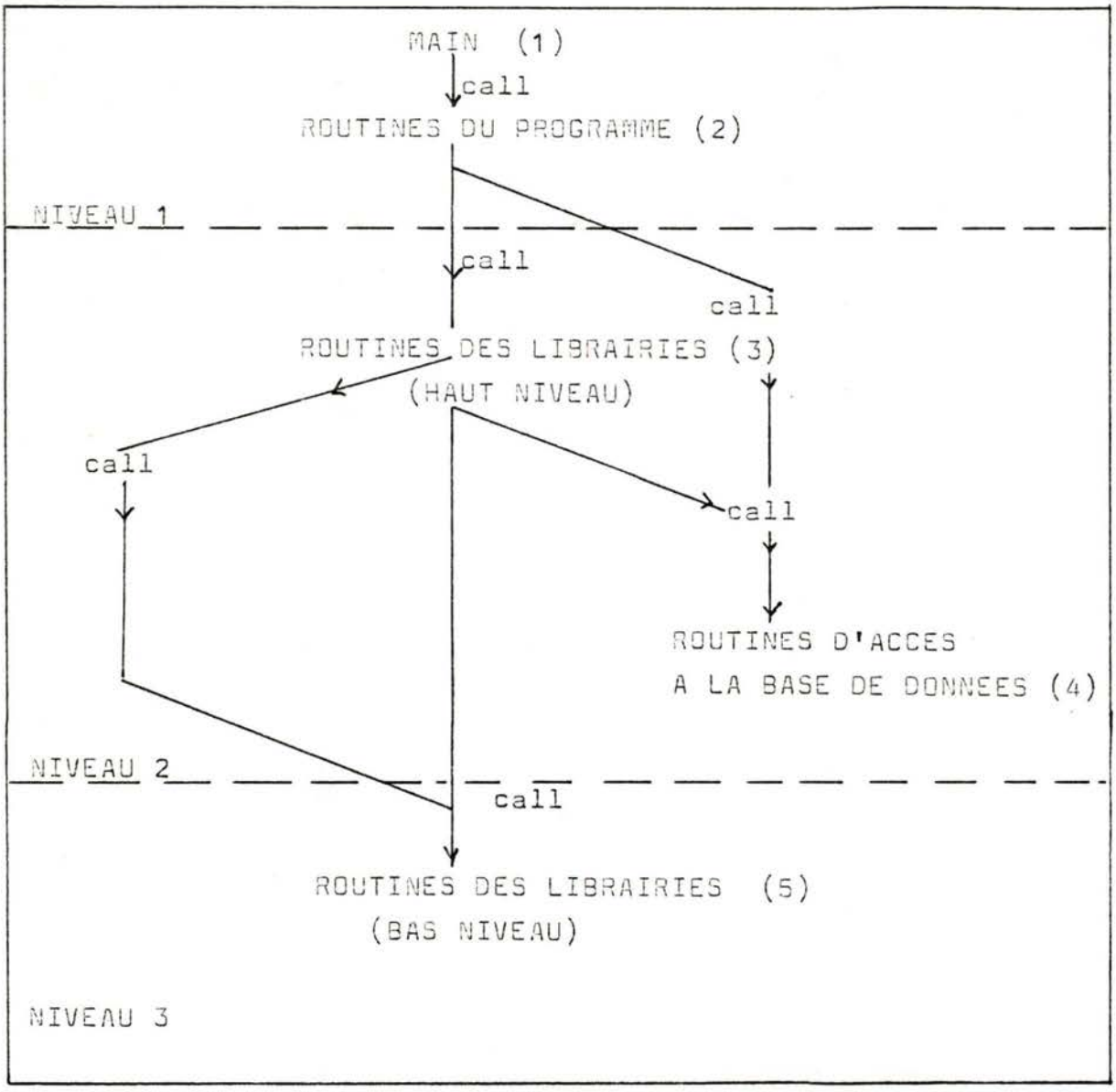
Ce second chapitre a pour but de mettre en évidence deux autres points qui caractérisent la programmation dans Isdos : la modularité et la portabilité.

Nous analysons tout d'abord la modularité des programmes qui jouent le rôle d'interface entre l'utilisateur et l'information enregistrée dans la base de données.
Nous analysons ensuite le rôle du langage de programmation utilisé par rapport au caractère de portabilité des logiciels Isdos.

II.2.1. Modularité des programmes

Tout programme qui joue le rôle d'interface entre l'utilisateur et l'information enregistrée dans la base de données peut être structuré selon trois niveaux, schématisés sur le graphique 23.

Graphique 23 : Structuration des programmes



NIVEAU 1 : Le niveau 1 regroupe les routines qui appartiennent au programme, c'est-à-dire la routine principale et les routines appelées par cette routine principale.

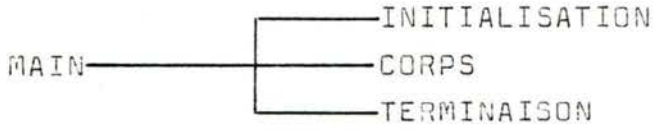
a) routine principale ((1) dans le graphique 23) : (19)

Tout programme comprend une routine principale, qui, généralement, dans le projet Isdos, reçoit le nom de Mainxxxx. où xxxx est remplacé par des abréviations significatives (ex abréviation du nom de rapport que le programme a pour but d'exécuter).

Le MAIN est structuré selon le graphique 23 bis. Il appelle d'autres routines du programme et au moins :

- une routine d'initialisation
- une ou plusieurs routines constituant le corps du programme
- une routine de terminaison.

Graphique 23 bis : Structuration du MAIN



b) routines du programme ((2) dans le graphique 23, page 108)

Le programme a une structure interne qui lui est propre. Généralement, il est subdivisé de façon à ce que chaque routine remplisse une fonction particulière. Cette structuration permet de mieux localiser les changements à apporter lorsqu'une modification est souhaitée concernant

---

(19) Isdos Project : "Procedures and Conventions for Developping Users' Software for PSA", Isdos réf. WP 164, August 1976.

l'information traitée ou la présentation de cette information.

Lors de notre stage à l'université du Michigan, nous avons eu l'occasion de modifier le Query System (programme parallèle au NAME-SELECTION). Nous pensons qu'il représente un bon exemple de structuration de programmes, c'est pourquoi nous reproduisons une partie de sa structure en annexe (cfr. ANNEXE 1).

Les routines (principales et autres) du niveau 1 peuvent appeler les routines des niveaux 2 et 3.

NIVEAU 2 : Le niveau 2 regroupe les routines évoluées c'est-à-dire les routines des librairies de haut niveau et les routines permettant au programme d'accéder aux informations enregistrées dans la base de données.

a) routines de haut niveau ((3) dans le graphique 23)

Les routines de haut niveau sont regroupées dans des librairies selon la fonction qu'elles remplissent. Elles ne sont pas incluses dans le programme proprement dit car la fonction qu'elles remplissent est commune à plusieurs programmes.

Citons par exemple :

DRAWLIB : librairie comprenant les routines chargées d'effectuer les dessins dans les rapports présentés sous forme graphique

PARMLIB : librairie s'occupant de la gestion des paramètres propres à chaque rapport

ROLIB : librairie s'occupant de la gestion des buffers

SMLIB : librairie s'occupant de la gestion des matrices.

b) routines d'accès à la base de données ((4) dans le graphique 23 p. 108)

Les rapports sont des interfaces entre l'utilisateur et les informations enregistrées dans la base de données. Les programmes doivent donc pouvoir accéder à la base de données.

Deux cas sont à distinguer : . le cas du PSA
. le cas du Generalized-Analyzer

Le PSA accède à la base de données uniquement via le chemin (1) du graphique 24 p. 112, c'est-à-dire, via les routines de ADBMS (A DATA BASE MANAGEMENT SYSTEM). ADBMS se base sur les spécifications d'un rapport Codosyl (20). Il a subi des restrictions et des extensions de façon (21)

- à satisfaire les exigences de PSA
- à ne pas occuper une trop grande place mémoire
- à pouvoir utiliser Fortran ou Cobol comme langage-hôte.

ADBMS a donc été développé pour PSA mais il est possible de l'employer comme système indépendant.

Le Generalized-Analyzer a deux solutions possibles pour accéder à la base de données où sont enregistrées les informations :

- soit il emploie les ordres ADBMS (chemin (1) dans le graphique 24 p. 112) et utilise donc ce système de gestion de base de données comme le font les rapports PSA.

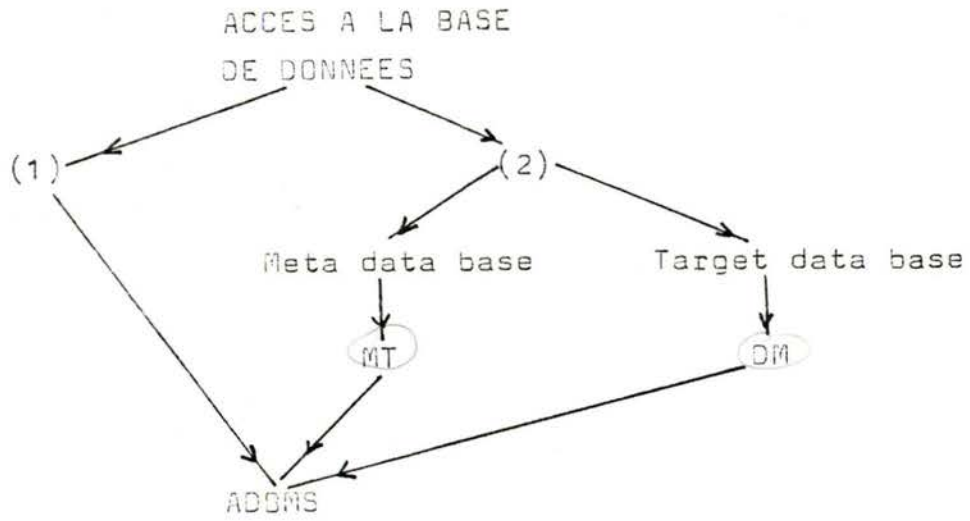
---

(20) "Codasyl Cobol Data Base Facility Proposal", Department of Supply and Services, Ottawa, Ontario, Canada, March 1973.

(21) E.A. HERSHEY : "A Data Base Management System (ADBMS) Based on DBTG 71", Revised for D3.0 by G. Fleming, Isdos Project, réf WP 197, November 1977.

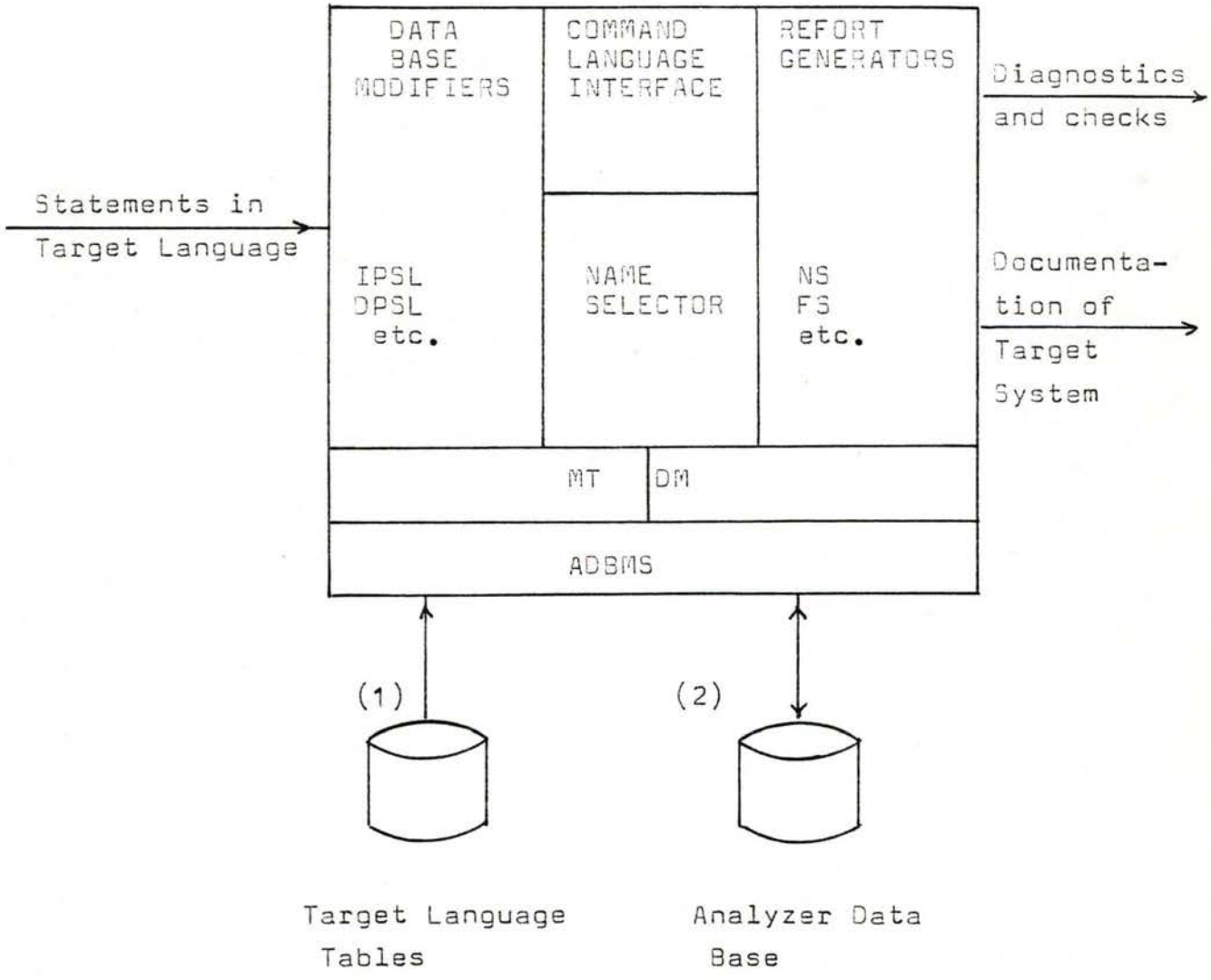
. soit il recourt aux systèmes appelés DM et MT qui eux-mêmes font appel à ADBMS (chemin (2) du graphique 24).

Graphique 24 : Chemins d'accès à la base de données



Pour bien comprendre le rôle des systèmes DM et MT, analysons le graphe de la structure du Generalyzed-Analyzer (graphique 25 p. 113) qui, comme nous l'avons vu au chapitre précédent, permet à l'utilisateur d'employer le langage qu'il a défini.

Graphique 25 : Structure du GA (22)



Nous voyons que le problème d'accès à l'information est double pour le GA, puisqu'il doit accéder aux tables du langage (rappel : ces tables sont construites par le meta-system) (1) et à la base de données de l'Analyzer (2). L'accès à la base de données lui permet de retrouver une information particulière et son type (2) et l'accès aux tables lui permet de savoir à quel type d'objet ce type d'information appartient (1).

---

(22) META System : op. cit. en (19) p. 109.

Le MT système s'occupe de tous les accès aux tables du langage.
Le DM système s'occupe de tous les accès à la base de données de l'Analyzer.

NIVEAU 3 : Nous venons de voir que les NIVEAUX 1 et 2 peuvent appeler des routines du NIVEAU 3. Le NIVEAU 3 regroupe les routines dites de "bas niveau" c'est-à-dire des routines employées très fréquemment et qui ne doivent pas être modifiées lors d'adaptation des programmes de niveau supérieur.

Ces routines sont classées dans cinq librairies :

-SLIB : routines dépendant du système
- BTLIB : routines indépendantes de l'Analyzer et qui ne nécessitent pas de block-data pour l'initialisation
- DBLIB : système de contrôle de base de données.
  DBLIB fait appel à BTLIB et à SLIB.
- FLIB : effectue les fonctions courantes demandées par le système PSA
  ex impression des erreurs
  FLIB fait appel à FLIB, BTLIB et SLIB
- CLIB : teste les paramètres et initialise les switches du programme.

II.2.2. Portabilité et langage de programmation

Toutes les routines que nous venons d'étudier concernent les programmes mis à la disposition de l'utilisateur pour gérer sa base de données.
Ces programmes sont écrits à l'aide d'un langage proche du fortran, appelé EXTENDED FORTRAN (EF) (23).

---

(23) E.A. HERSHEY, Y.YAMAMOTO, D. MIETLA : "EF Format", Isdos Project, réf TM 205, February 1979.

Voyons quelles sont les particularités de ce langage :

1. Opérateurs relationnels et booléens

En Extended Fortran, les opérateurs relationnels et booléens sont exprimés sous forme de signes.
Par exemple : ".LT." est écrit "<"
".EQ." est écrit "==".
Ces signes doivent être précédés et suivis de " ("ou", "ou")" ou blanc.

2. Strings

La manipulation de strings est simplifiée en Extended Fortran. Il y a quatre manières d'exprimer les strings constants. Le tableau 2 (p. 116) nous montre la manipulation de strings en Extended Fortran, la traduction de ces strings en fonction et la procédure de traduction.

L'Extended Fortran prévoit, de plus, une déclaration de variables "caractère". Cette déclaration se fait de la façon suivante :

*CHAR,name(char-length)

Cette méthode permet une déclaration aisée de buffers sur lesquels travaillent, par exemple, les routines de RQLIB.

Tableau 2 : Manipulation de strings

Format EF	Format traduit	Procédure
#A'string'	'string', ptr, length	Le processus EF trouve automatiquement le pointeur (ptr) et calcule la longueur (length)
#S'string'	length, 'string'	Le processus calcule lui-même la longueur (length)
#M'string'	'string.'	Le processus ajoute le point de ponctuation
#L'string'	length	Le processus calcule la longueur mais n'indique pas le string

3. Initialisation

Trois formes d'initialisation de variables sont prévues :

a) Initialisation de variables "character"

L'instruction *DATA permet d'initialiser des variables de type "character". Sa formalisation est la suivante :

```
*DATA,name
  'string' 'string'

  'string'

*DEND
```

Les variables doivent être déclarées à l'aide de l'instruction

*CHAR, avant leur initialisation.

ex *CHAR,CHSTNG(30) : définition du buffer comprenant les caractères apparaissant dans la matrice

```
*DATA,CHSTNG
    ' T123U123I123R123A123G123C123X'
*DEND
```

Le buffer est initialisé à l'aide de l'instruction *DATA. Il comprend les lettres T-U-I-R-A-G-C-X intervenant à l'intersection des lignes et des colonnes de la matrice et les chiffres 123 qui servent aux routines d'impression.

La traduction fortran de cet exemple est la suivante :

```
      INTEGER CHSTNG(6)
      DATA CHSTNG(1)/5H T123/,CHSTNG(2)/5HU123I/,
      CHSTNG(3)/5H123R1/,CHSTNG(4)/5H23A12/,CHSTNG(5)/
      5H3G123/,CHSTNG(6)/5HC123X.
```

b) initialisation de variables "character" avec répétition

L'instruction *DATA permet d'initialiser des variables de type "character" à l'aide d'un string répété n fois, sans avoir à écrire n fois ce string.

Par exemple, on veut initialiser la variable plus à 10 fois le caractère '+'

```
*CHAR,PLUS(10)
*DATA,PLUS10'+'
```

La traduction fortran est la suivante :

```
      INTEGER PLUS(10)
      DATA PLUS(1)/5H+++++/,PLUS(2)/5H+++++.
```

c) "initialisation" de constantes

L'instruction *DEFN permet à l'utilisateur d'assigner des valeurs différentes aux constantes lors d'exécutions successives du programme.
Une constante est définie comme *DEFN,name value et on y fait référence dans le texte du programme sous la forme de #name# . Pour changer la valeur de la constante au cours d'une autre exécution du programme, il suffit de changer la valeur "value" qui a été assignée à cette constante dans sa définition.

4. Instruction de format

L'instruction *FORM remplace l'instruction FORMAT du fortran.
Sa syntaxe est la suivante :

```
*FORM,number
     (format

                )
*FEND
```

5. Conventions d'écriture du MAIN et des BLOCK-DATA

a) MAIN : La première ligne du MAIN programme doit être précédée de
*MAIN,MAINxxxx où xxxx représente le nom du programme
ex *MAIN,MAINDI définit le MAIN du programme DI
(Dynamic Interaction)

b) BLOCK-DATA : la première ligne d'un block data doit être précédée de
*BLKD,Byyyyyy où yyyyyy représente le nom du COMMON que le block data initialise.

Les cinq points que nous venons de voir concernent principalement les définitions et les initialisations d'éléments du programme. Ils permettent de concentrer ces définitions de sorte que, s'il y a une modification à ajouter, on sache rapidement la localiser, ce qui facilite l'implémentation sur des machines différentes.

II.2.3. Exécution des programmes

Deux modes d'exécution des programmes sont prévus :

- le mode CLI qui passe par un interpréteur de commandes (command Language Interpreter)
- le mode STA (Stand Alone).

L'utilisateur travaille en mode CLI quand il utilise les commandes mises à sa disposition par PSA pour l'édition de rapports. CLI est entre autres un interpréteur de commandes de PSA.
Le mode STA (Stand Alone) permet de concevoir et de tester des programmes indépendamment de l'environnement PSA.

Conclusion

a) Point de vue langage

Le langage utilisé (Extended Fortran) est assez simple à utiliser, d'autant plus qu'il existe des programmes de conversion qui traduisent un programme écrit en Extended Fortran en fortran standard et vice-versa.

b) Point de vue de la structuration des programmes

L'analyse de la structure attachée aux programmes nous montre combien leur organisation est complexe.

Le rangement de routines dans les différentes librairies présente un grand avantage. En effet, lors de l'élaboration de programmes permettant l'édition de nouveaux rapports, on peut aisément faire appel aux routines existantes et le travail s'en trouve simplifié.

Cette organisation présente aussi certains inconvénients.
Le principal inconvénient est lié à la façon dont le projet Isdos est organisé. La plupart des membres de l'équipe du Michigan sont des étudiants. Ils travaillent au projet pendant une période d'une durée variant généralement entre un et deux ans.
Avant de pouvoir vraiment effectuer un travail productif, ils doivent se mettre au courant des objectifs du projet, de ce qui a été fait et de ce qui reste à faire. Ils doivent également connaître les possibilités des routines existantes. Or, il n'existe aucun document présentant de manière complète mais synthétique ce genre d'informations. La documentation concernant ces routines présente un caractère trop sommaire, si bien qu'elle ne peut pas être très utile lorsqu'on veut se rendre compte de l'étendue des possibilités offertes. L'étude du contenu de chaque librairie est longue et fastidieuse d'autant plus qu'il existe généralement plusieurs versions d'une même librairie.

Un effort en vue de répondre à cette critique a été fait à Michigan : il s'agit de l'élaboration du système SMS (24).
Ce système permet un accès assez aisé aux différentes routines des différentes versions des librairies. Il est assez performant mais présente encore une restriction : l'utilisateur doit spécifier certains éléments comme la version et la librairie dans laquelle se trouve la routine à laquelle il veut accéder. L'équipe Isdos se propose d'améliorer ce système.

---

(24) E.A. HERSHEY, Y. YAMAMOTO : "SMS User's Manual", preliminary release, Isdos Project, réf WP 232, September 1978.

CHAPITRE II.3 - CAS PARTICULIER DE RAPPORT : LE DYNAMIC INTERACTION

Pour illustrer l'aspect programmation des rapports Isdos, nous prenons l'exemple du DYNAMIC INTERACTION, rapport qui fonctionne pour la version 5.1 et que nous avons adapté à la version NAMUR.

Ce chapitre présente parallèlement pour la version 5.1 et pour la version NAMUR

- les objectifs du rapport DYNAMIC-INTERACTION (DI)
- le format
- les options
- l'analyse effectuée par le rapport
- les limitations

II.3.1. Objectifs du rapport

Dans la version 5.1, ce rapport existe et présente l'aspect dynamique de la sous-structure des traitements décrits par l'utilisateur.

Il montre quels sont les objets PSL sources d'actions dynamiques, les actions qu'ils accomplissent, l'ordre dans lequel ils le font et les objets PSL résultats de ces actions dynamiques.

Etant donné le but de ce rapport, il paraît important de l'adapter aux objets et aux actions de la version NAMUR qui justement a pour objectif d'insister sur une description rigoureuse de la dynamique des traitements.

II.3.2. Format

Dans la version 5.1 et dans la version NAMUR, le rapport DI est présenté à l'utilisateur sous forme matricielle.

Voyons la présentation pour chacune de ces versions :

Format version 5.1

Le rapport présente d'abord deux listes de noms. Ces deux listes représentent respectivement les lignes et les colonnes de la matrice. La constitution de ces listes se fait en plusieurs étapes.
Les lignes sont d'abord constituées des objets PSL, sujets d'une action dynamique, introduits en inputs pour le rapport. Ces objets agissent sur d'autres objets PSL, résultats d'action dynamique. Ces objets-"résultats" constituent les colonnes. La partie gauche d'une action (représentée par une relation PSL) se trouve toujours dans la liste des lignes et la partie droite de la relation est, elle, toujours dans la liste des colonnes.
Les objets-"résultats" sont alors repris comme inputs, c'est-à-dire placés à la suite des lignes déjà existantes. Ils déclenchent à leur tour des actions et donc d'autres objets, résultats de ces actions. Ces derniers objets complètent les colonnes existantes avant d'être repris comme lignes. La procédure se poursuit ainsi jusqu'à la fin de l'enchaînement, c'est-à-dire jusqu'au moment où plus aucune colonne ne peut être ajoutée étant donné qu'aucun élément n'est repris plus d'une fois dans une liste.

A la suite de ces listes, le rapport imprime alors la matrice. Elle représente les événements en ligne et en colonne selon leur ordre d'insertion dans les deux listes. A l'intersection d'une ligne et d'une colonne est associée une lettre représentant l'action unissant les deux événements. Il y a une lettre par action possible. Le rapport présente la signification de ces lettres en option (ex : A means : row i TRIGGERS column j).

Après avoir imprimé cette matrice, le rapport communique à l'utilisateur les résultats d'une analyse de la dynamique (cfr. paragraphe : analyse effectuée par le rapport p. 134).

Donnons un bref exemple de ce rapport :
Soit l'exemple du fonctionnement d'un système d'urgence p. 64
Le PSL de cet exemple est le suivant :

```
* DEFINE EVENT ev-declenchement-d'un-incendie;
      TRIGGERS prc-appel-900;
* DEFINE PROCESS prc-appel-900;
      INCEPTION-CAUSES ev-declenchement-alerte;
* DEFINE EVENT ev-declenchement-alerte;
      TRIGGERS prc-extinction-incendie;
* DEFINE PROCESS prc-extinction-incendie;
      TERMINATION-CAUSES ev-fin-alerte;
```

Si on donne comme input l'événement "ev-déclenchement-d'un-incendie", le rapport se présente de la façon suivante :

Row Names	Type
1. ev-declenchement-d'un-incendie	EVENT
2. prc-appel-900	PROCESS
3. ev-declenchement-alerte	EVENT
4. prc-extinction-incendie	PROCESS
5. ev-fin-alerte	EVENT

Column Names	Type
1. prc-appel-900	PROCESS
2. ev-declanchement-alerte	EVENT
3. prc-extinction-incendie	PROCESS
4. ev-fin-alerte	EVENT

Dynamic Interaction Matrix

	1	2	3	4
1	A			
2		O		
3			A	
4				P
5				

Meaning (optionnel)

A : row i TRIGGERS column j
O : row i INCEPTION-CAUSES column j
P : row i TERMINATION-CAUSES column j

Format_version_NAMUR

Nous venons d'examiner la forme du rapport pour la version 5.1. Voyons maintenant les modifications apportées au format de ce rapport, pour présenter à l'utilisateur l'aspect dynamique des traitements de la version NAMUR.

Une optique de travail assez différente a été choisie au départ, ce qui modifie sensiblement la forme du rapport : cette optique a été proposée par l'équipe Isdos de NAMUR dans le but de réaliser une étude aussi complète que possible de la dynamique.

Pour comprendre cette optique, il faut se rappeler que l'on dispose dans la version NAMUR de

- quatre objets PSL à la fois sources d'actions dynamiques et résultats d'actions,à savoir : EVENT - MESSAGE - PROCESS - SYNCHRONIZATION-POINT
- six changements d'état incorporés à la plupart des actions dynamiques et étant, en fait, les sujets de ces actions à savoir : ON INCEPTION - ON INTERRUPTION - ON RESUMPTION - ON ABORTION - ON TERMINATION - ON GENERATION

Dans la version NAMUR, les deux listes présentées en tête du rapport sont construites de la même manière que pour la version 5.1 et reprennent uniquement les objets PSL sources d'actions dynamiques.

En ce qui concerne la matrice, elle présente une différence fondamentale qui a des conséquences importantes au point de vue programmation : si la correspondance entre les objets présentés dans la liste Column Names et les colonnes de la matrice est inchangée,
par contre, il n'y a plus de correspondance entre les objets présentés dans la liste Row Names et les lignes "en-tête" de la matrice.

Cette distorsion entre les objets présentés dans la liste Row Names et les lignes "en-tête" de la matrice provient d'une option qui a été prise dès le départ :

dans la liste des Row Names, ne sont repris que les objets PSL, sources d'action dynamique,
par contre, dans les lignes "en-tête" de la matrice sont repris les objets PSL et les changements d'état de ces objets, sources d'actions dynamiques. De plus, les lignes "en-tête" de la matrice font apparaître explicitement les conditions dont dépendent les actions.

Expliquons pourquoi nous avons choisi cette distorsion.

1) introduction des changements d'état dans les lignes "en-tête" de la matrice

Rappelons tout d'abord que les objets PSL et les changements d'état de ces objets peuvent être sources de sept types d'action. Ces actions sont :

GENERATES - TRIGGERS - INTERRUPTS - RESUMES - ABORTS - CONTRIBUTES - UTILIZES (en option)

Or, dans la description du langage en Meta-langage, ces actions sont individualisées en fonction du sujet de l'action.

Ainsi les deux actions : ON INCEPTION TRIGGERS
ON INTERRUPTION TRIGGERS (1)
sont enregistrées selon deux codes de relation distincts.
Il a été décidé pour le rapport de les considérer comme un seul type d'action déclenché par le changement d'état d'un objet.
La distinction entre les deux actions (1) apparaît donc dans les lignes "en-tête" de la matrice et non dans le code représentant l'action à l'intersection d'une ligne et d'une colonne.

Les changements d'état n'apparaissent pas dans la liste des Row Names car ils sont toujours relatifs à un objet et nous avons voulu éviter toute redondance d'objets à ce niveau.
Ainsi, on n'a pas : process-p
INCEPTION OF process-p
TERMINATION OF process-p
dans la liste des Row Names. Mais on a uniquement process-p.

2) introduction des conditions dans les lignes "en-tête" de la matrice

Dans la description du langage en Meta-langage, toute action, si elle peut être conditionnelle, est enregistrée par deux codes distincts selon que la condition à laquelle elle est soumise, doit être vraie ou fausse.
Par exemple, ON INCEPTION TRIGGERS process-name (IF condition-name IS TRUE)
ON INCEPTION TRIGGERS process-name (IF condition-name IS FALSE)
sont enregistrés selon deux codes de relation distincts.
Il a été décidé pour le rapport de les considérer comme un seul type d'action. Le code associé à une relation ne permet donc pas d'identifier la valeur de la condition. C'est pourquoi nous avons introduit dans le rapport une troisième liste qui reprend toutes les conditions nuançant les actions inter-

venant dans le rapport, de façon à pouvoir faire référence à ces conditions.

Nous venons de voir quelles sont les raisons qui nous ont conduit à introduire une distorsion entre les éléments de la liste Row Names et les éléments intervenant dans les lignes "en-tête" de la matrice.
Voyons maintenant comment ces éléments sont repris dans l'output du DI.

Output du DI

Nous avions choisi au départ d'adopter la présentation de la version 5.1 c'est-à-dire de faire référence aux éléments se trouvant dans chacune des listes en les représentant par leur numéro d'ordre dans la liste.
Etant donné que l'on doit référencer ici trois listes (Row - Column - Condition), la solution de la représentation par le numéro d'ordre s'est avérée difficile à lire. C'est pourquoi, nous avons modifié la présentation du rapport DI.

Chaque ligne de la matrice possède maintenant un en-tête comprenant les éléments suivants :

1. abréviation de l'objet PSL ou du changement d'état, source de l'action
   ex EVT = Event
      INC = Inception
2. abréviation de l'objet PSL auquel la source de l'action fait référence
   - soit la source de l'action est un objet PSL et dans ce cas, on reprend uniquement la première lettre de cet objet
     ex EVT (= Event) E
   - soit la source de l'action est un changement d'état relatif à un type d'objet PSL et on reprend alors l'abréviation (1 lettre) de ce type d'objet
     ex INC (= Inception) P signifie Inception of Process

3. l'objet PSL qui est le sujet de l'action, soit direct (cas de EVENT), soit indirect (cas d'un changement d'état) est imprimé en toutes lettres.
4. dans le cas où l'action déclenchée est conditionnelle, on imprime IF-nom de la condition en toutes-lettres-abréviation de la valeur de la condition (T=True-F=False).

Signalons, en dernier lieu, que toutes les lignes blanches de la matrice ont été supprimées. Ceci évite de nombreux vides.
En effet, suite à un Process, par exemple, six sujets d'action peuvent être envisagés.
Par exemple, . si aucun objet ne succède à ce Process, il y aurait donc six lignes blanches dans la matrice
. si seulement un type d'action est attaché à ce Process, il y aurait une ligne significative pour cinq lignes blanches.

A partir de ces bases théoriques, voyons maintenant ce que devient le rapport appliqué à l'exemple du fonctionnement d'un service d'urgence (p. 90).
Cet exemple est libellé de la façon suivante dans le PSL, version NAMUR.

```
DEFINE EVENT ev-declenchement-d-un-incendie;
   TRIGGERS prc-appel-900;

DEFINE PROCESS prc-appel-900;
   ON INCEPTION TRIGGERS prc-extinction-d-un-incendie;

DEFINE PROCESS prc-extinction-d-un-incendie;
   ON INCEPTION TRIGGERS prc-hospitalisation
                IF cond-existence-de-blesses IS TRUE;
   ON TERMINATION GENERATES msg-fin-alerte;
```

Row Names

1	ev-declenchement-d-un-incendie	EVENT
2	prc-appel-900	PROCESS
3	prc-extinction-d-un-incendie	PROCESS
4	mss-fin-alerte	MESSAGE
5	prc-hospitalisation	PROCESS

Column Names

1	prc-appel-900	PROCESS
2	prc-extinction-d-un-incendie	PROCESS
3	prc-hospitalisation	PROCESS
4	mss-fin-alerte	MESSAGE

CONDITION NAMES

1	cond-existence-de-blesses	CONDITION

DYNAMIC INTERACTION MATRIX

			mss-fin-alerte	prc-hospitalisation	prc-extinction-d-un-incendie	prc-appel-900
EVT	E	ev-declenchement-d-un-incendie			T	
RLZ	S	ev-declenchement-d-un-incendie			T	
INC	P	prc-appel-900			XT	
INC	P	prc-extinction-d-un-incendie	IF cond-existence-de-blesses	T	XT	
TER	P	prc-extinction-d-un-incendie			X	O

Si on donne comme input l'événement "ev-declenchement-d'un-incendie", le rapport DI est présenté ainsi :

Signification (optionnel)

EVT means EVENT
INC means INCEPTION
TER means TERMINATION

T means : row i TRIGGERS column j
G means : row i GENERATES column j

La version NAMUR du DI est donc tout à fait spécifique. Le problème des index différents dans Row Names et pour les lignes de la matrice provoque des difficultés importantes au point de vue programmation : le programme du DI version NAMUR est donc beaucoup plus complexe que celui du DI version 5.1 du fait d'une gestion d'index plus élaborée.

II.3.3. Les options du DI

L'utilisateur peut choisir dans une certaine mesure ce qu'il veut voir figurer dans son rapport et il fixe ce choix à l'aide de différentes options qui sont identiques pour les deux versions que nous analysons (5.1 et NAMUR).

Le rapport doit porter sur un des objets ou un groupe d'objets PSL décrits dans la base de données, et susceptibles d'être source d'action dynamique.

Les listes d'objets peuvent être présentées dans un ordre déterminé par l'utilisateur (sauf l'ordre de la 3ème liste - CONDITION NAMES - introduite dans la version NAMUR).

- les événements peuvent être présentés dans l'ordre dans lequel le programme les retrouve dans la base de données. Il s'agit de l'ordre standard.
- ils peuvent être présentés par type d'événement (ordre = ALPHA)
- ils peuvent être présentés par ordre alphabétique dans chaque catégorie d'événement (ordre = BY TYPE).

Nous avons, en outre, implémenté deux options différentes au point de vue de la présentation de la matrice dans le cas où l'utilisateur spécifie un ordre différent de l'ordre standard.
En effet, la version 5.1, dans le cas où l'ordre n'est pas standard, modifie les lignes et les colonnes de la matrice en fonction des numéros d'ordre se trouvant dans les listes des Row et Column Names. L'aspect de la matrice, qui était au départ, à tendance "diagonale" est donc tout à fait quelconque, ce qui rend la lecture de cette matrice plus difficile.

Nous avons pris comme option dans la version NAMUR de laisser à la matrice son aspect premier, même si un ordre différent de l'ordre standard est spécifié.
Nous avons toutefois implémenté les deux solutions, ce qui permettrait d'ajouter un paramètre et de donner ainsi à l'utilisateur la possibilité de choisir la solution qu'il préfère.

La matrice ainsi que l'analyse dynamique peuvent être imprimées indépendamment l'une de l'autre.

L'analyse de la relation "UTILIZES" entre deux "PROCESS" est optionnelle.

Un nombre maximum de niveaux de relations peut être déterminé par l'utilisateur. Il y a un niveau de relation chaque fois qu'est exécutée la procédure qui construit progressivement la liste des Row Names.

II.3.4. Analyse effectuée par le rapport DI

Le rapport DI de la version 5.1 analyse l'information qu'il a présentée sous forme matricielle.
Etant donné la différence de présentation de cette information dans la version 5.1 et dans la version NAMUR, l'analyse de l'information présente des caractéristiques différentes pour chacune de ces versions.

Analyse dans la version 5.1

L'analyse de l'information est conçue de façon à donner à l'utilisateur le résultat de tests :

- un test de "completeness"
- un test de "consistancy"

a) completeness

Le DI de la version 5.1 met en valeur chaque objet qui se situe à la fin de l'enchaînement et qui n'agit donc pas sur le système.
Il signale
- les "PROCESS" qui n'ont pas de successeur
- les "EVENT" dont l'occurrence ne déclenche aucune action
- les "CONDITION" dont le changement d'état ne déclenche aucune action.

Il présente également chaque processus qu'aucun objet ne lie par l'intermédiaire d'une relation "TRIGGERS" ou "UTILIZES" (en option).

b) consistancy

Le rapport DI de la version 5.1 détecte quels sont les circuits possibles dans l'information qu'il présente.
Les circuits détectés sont d'une longueur maximum de deux.

Par exemple : INCEPTION OF prc-1 CAUSES ev-1;
ev-1 TRIGGERS prc-1;

Nous avons adapté ces deux tests à la version NAMUR. Voyons l'analyse présentée pour cette version.

Analyse dans la version NAMUR

a) completeness

Le DI de la version NAMUR met en valeur chaque objet qui se situe à la fin de l'enchaînement et qui n'agit donc sur le système, ni directement, ni par l'intermédiaire d'un changement d'état.
Cette analyse est semblable à celle de la version 5.1.

Le rapport signale
- les "PROCESS" qui n'ont pas de successeur
- les "EVENT" dont l'occurrence ne déclenche aucune action
- les "SYNCHRONIZATION-POINT" dont la réalisation ne déclenche aucune action
- les "MESSAGE" qui ne déclenchent aucune action lorsqu'ils sont générés.

Il présente également, tout comme la version 5.1, chaque processus qu'aucune source d'action ne lie par l'intermédiaire d'une relation "TRIGGERS" ou "UTILIZES" (en option).

b) consistancy

Nous avons vu quelle est l'importance de la cohérence au point de vue dynamique dans la version NAMUR. Il est donc indispensable de fournir à l'utilisateur un moyen assez per-

formant pour lui permettre de tester cette cohérence.
Nous avons donc amélioré le test de consistancy qui existe dans la version 5.1.
Rappelons que l'analyse dans cette version ne détecte que les circuits d'une longueur maximum de deux.

Méthode

Nous avons transformé cette analyse dans la version NAMUR de façon à ce qu'elle puisse détecter les circuits à tous les niveaux.
Pour détecter un circuit dans une matrice, plusieurs méthodes sont possibles (25) par exemple, la méthode de la multiplication latine, l'algorithme E.C (J.C. Tiernan) et enfin une méthode heuristique basée sur le dictionnaire du graphe.

Nous avons choisi cette troisième méthode pour deux raisons :

1) dans le cas qui nous occupe, les deux premières méthodes étaient difficiles à appliquer vu que nous ne disposons pas d'une matrice carrée.
2) Nous nous sommes basé sur le fait que la construction de la matrice nous permet de mémoriser les éléments qui pourraient former un circuit.
   En effet, une nouvelle colonne est créée pour tout événement n'appartenant pas encore à la liste des colonnes.
   Seuls les événements en relation avec des événements qui se trouvent déjà en colonnes risquent donc de provoquer des circuits.
   Lors de la construction des colonnes, nous mémorisons les événements pour lesquels aucune colonne nouvelle n'est créée et qui sont en relation avec des colonnes existantes.
   Pour chacun de ces éléments, lors de l'analyse, nous construisons le dictionnaire du graphe qui nous permet de dé-

---

(25) FICHEFET Jean : op. cit. en (14) p. 66.

tecter un circuit éventuel. Nous expliquons cette partie du programme en détails dans l'Annexe 2.

Les circuits ne constituent pas nécessairement une erreur. Nous en avons vu quelques exemples lors de l'analyse de la dynamique.
Par conséquent, nous signalons simplement à l'utilisateur qu'il y a un circuit et il lui appartiendra de corriger l'incohérence s'il s'agit réellement d'une incohérence.

II.3.5. Limitations

Nous analysons dans ce paragraphe :
- les limitations attachées au programme
- les restrictions d'applicabilité de ce programme.

1) Limitations attachées au programme

Ces limitations concernent trois aspects, tant pour la version 5.1 que pour la version NAMUR :

a) aspect analyse des circuits
b) aspect mode de travail
c) aspect dimension.

version 5.1

En ce qui concerne l'analyse des circuits, nous avons vu combien le fait que la version 5.1 ne détecte que les circuits de longueur deux, constitue une limite pour l'utilisateur.

Au point de vue mode de travail, le rapport retrouve toujours l'information en travaillant en mode "FORWARD" c'est-à-dire qu'il retrouve les relations actives TRIGGERS - INTERRUPTS etc. et non les relations passives TRIGGERED - INTERRUPTED etc.

Au point de vue de la dimension, la taille maximum des matrices est de 400 x 400 et le nombre maximum de liens est de 50.

version NAMUR

En ce qui concerne l'analyse des circuits, la version NAMUR peut détecter tous les circuits de n'importe quelle longueur.
Les informations données par l'analyse des circuits ne sont

cependant cohérentes que lorsque l'ordre paramétré par l'utilisateur est l'ordre standard.

En ce qui concerne le mode de travail et la dimension, les mêmes restrictions sont d'application pour la version NAMUR et pour la version 5.1.

2) Restrictions d'applicabilité

version 5.1

Le Dynamic Interaction fourni pour la version 5.1 peut tourner sur tous les langages de type binaire à condition d'effectuer quelques modifications en ce qui concerne :

- les types d'objet pouvant être analysés par le rapport
- les relations pouvant être analysées par le rapport et liant les types d'objet
- la routine (DIBMAT), qui, en fonction du type d'objet, appelle la routine (SUCORS) qui recherche les successeurs d'un objet
- la routine (DISUM) qui effectue les tests de completeness et de consistancy et la routine (DIMESS) qui imprime les messages relatifs à ces tests.

version NAMUR

En ce qui concerne la version NAMUR du Dynamic Interaction, elle se base sur la version 5.1 et est modifiée pour tourner dans la version META sur un langage de type ternaire.

Pour cette version, plusieurs options ont été prises en fonction de l'output spécifique que l'on voulait obtenir. Ainsi, on a distingué les objets PSL et les changements d'état, sources d'actions dynamiques. On a également introduit les conditions de façon explicite.

Cette version du DI pourrait tourner sur un langage de type ternaire, à condition de modifier :

- les types d'objets pouvant être analysés par le rapport
- les changements d'état intervenant dans la dynamique
- les relations pouvant être analysées par le rapport et liant 1) les types d'objet entre eux
  2) les changements d'état et les types d'objet
- tout ce qui concerne la mémorisation et l'impression des conditions
- l'analyse de completeness et de consistancy et les messages imprimés en fonction des types d'objet.

En résumé, on peut dire que cette version du DI est aisément applicable à un autre langage ternaire s'il est structuré de façon identique c'est-à-dire :

source d'action - action - condition - valeur

car les modifications à apporter alors sont pour la plupart centralisées dans les block data (relations - types d'événements - etc.).
Un grand nombre de modifications surtout au point de vue de la gestion des index devrait être apporté si le langage était structuré de manière différente.

II.3.6. Présentation du DI - version NAMUR

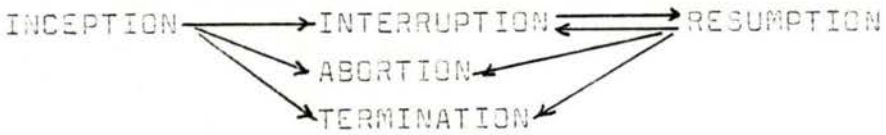
Etant donné qu'Isdos constitue un système d'information qui a pour but de développer des outils d'aide à l'analyse et à la conception de systèmes d'information, il est très envisageable que le projet Isdos gère ses propres informations en employant les moyens qu'il développe.

Nous avons donc employé les outils de documentation proposés par Isdos pour décrire le programme du DYNAMIC-INTERACTION. La façon dont la description a été effectuée et la description elle-même figurent dans l'annexe 3.

II.3.7. Extension et conclusion

Le DI version NAMUR pourrait, à notre avis, être l'objet de deux améliorations :

1) Une troisième analyse pourrait être inclue concernant les transitions possibles entre les états d'un processus.
Voici le schéma des transitions possibles :



Nous pourrions donc avoir les contrôles suivants :
- process RESUMED and never INTERRUPTED
- process INTERRUPTED and never RESUMED
  (il devrait être ABORTED)
- process not TRIGGERED and INTERRUPTED or ABORTED or RESUMED or TERMINATED.

2) La deuxième extension est beaucoup plus complexe.
Elle adapterait le DI mais à la place de représenter et d'analyser des événements, elle travaillerait sur la sous-structure des ressources.
Le DI présenterait donc les "PROCESS", les "PROCESSOR" qui les accomplissent et les "CONSUMABLE or "REUSABLE RESOURCE" utilisées par les "PROCESSOR" pour accomplir les "PROCESS".

Le DI a donc été adapté à la version NAMUR mais ses possibilités sont loin d'être exploitées au maximum.

Conclusion de la Deuxième Partie

Au cours de cette Deuxième Partie, nous avons développé la conception d'un moyen permettant de tester les informations décrites à l'aide de la version NAMUR : le rapport DYNAMIC-INTERACTION (DI).

Nous avons, tout d'abord, expliqué les outils qui ont permis d'élaborer cette version : le META-SYSTEM et le GENERALYZED ANALYZER.

Nous avons également analysé l'environnement de programmation dans lequel se situe le DI. La façon dont cette programmation est conçue dans le projet Isdos nous a permis de faire appel à bon nombre de moyens existants (routines des différentes librairies) et également de localiser les instructions à modifier dans le programme tel qu'il est écrit pour la version 5.1, de façon à l'adapter à la version NAMUR. Tel était l'objet des deux premiers chapitres.

Dans le troisième chapitre nous avons analysé le programme de DI tel qu'il existe pour la version 5.1. Nous avons expliqué les options que nous avons choisies pour l'adapter à la version NAMUR.
Ces options ont été dictées par le souci d'apporter à l'utilisateur une vue synthétique de la description de la dynamique de son système et de lui fournir une analyse de cohérence aussi complète que possible.

TROISIEME PARTIE

S Y N T H E S E E T C O N C L U S I O N

Cette dernière partie est subdivisée en trois chapitres.

Le premier chapitre présente la synthèse du mémoire.

Au cours du second chapitre, nous apprécions le travail effectué tant au point de vue théorique qu'au point de vue pratique.

Le troisième chapitre concerne les extensions envisageables vis-à-vis de notre étude.

III.1. SYNTHESE

Pour répondre aux difficultés posées par la gestion de systèmes d'informations de plus en plus complexes - systèmes situés dans des environnements eux-mêmes évolutifs - l'université du Michigan, à l'initiative du Pr D. Teichroew, a développé le projet Isdos.

Ce projet a donc pour objectif d'élaborer des outils d'aide à l'analyse et à la conception de systèmes d'informations.

Nous avons analysé trois de ces outils en fonction des objectifs qu'ils s'étaient fixés. Deux de ces outils ont été développés à l'Université du Michigan : ce sont les versions 4.2 et 5.1. La troisième version est la version élaborée à l'Institut d'Informatique de NAMUR (version "NAMUR").

La version 4.2 a pour but de développer surtout les spécifications statiques. Elle permet une description complète d'un système d'informations à l'aide de concepts assez généraux. Cette généralité laisse l'utilisateur libre de définir des caractéristiques particulières à son système.

La version 5.1 a pour but d'intégrer la description des aspects dynamiques du système. Elle donne à l'utilisateur le moyen de décrire de façon détaillée les aspects utilisation des données, enchaînement dynamique des traitements et exploitation des ressources. Cependant, elle n'autorise pas la définition d'actions conditionnelles.
La version 5.1 prévoit un grand nombre de moyens permettant à l'utilisateur de retrouver et d'analyser les informations qu'il a décrites, selon le format qu'il désire.

La version NAMUR s'est fixé comme objectif de simuler un système à partir des spécifications le décrivant. Elle développe donc les moyens permettant de décrire les aspects dynamiques d'un système avec rigueur et précision. Jusqu'à présent, elle a essentiellement étudié la dynamique des traitements et l'exploitation des ressources.
Elle prévoit cependant de compléter l'aspect dynamique de la sous-structure des données et les aspects statiques de la description du système. D'autre part, elle développe les moyens permettant à l'utilisateur de retrouver les informations qu'il a décrites en insistant sur la présentation et l'analyse des aspects dynamiques.

Ces trois versions sont donc développées dans un certain contexte et ont des buts bien définis. L'utilisateur qui devrait choisir une de ces versions devrait donc le faire en fonction de l'objectif qu'il poursuit en décrivant son système.

Au cours de la première partie du mémoire, nous avons analysé ces trois versions par rapport à leurs objectifs en insistant sur les aspects dynamiques.

Dans la deuxième partie de ce travail, nous avons élaboré un moyen permettant à l'utilisateur d'avoir une vue synthétique de la dynamique des traitements dans son système : le rapport DYNAMIC-INTERACTION.
Nous avons adapté ce rapport à la version NAMUR sur base de ce même rapport travaillant sur les informations de la version 5.1 tout en respectant l'esprit de programmation du projet Isdos.

Nous avons mis en évidence :

- les types d'objet PSL et les changements d'état, sources d'actions dynamiques
- les conditions nuançant les actions intervenant dans l'enchaînement dynamique des traitements

De plus, étant donné l'importance de la cohérence des informations dans une optique de simulation, nous signalons à l'utilisateur :

. les événements caractérisant la fin des enchaînements
. les événements qui ne sont pas reliés par une relation de type "déclenchement" ou "utilisation"
. les circuits apparaissant dans la dynamique
Vu les conséquences que provoquerait la simulation de circuits, nous détectons dans notre analyse de la version NAMUR, les circuits de toute longueur (Dans la version 5.1, seuls les circuits de longueur deux sont détectés).

III.2. APPRECIATION DU TRAVAIL EFFECTUE

1. Connaissances apportées par l'étude de la dynamique

Le logiciel Isdos permet d'améliorer le processus d'élaboration d'un système d'informations.
Les différentes versions du projet Isdos ont développé des concepts qui permettent de standardiser et de schématiser les aspects statiques et dynamiques dans un système.
Les concepts classifient les éléments d'un système en faisant apparaître les liens existant à l'intérieur et entre les différents niveaux.

Notre étude de la dynamique dans le projet Isdos nous a amené à détailler cet aspect à travers des exemples concrets.
Elle nous a fait découvrir toutes les facettes des problèmes

posés par les enchaînements au sein des différentes sous-structures caractérisant un système.

D'autre part, nous avons signalé que la version NAMUR est toujours en cours de développement. Cette version a évolué tout au long de l'année académique 1978-1979. Son évolution est le résultat de nombreux problèmes découverts au cours d'analyses détaillées. Ces analyses nous ont permis d'évaluer les difficultés posées par la définition d'un langage. Les spécifications constituant la versionNAMUR, particulièrement en ce qui concerne la sous-structure des ressources, représentent un domaine où les implications des définitions sont difficilement contrôlables.

2. Connaissances apportées par la pratique d'un logiciel d'aide à la conception de systèmes d'informations

Durant ce travail, nous avons eu à plusieurs reprises l'occasion de nous familiariser avec la programmation du logiciel Isdos.

La conception modulaire des différents programmes est appréciable puisqu'elle permet d'exploiter ce qui a déjà été réalisé (routines des différentes librairies), bien que cette conception puisse poser des problèmes au point de vue de l'apprentissage.

Nous avons pu travailler avec le logiciel Isdos sur différentes machines (Siemens - Amdahl - Dec/Digital) et ce grâce au caractère portable de ce logiciel.

Le travail de documentation d'un programme est long et fastidieux mais il est apparu qu'une fois la documentation réalisée, la compréhension du programme est plus aisée et plus rapide.

Le projet Isdos est avant tout un travail d'équipe et l'aspect programmation de ce mémoire nous a montré l'importance et la difficulté de la coordination entre les différents membres de l'équipe.

III.3. EXTENSIONS

1. Analyse des systèmes d'information

Les concepts du logiciel Isdos ne sont pas évidents et à plusieurs reprises, nous avons pu nous rendre compte de la difficulté d'appliquer pratiquement ces concepts, tout en respectant une certaine cohérence au niveau de la description du problème.

2. Méthodologie

Une fois les informations enregistrées, il faut les analyser et les tester. Il est important pour l'utilisateur de déterminer la façon dont il veut retrouver les informations qu'il a décrites avant d'en entamer la description à l'aide de PSL.
Une analyse détaillée doit donc précéder la description du problème et de ce point de vue, Isdos est un langage d'aide à la conception de systèmes d'informations plutôt qu'un véritable outil.

Pour être un véritable outil, nous pensons qu'il devrait être accompagné d'une méthodologie d'analyse, applicable en toute généralité.

3. Pédagogie

Les lacunes existant au point de vue documentation ont été signalées dans ce travail.

Du point de vue del'utilisateur, l'apprentissage d'Isdos semble assez long étant donné le fait que les concepts utilisés sont, pour la plupart, connus de l'utilisateur et proches de leur signification en anglais courant.

L'utilisateur risque de ne pas exploiter au maximum les possibilités que lui offre Isdos.
Les liens entre le logiciel et l'utilisation de ce logiciel devront faire l'objet d'une étude au point de vue pédagogique, de façon à faire d'Isdos un réel outil.

BIBLIOGRAPHIE

BENCI E., BODART F., BOGAERT H. et CABANES A. : "Concepts For the Design of a Conceptual Schema", in G.M. Nijssen ed., "Modelling in Data Base Management Systems", North Holland Publishing Company, 1976.

BODART F. : "Problèmes d'Organisation et Méthodes d'analyse fonctionnelle" : cours FNDP; 1978-1979, Namur.

BODART F. et PIGNEUR Y. : "A Model and a Language for functional Specifications and Evaluation of Information System Dynamics", Working Conference - Oxford - 1979.

CODASYL COBOL DATA BASE FACILITY PROPOSAL : Department of Supply and Services, Ottawa, Ontario, Canada, March 1973.

DURIGNEUX Francis : "Mise en oeuvre d'une méthode d'analyse fonctionnelle à l'aide du logiciel ISDOS", Mémoire, Facultés Universitaires Notre-Dame de la Paix à Namur, Institut d'Informatique, 1978.

FICHEFET J. : "Théorie des Graphes", cours FNDP, 1977-1978, Namur.

FORRESTER W. Jay : "World Dynamics", Wright Allen Press, 1971, pp. 20-23.

MARCH J.G. et H.A. SIMON : "Les Organisations", Dunod 1969.

LANGEFORS Borge : "Computer-Aided Information System Design", in J. Bubenko, B. Langefors and A. Solvbery (Eds.), "Computer-Aided Information Systems Analysis and Design".

LAWRENCE R. Paul and Jay W. LORSCH : "Organization and Environment", Harvard University, Boston, 1967.

POPPER Jacques : "La dynamique des Systèmes", Les Editions d'Organisation, Eyrolles éditeur, Paris, 1973.

Références ISDOS

Références générales

"Procedures and Conventions for Developing Users' Software for PSA" - Isdos Project - WP 164.

HERSHEY A. : "A Data Base Management System (ADBMS) based on DBTG 71" - revised for D3.0 by G. Fleming - Isdos Project - WP 191 - November 1977.

HERSHEY A., MORAES P., YAMAMOTO Y. : "SMS User's Manual" (preliminary release) - Isdos Project - WP 232 - Sept 1978.

SPEWAK H. Steve : "Analysis of the Logical Design of Information Systems" - University of Michigan (Isdos Project) - WP 260 - February 1979.

HERSHEY A., YAMAMOTO Y., MIETLA D. : "EF Format" - Isdos Project - TM 205 - February 1979.

TEICHROEW D. : "Isdos Meta System Memorandum META-1" - Isdos Project - May 1977 revised February 1978.

Références spécifiques à la version 4.2

TEICHROEW D. and GACKWSKI Z. : "Beginning Use of PSL/PSA" Isdos WP 184, June 1977.

"Problem Statement Language (PSL) - Introduction and Users' Manual, Version 4.2", University of Michigan, Isdos Project, July 1977.

"Problem Statement Language (PSL) - Language Reference Manual, Version 4.2", Isdos Project, May 1977.

"Problem Statement Language (PSL) - Language Reference Summary, Version 4.2", Isdos Project, November 1977.

"Problem Statement Analyzer Reports, Version 4.2", Isdos Project, July 1977.

"Problem Statement Analyzer, Version 4.2; Command Description", Isdos Project, June 1977.

"Problem Statement Analyzer, Version 4.2; Users' Manual", Isdos Project, May 1977.

"Problem Statement Analyzer (PSA), Version 4.2; Command Reference Summary", - Isdos Project, November 1977.

Références spécifiques à la version 5.1

"PSL, Language Reference Summary" - Isdos Project - WP 174 - Mei 1979.

"Overview of the PSA" - Isdos Project - WP 210 - March 1979.

"PSA Reports and Report Commands" - Isdos Project - WP 173 - Jan. 1979.

"PSA Modifier Commands" - Isdos Project - WP 178 - March 1979.

BUMP
0 0 6 5 0 3 7 8 4
*FM B16/1979/12/1

ANALYSE DE LA DYNAMIQUE

DANS LE PROJET ISDOS

ANNEXES

LBS 650 3794
296548

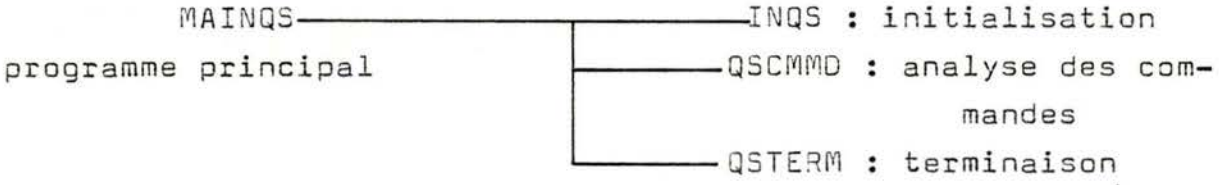
A N N E X E I :

STRUCTURATION DES PROGRAMMES

Voyons un exemple de programmes structurés de façon à ce que chaque routine remplisse une fonction particulière : le Query System.

Le Query System est un programme qui a pour but d'analyser et d'exécuter une dizaine de commandes appartenant à un langage (Query) permettant de sélectionner de l'information dans une base de données (enregistrée suivant la procédure METAsystem et Generalyzed Analyzer). Cette information est sélectionnée suivant certains critères précisés par l'utilisateur.

Le programme de Query System est structuré de la façon suivante :



Le corps du programme, QSCMMD, peut faire appel à dix routines, chaque routine étant chargée d'analyser et d'exécuter une commande spécifique du Query System.

Sa structure se présente ainsi :

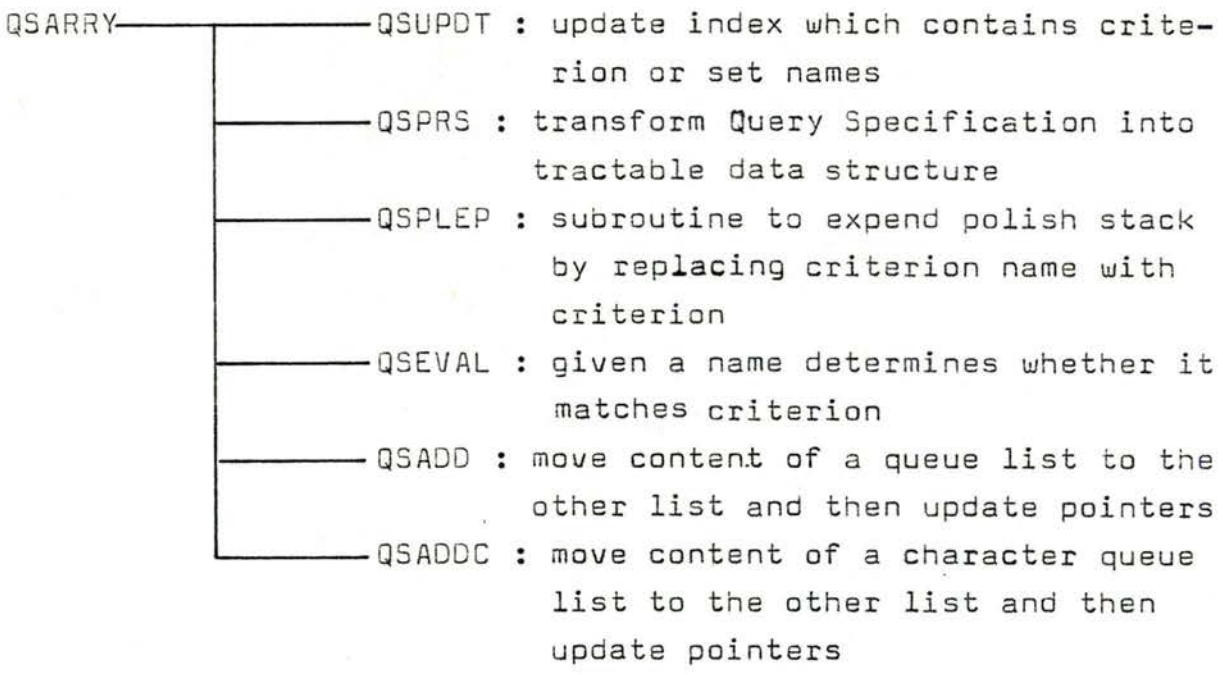
QSCMMD

- QSARRY : commande SET-LET
- QSCHNG : commande CHANGE
- QSCHCK : commande CHECK
- QSCRTN : commande CRITERION
- QSDSPL : commande DISPLAY
- QSEXCT : commande EXECUTE
- QSEXPL : commande EXPLAIN
- QSLIST : commande LIST
- QSPNCH : commande PUNCH
- QSREAD : commande READ

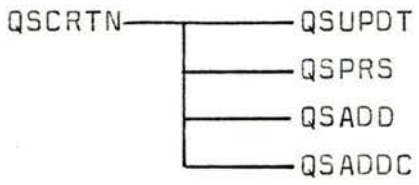
Chacune de ces dix routines appelle un certain nombre d'autres routines, organisées de façon à ce que chaque routine ainsi appelée remplisse une fonction particulière.

Cette structuration fait en sorte que les routines appelées par la routine analysant la première commande sont encore appelées, pour la plupart, par les routines analysant les autres commandes.

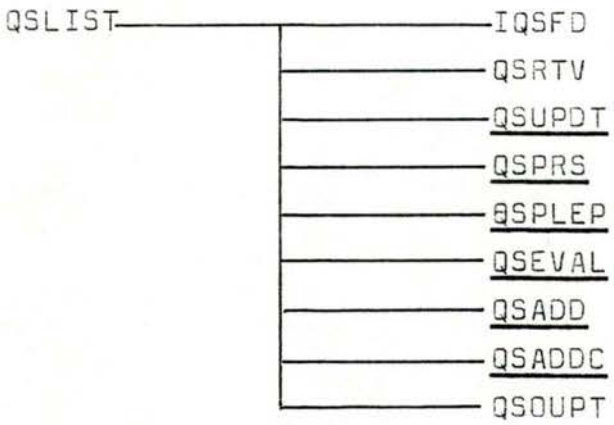
Ainsi, par exemple :



Les routines appelées par QSARRY se retrouvent pour la plupart dans les routines appelées par QSCRTN dont la structure est la suivante :



Elles se retrouvent également pour la plupart dans les routines appelées par QSLIST dont la structure est la suivante :


QSLIST
IQSFD
QSRTV
QSUPDT
QSPRS
QSPLEP
QSEVAL
QSADD
QSADDC
QSOUPT

A N N E X E 2

ANALYSE DES CIRCUITS

Voyons un peu plus en détails comment a été implémentée l'analyse des circuits réalisée par le programme du Dynamic-Interaction.

1) Mémorisation des éléments

Lors de l'insertion des relations dans la matrice, on mémorise dans le vecteur LOOP les éléments qui risquent de provoquer un circuit, c'est-à-dire les éléments pour lesquels aucune nouvelle colonne n'est créée.

ex cas où il y a effectivement circuit : circuit B-C-E

	B	C	D	E	F
A	T				
B		T	T		
C				T	T
E	T				

(ex 1)

ex cas où il n'y a pas circuit bien qu'il y ait retour en arrière

	B	C	D	E	F
A	T	T			
B			T	T	
D					T
E		T			

(ex 2)

2) Procédure préliminaire

Etant donné que les éléments en lignes et en colonnes ne sont pas référencés de la même façon, nous mémorisons la correspondance dans le vecteur CORROW pour les lignes et CORCOL pour les colonnes.

a) correspondance_lignes

on mémorise dans le tableau CORROW

- le numéro d'ordre (initialisé à 0) et auquel on ajoute 1 chaque fois qu'on mémorise un nouvel élément
- chaque référence à l'index des lignes où se trouve le nom intervenant en ligne de la matrice

Pour l'exemple (ex 1) CORROW se présente de la façon suivante :

nø d'ordre	nø d'index
1	1
2	2
3	3
4	4

Si les lignes de la matrice avaient été :

INC OF process-a
TER OF process-a
INC OF process-b

nø d'ordre	nø d'index
1	1
2	1
3	2

b) correspondance_colonnes

on mémorise dans le tableau CORCOL

- le numéro d'ordre (initialisé à 0) et auquel on ajoute 1 chaque fois qu'on mémorise un nouvel élément
- chaque référence à l'index des lignes où se trouve le

nom intervenant en colonne dans la matrice.

Pour l'exemple (ex 1) CORCOL se présente de la façon suivante :

nø d'ordre	nø d'index
1	2
2	3
3	0
4	4
5	0

c) correspondance_LOOP

les noms mémorisés dans LOOP doivent eux aussi correspondre à une représentation suivant l'index qu'ils ont reçu dans la liste des lignes (vecteur correspondant = LOOPL). Cette correspondance permet de supprimer les noms qui ne sont pas repris en lignes et qui ne provoquent donc certainement pas de circuits (fin d'enchaînement - cfr (ex 2)).

3) Analyse du circuit

Pour détecter un circuit, nous avons choisi la méthode du dictionnaire du graphe, étant donné que nous n'avons pas une matrice carrée.

Pour chacun des éléments du LOOPL, on construit donc le dictionnaire du graphe dans la matrice CIRCUI.

Cette matrice se présente de la façon suivante pour l'exemple (ex 1)

nø suivant	nø niveau	nø précédent	nø imprimé
B = 2	1	B = 2	
C = 3	2	B = 2	
D = 0	2	B = 2	
E = 4	3	C = 3	JBB = 1
F = 0	3	C = 3	
B = 2	4	E = 4	

Le dictionnaire du graphe est construit par niveau :

- le Ier élément de la colonne suivant et de la colonne précédent, de niveau 1 est un élément de LOOPL.
- pour cet élément, on enregistre le ou les suivants de niveau 2 c'est-à-dire les éléments avec lesquels il est en relation directe
- pour chacun de ces éléments, on enregistre le ou les éléments de niveau 3 c'est-à-dire les éléments avec lesquels les éléments de niveau 2 sont en relation directe.

On procède de la même manière jusqu'à atteindre la fin de tous les enchaînements ou jusqu'à ce qu'on découvre une relation aboutissant à l'élément de départ (élément de LOOPL).

A ce moment, on a découvert un circuit qu'il est possible de reconstruire en reprenant dans le dictionnaire du graphe le numéro du précédant dans la colonne des suivants, à un niveau égal au niveau moins 1.

Dès qu'un circuit a été découvert, on l'imprime et on note dans la colonne des nø imprimés la colonne où se trouve la relation qui a bouclé le circuit.

On examine alors s'il est possible de prolonger l'enchaînement avec une autre relation que celle qui a fermé le circuit. Si oui, on continue de même jusqu'à découvrir un nouveau circuit, ou jusqu'à la fin de l'enchaînement.

Une fois ce travail terminé, on analyse l'élément suivant LOOPL.

A N N E X E III

DYNAMIC INTERACTION

INTRODUCTION

Cette description est une présentation générale du rapport D.I.

Elle montre :

- le but de ce rapport
- la liste des routines du programme avec leur description et la façon dont ces routines sont structurées
- la liste des routines de librairie utilisées par le programme et une brève description de ces routines
- la liste des COMMON utilisés par le programme . Cette liste est également présentée par routine (une définition détaillée de ces common fait partie de la librairie des common) .
- la liste des arguments transmis d'une routine à l'autre et leur signification .

Cette description devrait permettre à un utilisateur de comprendre le but et la structure générale de ce programme .

Dans le cadre de ce mémoire , la documentation ci-jointe montre comment Isdos peut se documenter à l'aide des outils qu'il développe .
Isdos permettrait, en outre, une description beaucoup plus détaillée de ce programme, à l'usage des informaticiens .

PRESENTATION GENERALE
=====================

1 Generalized Analyzer Version G1.0 19-Aug-79 17:25:21 Page 1
University of Namur - DEC 20
Query System

Set Name: S1
Query: ? ATTRIBUTES ARE attr-fonction report
The Number of Element is: 1
The Elements are:

Object Name	Object Type
DYNAMIC-INIERACTION-report	PROCESS

1 Generalized Analyzer Version G1.0 19-Aug-79 18:19:22 Page 1
University of Namur - DEC 20
Formatted Statements

```
1  DEFINE PROCESS                    DYNAMIC-INTERACTION-report;
2  # LAST CHANGED - 19-Aug-79     16:19:20
3      ATTRIBUTES ARE
4                    attr-fonction
5                    report;
6      DESCRIPTION;
7          purpose: this report shows the sources and results
8          -------  of dynamic actions in a matrix.
9                   It also makes an analysis of the dynamics.
10         structure :
11         ---------
12               A the report shows the objects sources of action
13                 i.e. 1 the PSL objects : EVENT
14                                          MESSAGE
15                                          PROCESS
16                                          SYNCHRONIZATION-POINT
17                      2 the state changes INCEPTION
18                                          INTERRUPTION
19                                          RESUMPTION
20                                          ABORTION
21                                          TERMINATION
22                                          GENERATION
23
24               B the report shows the objects results of action
25                 i.e. the PSL objects EVENT
26                                      MESSAGE
27                                      PROCESS
28                                      SYNCHRONIZATION-POINT
29
30               C the report also the matrix of actions:
31                 the lines are the objects sources of action
32                 the columns are the objects results of action
33                 at the intersection of a row and a column,
34                 there is a letter identifying the action between
35                 that row and that column.
36
37               D the report makes an analysis:
38                 it analyses the completeness
39                         and the consistancy ;
```

STRUCTURE GENERALE

STRUCTURE DU PROGRAMME

1 Liste des Routines du Programme

Set Name: S2
Query: ? ATTRIBUTES ARE attr-level !
The Number of Element is: 14
The Elements are:

Object Name	Object Type
prc-diadhr	PROCESS
prc-dibmat	PROCESS
prc-dicond	PROCESS
prc-difhdr	PROCESS
prc-dimess	PROCESS
prc-diprrf	PROCESS
prc-diprtc	PROCESS
prc-dismpa	PROCESS
prc-dismpr	PROCESS
prc-disort	PROCESS
prc-disum	PROCESS
prc-indi	PROCESS
prc-main	PROCESS
prc-sucors	PROCESS

2 Description des Routines du Programme


```
 1  DEFINE PROCESS                          prc-diadhr;
 2  # LAST CHANGED - 19-Aug-79     18:11:36
 3      ATTRIBUTES ARE
 4                      attr-level
 5                      quatre;
 6      DESCRIPTION;
 7                      add stuff in header matrix;
 8
 9  DEFINE PROCESS                          prc-dibmat;
10  # LAST CHANGED - 19-Aug-79     18:11:36
11      ATTRIBUTES ARE
12                      attr-level
13                      deux;
14      DESCRIPTION;
15                      builds the matrix for the next name;
16
17  DEFINE PROCESS                          prc-dicond;
18  # LAST CHANGED - 19-Aug-79     18:11:36
19      ATTRIBUTES ARE
20                      attr-level
21                      quatre;
22      DESCRIPTION;
23                      insert condition in header matrix;
24
25  DEFINE PROCESS                          prc-difhdr;
26  # LAST CHANGED - 19-Aug-79     18:11:36
27      ATTRIBUTES ARE
28                      attr-level
29                      trois;
30      DESCRIPTION;
31                      find where to put D.B. key in header matrix;
32
```

```
DEFINE PROCESS                                  prc-dimess;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    trois;
    DESCRIPTION;
                    print messages for the DI report;



DEFINE PROCESS                                  prc-diprrf;
# LAST CHANGED - 16-Aug-79      16:02:03
    ATTRIBUTES ARE
                    attr-level
                    trois;
    DESCRIPTION;
                   print rows and columns in case order different
                    of standard is required by user;



DEFINE PROCESS                                  prc-diprtc;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    trois;
    DESCRIPTION;
                    print list of conditions;



DEFINE PROCESS                                  prc-dismpa;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    trois;
    DESCRIPTION;
                    print an entire char matrix in a reasonable
                    manner-
                    this subroutine was called smprta and has
                    been adapted specifically to th DI report;



DEFINE PROCESS                                  prc-dismpr;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    quatre;
    DESCRIPTION;
                    print a table;



DEFINE PROCESS                                  prc-disort;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    trois;
    DESCRIPTION;
                     sort the columns and rows depending on the
                     order specified;



DEFINE PROCESS                                  prc-disum;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    deux;
    DESCRIPTION;
                    print matrix and make analysis;



DEFINE PROCESS                                  prc-indi;
# LAST CHANGED - 19-Aug-79      18:11:36
    ATTRIBUTES ARE
                    attr-level
                    deux;
    DESCRIPTION;
                    initialization for the dynamic interaction report;
```

```
102  DEFINE PROCESS                                  prc-main;
103  # LAST CHANGED - 19-Aug-79     18:11:36
104      ATTRIBUTES ARE
105                      attr-level
106                      un;
107      DESCRIPTION;
108                       main program for DYNAMIC-ANALYSIS;
109
110  DEFINE PROCESS                                  prc-sucors;
111  # LAST CHANGED - 19-Aug-79     18:11:36
112      ATTRIBUTES ARE
113                      attr-level
114                      trois;
115      DESCRIPTION;
116                      find successor and put in stack and matrix
117                      find condition and if any insert it in header
118                      matrix;
119
```

Number of statements printed: 42

3 Structure des Routines du Programme
++++++++++++++++++++++++++++++++++++

Analyzer Utility Program

Structure Report

Parameters: DB=Unknown NAME=prc-main OBJECTS=PROCESS RELATIONS=ON-RES-TRIGGERS INDENT=3
NOINDEX NOPUNCHED-NAMES LEVELS=ALL TYPES-MARGIN=60 RELATIONS-MARGIN=75 LINE-NUMBERS
LEVEL-NUMBERS STATISTICS NONEW-PAGE RELATIONS-MATRIX NOEXPLANATION NAME-TYPES NOUPWARDS
DOWNWARDS ROW-ORDER=STANDARD COLUMN-ORDER=STANDARD NOBOTTOM-LEVEL-ONLY NOTOP-LEVEL-ONLY
COMPRESS-RELATION-MATRIX

Structure

1	1	prc-main	PROCESS	
2	2	prc-indi	PROCESS	(TRGRED-BY-RES)
3	2	prc-dibmat	PROCESS	(TRGRED-BY-RES)
4	3	prc-difhdr	PROCESS	(TRGRED-BY-RES)
5	3	prc-sucors	PROCESS	(TRGRED-BY-RES)
6	4	prc-diadhr	PROCESS	(TRGRED-BY-RES)
7	4	prc-dicond	PROCESS	(TRGRED-BY-RES)
8	2	prc-disum	PROCESS	(TRGRED-BY-RES)
9	3	prc-diprtc	PROCESS	(TRGRED-BY-RES)
10	3	prc-dimess	PROCESS	(TRGRED-BY-RES)
11	3	prc-dismpa	PROCESS	(TRGRED-BY-RES)
12	4	prc-dismpr	PROCESS	(TRGRED-BY-RES)
13	3	prc-disort	PROCESS	(TRGRED-BY-RES)
14	3	prc-diprf	PROCESS	(TRGRED-BY-RES)

STATISTICS:

LEVEL	COUNT	LEVEL	COUNT	LEVEL	COUNT	LEVEL	COUNT
1	1	2	3	3	7	4	3

4 Enchainement Dynamique des Routines du Programme

Analyzer Utility Program

Dynamic Interaction Report

Parameters: DB=Unknown NAME=prc-main DYNAMIC-INTERACTION-MATRIX NODYNAMIC-ANALYSIS NOUTILIZES NOEXPLANATION LINKS=1000 ROW-COLUMN-ORDER=BYTYPE

	Row Names	Order	
1	msg-analysis	0	MESSAGE
2	msg-column-names	0	MESSAGE
3	msg-list-conditions	0	MESSAGE
4	msg-rows-list	0	MESSAGE
5	prc-diadhr	28	PROCESS
6	prc-dibmat	19	PROCESS
7	prc-dicond	26	PROCESS
8	prc-difndr	30	PROCESS
9	prc-dimess	14	PROCESS
10	prc-diprf	0	PROCESS
11	prc-diprtc	16	PROCESS
12	prc-dismpa	11	PROCESS
13	prc-dismpr	12	PROCESS
14	prc-disort	9	PROCESS
15	prc-disum	3	PROCESS
16	prc-indi	32	PROCESS
17	prc-main	1	PROCESS
18	prc-sucors	22	PROCESS

	Column Names	Order	
1	msg-analysis	6	MESSAGE
2	msg-column-names	5	MESSAGE
3	msg-list-conditions	14	MESSAGE
4	msg-rows-list	4	MESSAGE
5	prc-diadhr	17	PROCESS
6	prc-dibmat	2	PROCESS
7	prc-dicond	18	PROCESS
8	prc-difndr	15	PROCESS
9	prc-dimess	9	PROCESS
10	prc-diprf	12	PROCESS
11	prc-diprtc	8	PROCESS
12	prc-dismpa	10	PROCESS
13	prc-dismpr	13	PROCESS
14	prc-disort	11	PROCESS
15	prc-disum	3	PROCESS
16	prc-indi	1	PROCESS
17	prc-main	7	PROCESS
18	prc-sucors	16	PROCESS

CONDITION NAMES

1	cond-abortsw	CONDITION
2	cond-numhan-positif	CONDITION
3	cond-order-standard	CONDITION
4	cond-inssw	CONDITION
5	cond-conditional-action	CONDITION

Analyzer Utility Program

Dynamic Interaction Report

DYNAMIC INTERACTION MATRIX

```
                                                
```

STRUCTURE DU PROGRAMME ET DE SON ENVIRONNEMENT
****

1 Liste des Routines de Librairies Employees par le Programme
++

Set Name: S3
Query: ? ATTRIBUTES ARE attr-library !
The Number of Element is: 53
The Elements are:

Object Name	Object Type
prc-abparm	PROCESS
prc-acparm	PROCESS
prc-aeparm	PROCESS
prc-aiparm	PROCESS
prc-aktns3	PROCESS
prc-aktsr3	PROCESS
prc-alparm	PROCESS
prc-cldi	PROCESS
prc-enparm	PROCESS
prc-fnish	PROCESS
prc-fnm	PROCESS
prc-fnsk	PROCESS
prc-gfk	PROCESS
prc-gkm	PROCESS
prc-gko	PROCESS
prc-hyusrn	PROCESS
prc-inpar	PROCESS
prc-inparm	PROCESS
prc-irginf	PROCESS
prc-namlen	PROCESS
prc-newgkb	PROCESS
prc-nexnam	PROCESS
prc-otpar	PROCESS
prc-plong	PROCESS
prc-prtcol	PROCESS
prc-prterr	PROCESS
prc-rqbada	PROCESS
prc-rqbadi	PROCESS
prc-rqbadn	PROCESS
prc-rqbads	PROCESS
prc-rqbblk	PROCESS
prc-rqbotp	PROCESS
prc-rqbprt	PROCESS
prc-rqcads	PROCESS
prc-rqcblk	PROCESS
prc-rqcpar	PROCESS
prc-rqhead	PROCESS
prc-rqnewp	PROCESS
prc-skfm	PROCESS
prc-smcnxt	PROCESS
prc-smdtbl	PROCESS
prc-smexmp	PROCESS
prc-smget	PROCESS
prc-smk	PROCESS
prc-smlook	PROCESS
prc-smm	PROCESS
prc-smnext	PROCESS
prc-smnrow	PROCESS
prc-smrnxt	PROCESS
prc-smset	PROCESS
prc-smtint	PROCESS
prc-som	PROCESS
prc-typcnv	PROCESS

2 Presentation de ces Routines par Librairie

1 Generalized Analyzer Version G1.0 23-Aug-79 10:56:15 Page 1
University of Namur - DEC 20
Query System

Set Name: S1
Query: ? ATTRIBUTES ARE attr-library adbms
The Number of Element is: 9
The Elements are:

Object Name	Object Type
prc-fnm	PROCESS
prc-fnsk	PROCESS
prc-gfk	PROCESS
prc-gkm	PROCESS
prc-gko	PROCESS
prc-skfm	PROCESS
prc-smk	PROCESS
prc-smm	PROCESS
prc-som	PROCESS

1 Generalized Analyzer Version G1.0 23-Aug-79 10:56:15 Page 2
University of Namur - DEC 20
Query System

Set Name: S2
Query: ? ATTRIBUTES ARE attr-library cllib
The Number of Element is: 1
The Elements are:

Object Name	Object Type
prc-cldi	PROCESS

Set Name: S3
Query: ? ATTRIBUTES ARE attr-library flib
The Number of Element is: 13
The Elements are:

Object Name	Object Type
prc-aktns3	PROCESS
prc-aktsr3	PROCESS
prc-fnish	PROCESS
prc-hyusrn	PROCESS
prc-inpar	PROCESS
prc-namlen	PROCESS
prc-newgkb	PROCESS
prc-nexnam	PROCESS
prc-otpar	PROCESS
prc-plong	PROCESS
prc-prtcol	PROCESS
prc-prterr	PROCESS
prc-typcnv	PROCESS

Set Name: S4
Query: ? ATTRIBUTES ARE attr-library parmlib
The Number of Element is: 7
The Elements are:

Object Name	Object Type
prc-abparm	PROCESS
prc-acparm	PROCESS
prc-aeparm	PROCESS
prc-aiparm	PROCESS
prc-alparm	PROCESS
prc-enparm	PROCESS
prc-inparm	PROCESS

Set Name: S5
Query: ? ATTRIBUTES ARE attr-library rqlib
The Number of Element is: 13
The Elements are:

Object Name	Object Type
prc-irginf	PROCESS
prc-rqbada	PROCESS
prc-rqbadi	PROCESS
prc-rqbadn	PROCESS
prc-rqbads	PROCESS
prc-rqbblk	PROCESS
prc-rqbotp	PROCESS
prc-rqbprt	PROCESS
prc-rqcads	PROCESS
prc-rqcblk	PROCESS
prc-rqcpar	PROCESS
prc-rqhead	PROCESS
prc-rqnewp	PROCESS

Set Name: S6
Query: ? ATTRIBUTES ARE attr-library smlib
The Number of Element is: 10
The Elements are:

Object Name	Object Type
prc-smcnxt	PROCESS
prc-smdtbl	PROCESS
prc-smexmp	PROCESS
prc-smget	PROCESS
prc-smlook	PROCESS
prc-smnext	PROCESS
prc-smnrow	PROCESS
prc-smrnxt	PROCESS
prc-smset	PROCESS
prc-smtint	PROCESS

3 Description des Routines de Librairie par Ordre Alphabetique
++

Formatted Statements

```
1  DEFINE PROCESS                                  prc-abparm;
2  # LAST CHANGED - 14-Aug-79     15:04:42
3      ATTRIBUTES ARE
4                      attr-library
5                      parmlib;
6      DESCRIPTION;
7                       add DB=DBNMNM to parameter output;
8
```

```
9  DEFINE PROCESS                                  prc-acparm;
10 # LAST CHANGED - 14-Aug-79     15:04:42
11     ATTRIBUTES ARE
12                     attr-library
13                     parmlib;
14     DESCRIPTION;
15                      add CH=string to parameter output;
16
```

```
17 DEFINE PROCESS                                  prc-aeparm;
18 # LAST CHANGED - 14-Aug-79     15:04:42
19     ATTRIBUTES ARE
20                     attr-library
21                     parmlib;
22     DESCRIPTION;
23                      handle either NAME=NAM or FILE=FILENM;
24
```

```
25 DEFINE PROCESS                                  prc-aiparm;
26 # LAST CHANGED - 14-Aug-79     15:04:42
27     ATTRIBUTES ARE
28                     attr-library
29                     parmlib;
30     DESCRIPTION;
31                     add CH=integer to parameter output;
32
```

```
33 DEFINE PROCESS                                  prc-aktns3;
34 # LAST CHANGED - 16-Aug-79     16:02:03
35     ATTRIBUTES ARE
36                     attr-library
37                     flib;
38     DESCRIPTION;
39                     sort the array V and attaches the tag vector T
40                     ascending order by type, then by name;
41
```

```
42 DEFINE PROCESS                                  prc-aktsr3;
43 # LAST CHANGED - 16-Aug-79     16:02:03
44     ATTRIBUTES ARE
45                     attr-library
46                     flib;
47     DESCRIPTION;
48                     sorts the array V and attaches the tag vector T
49                     ascending order by type, maintaining order if
50                     same type;
51
```

```
52 DEFINE PROCESS                                  prc-alparm;
53 # LAST CHANGED - 14-Aug-79     15:04:42
54     ATTRIBUTES ARE
55                     attr-library
56                     parmlib;
57     DESCRIPTION;
58                     add (no)CH to parameter output;
59
```

```
60 DEFINE PROCESS                                  prc-cldi;
61 # LAST CHANGED - 14-Aug-79     14:54:38
62     ATTRIBUTES ARE
63                     attr-library
64                     cllib;
65     DESCRIPTION;
66                     dynamic interaction report - parameters;
67
```

```
DEFINE PROCESS                                  prc-enparm;
# LAST CHANGED - 14-Aug-79     15:04:42
    ATTRIBUTES ARE
                    attr-library
                    parmlib;
    DESCRIPTION;
                    terminate parameter output;



DEFINE PROCESS                                  prc-fnish;
# LAST CHANGED - 14-Aug-79     14:54:38
    ATTRIBUTES ARE
                    attr-library
                    flib;
    DESCRIPTION;
                     subroutine to close data base and do any
                     other stuff;



DEFINE PROCESS                                  prc-fnm;
# LAST CHANGED - 14-Aug-79     19:07:32
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    find the next member of a set;



DEFINE PROCESS                                  prc-fnsk;
# LAST CHANGED - 14-Aug-79     19:07:32
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    find next member of a sorted set based on
                    sort key;
```

```
DEFINE PROCESS                                  prc-gfk;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    get field from a record based on key;



DEFINE PROCESS                                  prc-gkm;
# LAST CHANGED - 16-Aug-79     14:22:45
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    get key of current member of a set;



DEFINE PROCESS                                  prc-gko;
# LAST CHANGED - 14-Aug-79     19:07:32
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    get key of current owner of a set;



DEFINE PROCESS                                  prc-hyusrn;
# LAST CHANGED - 16-Aug-79     14:55:53
    ATTRIBUTES ARE
                    attr-library
                    flib;
    DESCRIPTION;
                    give an error comment to user (at terminal);
```

```
134  DEFINE PROCESS                                  prc-inpar;
135  # LAST CHANGED - 14-Aug-79     14:54:38
136      ATTRIBUTES ARE
137                      attr-library
138                      flib;
139      DESCRIPTION;
140                      standard initialization stuff;
141
142  DEFINE PROCESS                                  prc-inparm;
143  # LAST CHANGED - 14-Aug-79     15:04:42
144      ATTRIBUTES ARE
145                      attr-library
146                      parmlib;
147      DESCRIPTION;
148                       initialize parameter routines;
149
150  DEFINE PROCESS                                  prc-irqinf;
151  # LAST CHANGED - 16-Aug-79     18:05:38
152      ATTRIBUTES ARE
153                      attr-library
154                      rqlib;
155      DESCRIPTION;
156                      obtain information about printer output;
157
158  DEFINE PROCESS                                  prc-namlen;
159  # LAST CHANGED - 17-Aug-79     16:17:08
160      ATTRIBUTES ARE
161                      attr-library
162                      flib;
163      DESCRIPTION;
164                      figure out length of name which has been padded
165                      with blanks;
166
167  DEFINE PROCESS                                  prc-newgkb;
168  # LAST CHANGED - 14-Aug-79     14:54:38
169      ATTRIBUTES ARE
170                      attr-library
171                      flib;
172      DESCRIPTION;
173                       find basic name, check for allowable types,
174                       and make synonyms transparent;
175
176  DEFINE PROCESS                                  prc-nexnam;
177  # LAST CHANGED - 14-Aug-79     14:54:38
178      ATTRIBUTES ARE
179                      attr-library
180                      flib;
181      DESCRIPTION;
182                       get next name from file if readsw otherwise
183                       nam already has it;
184
185  DEFINE PROCESS                                  prc-otpar;
186  # LAST CHANGED - 14-Aug-79     14:54:38
187      ATTRIBUTES ARE
188                      attr-library
189                      flib;
190      DESCRIPTION;
191                       write standard initialization stuff;
192
193  DEFINE PROCESS                                  prc-plong;
194  # LAST CHANGED - 16-Aug-79     14:55:53
195      ATTRIBUTES ARE
196                      attr-library
197                      flib;
198      DESCRIPTION;
199                       obtain from a file and print a several line comment;
200
```

```
DEFINE PROCESS                                  prc-prtcol;
# LAST CHANGED - 16-Aug-79     15:39:03
    ATTRIBUTES ARE
                    attr-library
                    flib;
    DESCRIPTION;
                print column of names and types for matrices
                longer list is printed on left;


DEFINE PROCESS                                  prc-prterr;
# LAST CHANGED - 16-Aug-79     14:55:53
    ATTRIBUTES ARE
                    attr-library
                    flib;
    DESCRIPTION;
                    print an error on output report;


DEFINE PROCESS                                  prc-rqbada;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                     add stuff to current buffer;

DEFINE PROCESS                                  prc-rqbadi;
# LAST CHANGED - 16-Aug-79     17:52:45
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    put an integer into the output buffer;


DEFINE PROCESS                                  prc-rqbadn;
# LAST CHANGED - 16-Aug-79     17:42:13
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    add a name to the output buffer;


DEFINE PROCESS                                  prc-rqbads;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    add string to buffer;


DEFINE PROCESS                                  prc-rqbblk;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                     blank line buffer;


DEFINE PROCESS                                  prc-rqbotp;
# LAST CHANGED - 16-Aug-79     18:05:38
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    set pointers to bottom of page, so next write
                    will cause page skip;
```

```
DEFINE PROCESS                                  prc-rqbprt;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    pritn current buffer;



DEFINE PROCESS                                  prc-rqcads;
# LAST CHANGED - 16-Aug-79     15:19:52
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    add stuff to column header;



DEFINE PROCESS                                  prc-rqcblk;
# LAST CHANGED - 16-Aug-79     15:19:52
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                     blank out column heading;



DEFINE PROCESS                                  prc-rqcpar;
# LAST CHANGED - 16-Aug-79     15:39:03
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    ATTRIBUTES ARE
                    attr-belonging
                    disum;
    DESCRIPTION;

                 give info for column heading;



DEFINE PROCESS                                  prc-rqhead;
# LAST CHANGED - 14-Aug-79     15:04:42
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    supply header for this report;



DEFINE PROCESS                                  prc-rqnewp;
# LAST CHANGED - 16-Aug-79     15:19:52
    ATTRIBUTES ARE
                    attr-library
                    rqlib;
    DESCRIPTION;
                    skip to top of page, print heading etc. if less
                    than given number of lines left;



DEFINE PROCESS                                  prc-skfm;
# LAST CHANGED - 16-Aug-79     14:22:45
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    set owner based on key and find member;



DEFINE PROCESS                                  prc-smcnxt;
# LAST CHANGED - 16-Aug-79     14:55:53
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                     find next element in column;
```

```
DEFINE PROCESS                                  prc-smdtbl;
# LAST CHANGED - 16-Aug-79      15:19:52
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                    delete a complete table;



DEFINE PROCESS                                  prc-smexmp;
# LAST CHANGED - 17-Aug-79      16:17:08
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                     construct empty section message;



DEFINE PROCESS                                  prc-smget;
# LAST CHANGED - 17-Aug-79      16:17:08
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                     get the value for N(i,j);



DEFINE PROCESS                                  prc-smk;
# LAST CHANGED - 16-Aug-79      14:22:45
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    set the current member of a set based on key;



DEFINE PROCESS                                  prc-smlook;
# LAST CHANGED - 14-Aug-79      18:55:52
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                    routine to look for a value in a sparse matrix;



DEFINE PROCESS                                  prc-smm;
# LAST CHANGED - 16-Aug-79      14:22:45
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    set the current member of a set based on the
                    current member;



DEFINE PROCESS                                  prc-smnext;
# LAST CHANGED - 17-Aug-79      16:17:08
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                    find the next element in a table;



DEFINE PROCESS                                  prc-smnrow;
# LAST CHANGED - 16-Aug-79      14:22:45
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                    count number of elements in row;
```

```
DEFINE PROCESS                                  prc-smrnxt;
# LAST CHANGED - 16-Aug-79     15:19:52
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                    find next element in row;



DEFINE PROCESS                                  prc-smset;
# LAST CHANGED - 16-Aug-79     14:55:53
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                    set a value in the table;



DEFINE PROCESS                                  prc-smtint;
# LAST CHANGED - 16-Aug-79     14:55:53
    ATTRIBUTES ARE
                    attr-library
                    smlib;
    DESCRIPTION;
                     allocate a new table;



DEFINE PROCESS                                  prc-som;
# LAST CHANGED - 16-Aug-79     14:22:45
    ATTRIBUTES ARE
                    attr-library
                    adbms;
    DESCRIPTION;
                    set the current owner of a set based on the current
                    member of a set;


DEFINE PROCESS                                  prc-typcnv;
# LAST CHANGED - 16-Aug-79     17:52:45
    ATTRIBUTES ARE
                    attr-library
                    flib;
    DESCRIPTION;
                    supply a printable version of object type;


Number of statements printed: 160
```

3 Enchainement Dynamique du Programme et de son Environnement
++

Analyzer Utility Program

Dynamic Interaction Report

Parameters: DB=Unknown NAME=prc-main DYNAMIC-INTERACTION-MATRIX NODYNAMIC-ANALYSIS UTILIZES NOEXPLANATION LINKS=1000 ROW-COLUMN-ORDER=BYTYPE

	Row Names	Order	
1	msg-analysis	0	MESSAGE
2	msg-column-names	0	MESSAGE
3	msg-list-conditions	0	MESSAGE
4	msg-rows-list	0	MESSAGE
5	prc-abparm	0	PROCESS
6	prc-acparm	0	PROCESS
7	prc-aeparm	0	PROCESS
8	prc-aiparm	0	PROCESS
9	prc-aktns3	0	PROCESS
10	prc-aktsr3	0	PROCESS
11	prc-alparm	0	PROCESS
12	prc-cldi	0	PROCESS
13	prc-diadhr	36	PROCESS
14	prc-dibmat	24	PROCESS
15	prc-dicond	33	PROCESS
16	prc-difhdr	39	PROCESS
17	prc-dimess	18	PROCESS
18	prc-diprf	0	PROCESS
19	prc-diprtc	21	PROCESS
20	prc-dismpa	13	PROCESS
21	prc-dismpr	15	PROCESS
22	prc-disort	10	PROCESS
23	prc-disum	4	PROCESS
24	prc-emparm	0	PROCESS
25	prc-fnish	0	PROCESS
26	prc-fnm	0	PROCESS
27	prc-fnsk	0	PROCESS
28	prc-gbinit	0	PROCESS
29	prc-gbterm	0	PROCESS
30	prc-gfk	0	PROCESS
31	prc-gkm	0	PROCESS
32	prc-gko	0	PROCESS
33	prc-hyursn	0	PROCESS
34	prc-indi	42	PROCESS
35	prc-inpar	0	PROCESS

Analyzer Utility Program

Dynamic Interaction Report

	Row Names	Order	
36	prc-inparm	0	PROCESS
37	prc-irginf	0	PROCESS
38	prc-ivlook	0	PROCESS
39	prc-main	1	PROCESS
40	prc-namlen	0	PROCESS
41	prc-newgkb	0	PROCESS
42	prc-nexnam	0	PROCESS
43	prc-otpar	0	PROCESS
44	prc-plong	0	PROCESS
45	prc-prtcol	0	PROCESS
46	prc-prterr	0	PROCESS
47	prc-rqbada	0	PROCESS
48	prc-rqbadi	0	PROCESS
49	prc-rqbadn	0	PROCESS
50	prc-rqbads	0	PROCESS
51	prc-rqbblk	0	PROCESS
52	prc-rqbotp	0	PROCESS
53	prc-rqbprt	0	PROCESS
54	prc-rqcblk	0	PROCESS
55	prc-rqcpar	0	PROCESS
56	prc-rqhead	0	PROCESS
57	prc-skfm	0	PROCESS
58	prc-smdtbl	0	PROCESS
59	prc-smexmp	0	PROCESS
60	prc-smget	0	PROCESS
61	prc-sminit	0	PROCESS
62	prc-smlook	0	PROCESS
63	prc-smm	0	PROCESS
64	prc-smnext	0	PROCESS
65	prc-smnrow	0	PROCESS
66	prc-smove	0	PROCESS
67	prc-smrnxt	0	PROCESS
68	prc-smset	0	PROCESS
69	prc-smtint	0	PROCESS
70	prc-som	0	PROCESS
71	prc-sucors	28	PROCESS
72	prc-typcnv	0	PROCESS

Analyzer Utility Program

Dynamic Interaction Report

	Column Names	Order	
1	msg-analysis	15	MESSAGE
2	msg-column-names	14	MESSAGE
3	msg-list-conditions	49	MESSAGE
4	msg-rows-list	13	MESSAGE
5	prc-abparm	72	PROCESS
6	prc-acparm	71	PROCESS
7	prc-aeparm	66	PROCESS
8	prc-aiparm	68	PROCESS
9	prc-aktns3	38	PROCESS
10	prc-aktsr3	39	PROCESS
11	prc-alparm	65	PROCESS
12	prc-cldi	12	PROCESS
13	prc-diadhr	53	PROCESS
14	prc-dibmat	2	PROCESS
15	prc-dicond	54	PROCESS
16	prc-difhdr	51	PROCESS
17	prc-dimess	18	PROCESS
18	prc-diprf	21	PROCESS
19	prc-diprtc	17	PROCESS
20	prc-dismpa	19	PROCESS
21	prc-dismpr	40	PROCESS
22	prc-disort	20	PROCESS
23	prc-disum	3	PROCESS
24	prc-emparm	67	PROCESS
25	prc-fnish	10	PROCESS
26	prc-fnm	56	PROCESS
27	prc-fnsk	57	PROCESS
28	prc-gbinit	11	PROCESS
29	prc-gbterm	5	PROCESS
30	prc-gfk	34	PROCESS
31	prc-gkm	59	PROCESS
32	prc-gko	55	PROCESS
33	prc-nyursn	25	PROCESS
34	prc-indi	1	PROCESS
35	prc-inpar	8	PROCESS
36	prc-inparm	64	PROCESS
37	prc-irginf	41	PROCESS
38	prc-ivlook	22	PROCESS

Analyzer Utility Program

Dynamic Interaction Report

	Column Names	Order	
39	prc-main	16	PROCESS
40	prc-namlen	46	PROCESS
41	prc-newgkb	4	PROCESS
42	prc-nexnam	7	PROCESS
43	prc-otpar	6	PROCESS
44	prc-plong	28	PROCESS
45	prc-prtcol	29	PROCESS
46	prc-prterr	9	PROCESS
47	prc-rqbada	42	PROCESS
48	prc-rqbadi	47	PROCESS
49	prc-rqbadn	50	PROCESS
50	prc-rqbads	27	PROCESS
51	prc-rqbblk	45	PROCESS
52	prc-rqbotp	23	PROCESS
53	prc-rqbprt	31	PROCESS
54	prc-rqcblk	36	PROCESS
55	prc-rqcpar	37	PROCESS
56	prc-rqhead	70	PROCESS
57	prc-skfm	60	PROCESS
58	prc-smdtbl	33	PROCESS
59	prc-smexmp	44	PROCESS
60	prc-smget	35	PROCESS
61	prc-sminit	69	PROCESS
62	prc-smlook	63	PROCESS
63	prc-smm	62	PROCESS
64	prc-smnext	26	PROCESS
65	prc-smnrow	58	PROCESS
66	prc-smove	43	PROCESS
67	prc-smrnxt	30	PROCESS
68	prc-smset	24	PROCESS
69	prc-smtint	32	PROCESS
70	prc-som	61	PROCESS
71	prc-sucors	52	PROCESS
72	prc-typcnv	48	PROCESS

CONDITION NAMES

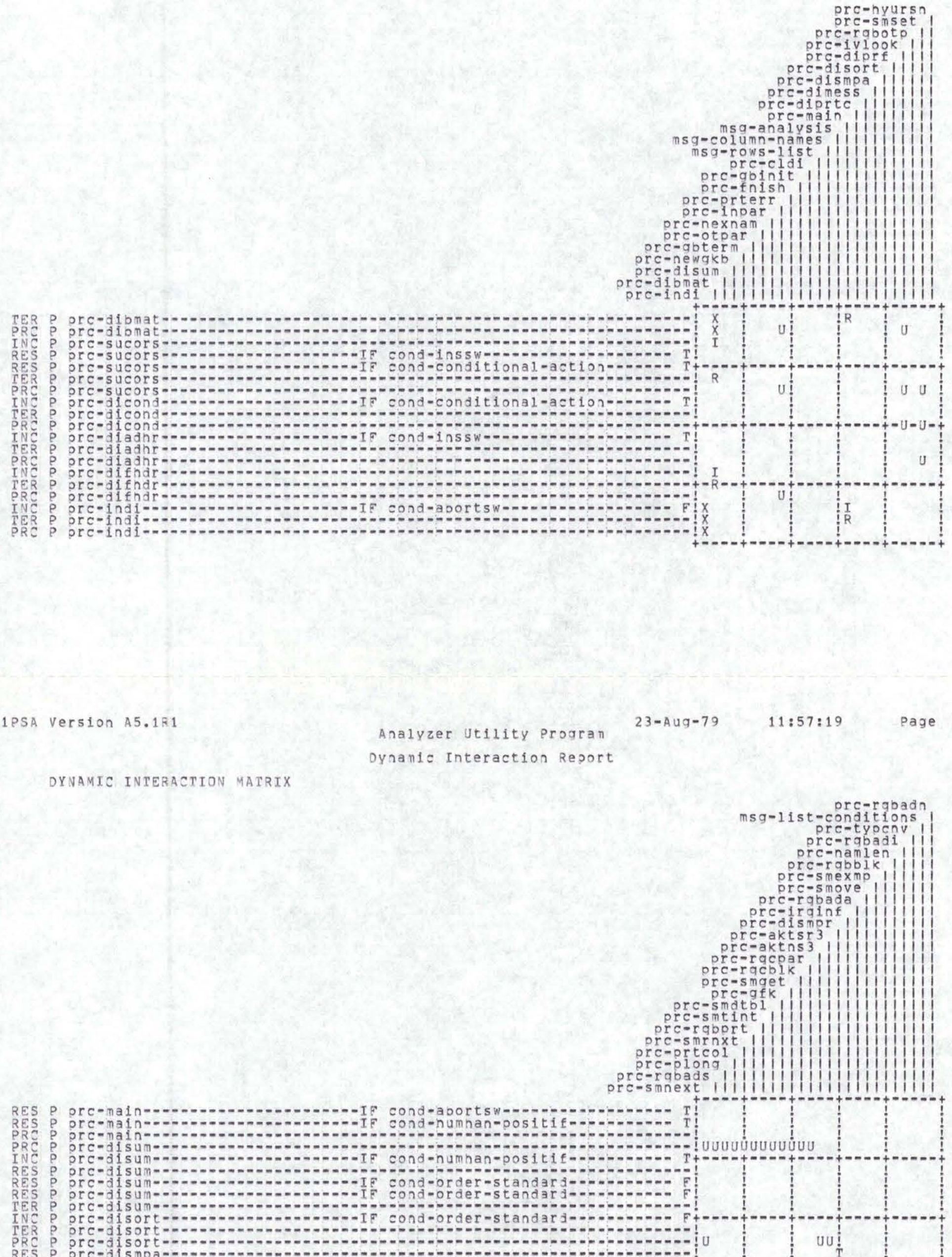
1	cond-abortsw	CONDITION
2	cond-numhan-positif	CONDITION
3	cond-order-standard	CONDITION
4	cond-inssw	CONDITION
5	cond-conditional-action	CONDITION

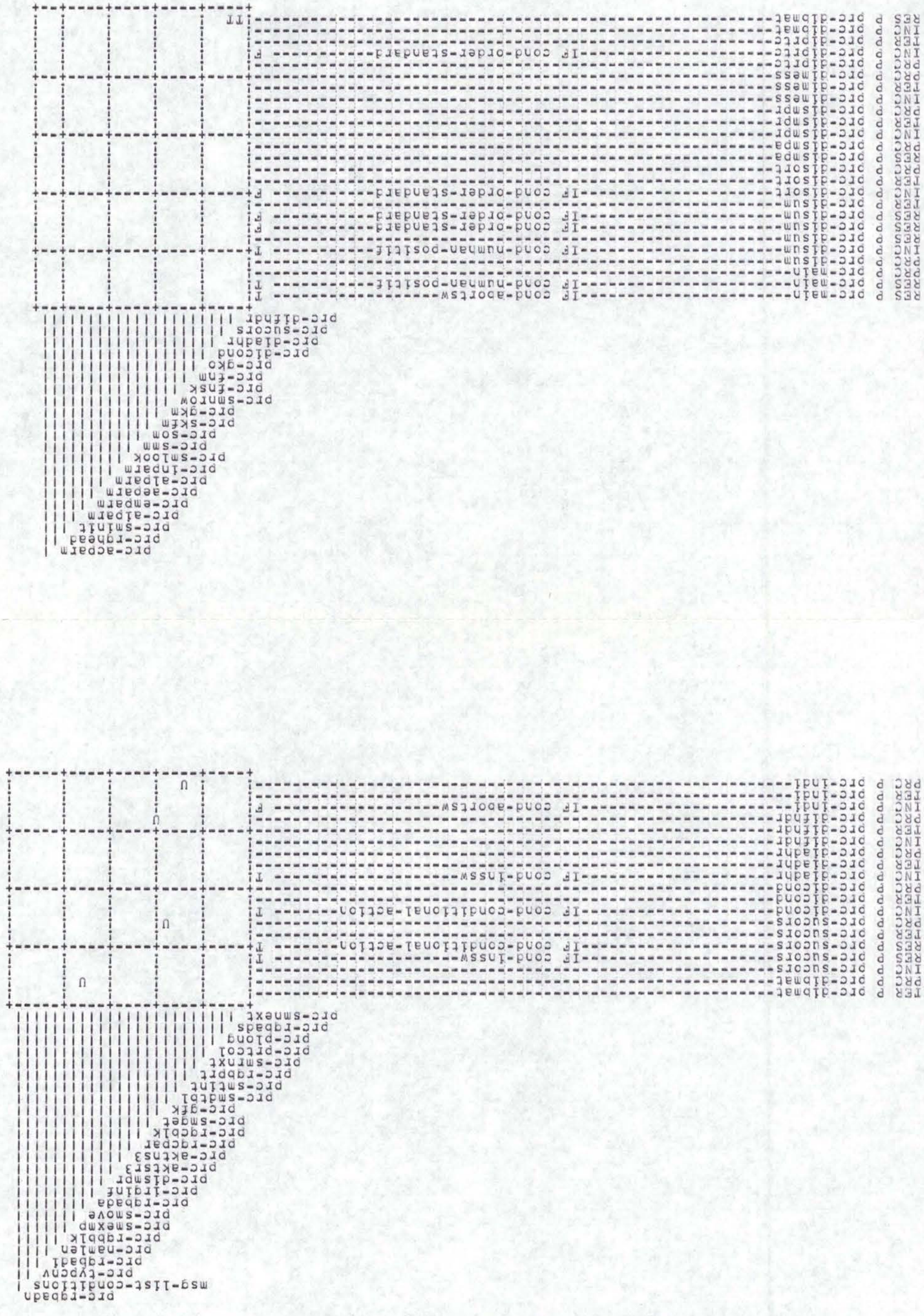
DYNAMIC INTERACTION MATRIX

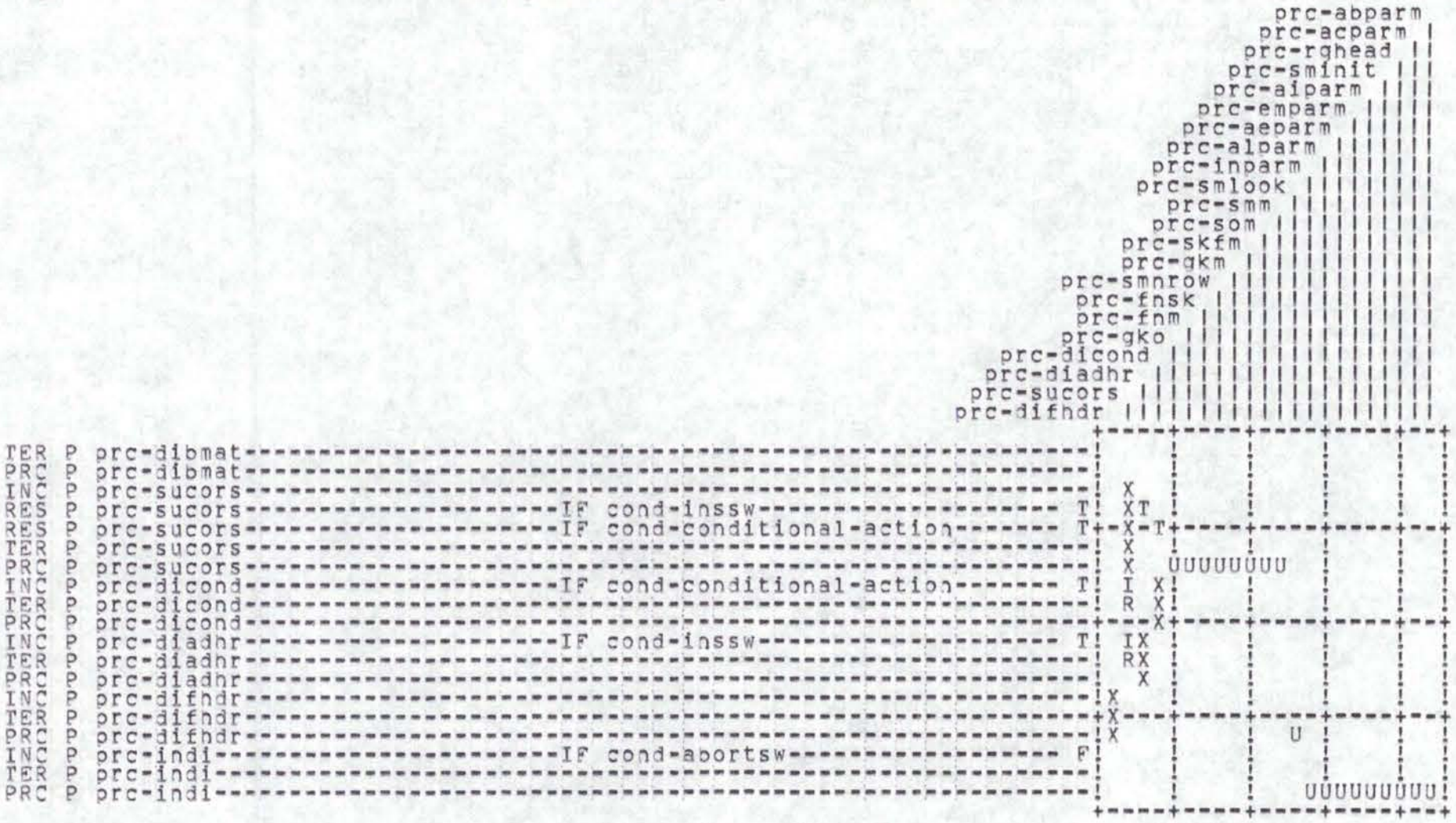
Analyzer Utility Program

Dynamic Interaction Report

DYNAMIC INTERACTION MATRIX




pic-acpairm
pic-rqhead
pic-smlnlt
pic-ajpairm
pic-empairm
pic-aepairm
pic-alpairm
pic-inpairm
pic-smlook
pic-smam
pic-som
pic-skfim
pic-qkom
pic-smrtow
pic-fnsk
pic-fnm
pic-qko
pic-dlcond
pic-dladhr
pic-succors
pic-dlfndr
msg-list-conditions
pic-typcnv
pic-rqbadl
pic-namlen
pic-rqbblk
pic-smexmp
pic-smove
pic-rqbada
pic-lrqlnf
pic-dismpr
pic-aktsr3
pic-aktns3
pic-rqcpar
pic-rqcblk
pic-smget
pic-gfk
pic-smdtbl
pic-smtlnt
pic-rqbprt
pic-smrnxt
pic-prtcol
pic-plong
pic-rqbads
pic-smnext
cond-ordering-standard
cond-numnan-positif
cond-abortsw
cond-insw
cond-conditional-action
pic-dlbmat
pic-indl
pic-dlfndr
pic-dladhr
pic-dlcond
pic-succors
pic-dlsmpr
pic-dlmess
pic-dlprtc
pic-dlsort
pic-dlsum
pic-dlsmpa
pic-dlsmpr
pic-dlmess
pic-dlprtc
pic-dlmess
pic-dlprtc
pic-dlbmat
pic-dlbmat



UTILISATION DES VARIABLES
==========

COMMON UTILISES DANS LE PROGRAMME
****

1 Liste des Common
++++++++++++++++

Set Name: S4
Query: ? ATTRIBUTES ARE attr-statut common
The Number of Element is: 20
The Elements are:

Object Name	Object Type
set-blanks	SET
set-cmnmod	SET
set-colkey	SET
set-condky	SET
set-dicm	SET
set-didefn	SET
set-dihdr	SET
set-diin	SET
set-epstak	SET
set-evcode	SET
set-loopng	SET
set-name	SET
set-parin	SET
set-prtbuf	SET
set-psagcm	SET
set-refrce	SET
set-rltcod	SET
set-rowkey	SET
set-srtvec	SET
set-typcom	SET

2 Description des Common
++++++++++++++++++++

Formatted Statements

```
1  DEFINE SET                                   set-blanks;
2  # LAST CHANGED - 17-Aug-79     16:17:08
3      ATTRIBUTES ARE
4                    attr-statut
5                    common;
6      DESCRIPTION;
7                    contains blanks for various purposes,
8                    initialized by block data;
9      CONSISTS OF   el-blank;
10
11 DEFINE SET                                   set-cmnmod;
12 # LAST CHANGED - 13-Aug-79     21:53:34
13     ATTRIBUTES ARE
14                   attr-statut
15                   common;
16     DESCRIPTION;
17                    hold some useful names;
18     SUBSET OF     set-main;
19     CONSISTS OF   el-cmdnam,
20                   el-modnam;
21
22 DEFINE SET                                   set-colkey;
23 # LAST CHANGED - 17-Aug-79     16:17:08
24     ATTRIBUTES ARE
25                   attr-statut
26                   common;
27     DESCRIPTION;
28                   storage for keys of columns of matrices;
29     SUBSET OF     set-indi,
30                   set-sucors,
31                   set-disum,
32                   set-disort,
33                   set-diprrf,
34                   set-dismpr;
35     CONSISTS OF   el-ncol,
36                   el-col;
37
38 DEFINE SET                                   set-condky;
39 # LAST CHANGED - 17-Aug-79     16:17:08
40     ATTRIBUTES ARE
41                   attr-statut
42                   common;
43     DESCRIPTION;
44                   storage for keys of conditions of matrices;
45     SUBSET OF     set-indi,
46                   set-dicond,
47                   set-disum,
48                   set-diprtc,
49                   set-dismpr;
50     CONSISTS OF   el-ncond,
51                   el-cond;
52
53 DEFINE SET                                   set-dicm;
54 # LAST CHANGED - 16-Aug-79     17:52:45
55     ATTRIBUTES ARE
56                   attr-statut
57                   common;
58     DESCRIPTION;
59                    whether to print header on output;
60     SUBSET OF     set-disum,
61                   set-dimess;
62     CONSISTS OF   el-hed1sw,
63                   el-hed2sw,
64                   el-hed3sw;
65
```

```
DEFINE SEI                                      set-didefn;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    define matrices used by the program;
    SUBSEI OF       set-indi,
                    set-difhdr,
                    set-sucors,
                    set-diadhr,
                    set-dicond,
                    set-disum,
                    set-disort,
                    set-dismpr;
    CONSISTS OF     el-mat1,
                    el-mat2,
                    el-mat3,
                    el-mat4,
                    el-mat5;


DEFINE SEI                                      set-dihdr;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    information about header matrix;
    SUBSEI OF       set-indi,
                    set-dibmat,
                    set-difhdr,
                    set-sucors,
                    set-diadhr,
                    set-dicond,
                    set-disum,
                    set-disort,
                    set-dismpr;
    CONSISTS OF     el-nhdr,
                    el-idxhdr;


DEFINE SEI                                      set-diin;
# LAST CHANGED - 16-Aug-79     16:02:03
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                     hold switches set by parameters given to
                      the program;
    SUBSEI OF       set-main,
                    set-indi,
                    set-dibmat,
                    set-disum,
                    set-disort;
    CONSISTS OF     el-dimtsw,
                    el-danlsw,
                    el-order,
                    el-links,
                    el-utilsw,
                    el-rcosw;


DEFINE SEI                                      set-epstak;
# LAST CHANGED - 16-Aug-79     14:29:37
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    stacks for use in extended picture report;
    SUBSEI OF       set-dibmat,
                    set-sucors;
    CONSISTS OF     el-staktp,
                    el-lastpt,
                    el-stack,
                    el-typstk,
                    el-rlnstk,
                    el-stakpt,
                    el-ovflsw;
```

```
DEFINE SET                                      set-evcode;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    define every type of event(sensu lato) possible
                    in the version NAMJR;
    SUBSET OF       set-dibmat,
                    set-difhdr,
                    set-sucors,
                    set-diadhr,
                    set-dismpr;
    CONSISTS OF     el-evgen,
                    el-evpess,
                    el-evevt,
                    el-evsyn,
                    el-evmsg,
                    el-evinc,
                    el-evint,
                    el-evres,
                    el-evabr,
                    el-evterm,
                    el-evtyp,
                    el-nmevnt,
                    el-ievnt,
                    el-ievcod;


DEFINE SET                                      set-loopng;
# LAST CHANGED - 16-Aug-79     15:39:03
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    storage for keys of elements that might be in a
                    circuit;
    SUBSET OF       set-sucors,
                    set-disum;
    CONSISTS OF     el-nloop,
                    el-nloopi,
                    el-loop,
                    el-loopi;


DEFINE SET                                      set-name;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                     handy place for storing a user-defined name;
    SUBSET OF       set-main,
                    set-disum,
                    set-dimess,
                    set-dismpr;
    CONSISTS OF     el-namel,
                    el-name,
                    el-namei;


DEFINE SET                                      set-parin;
# LAST CHANGED - 16-Aug-79     15:39:03
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                     standard parameters and switches set by cli and
                     used by programs;
    SUBSET OF       set-main,
                    set-indi,
                    set-dibmat,
                    set-disum;
    CONSISTS OF     el-stasw,
                    el-wparsw,
                    el-readsw,
                    el-punsw,
                    el-outsw,
                    el-nam,
                    el-donbuf,
                    el-abrtsw,
                    el-explsw,
                    el-newpsw,
                    el-smrysw,
                    el-dbupsw,
                    el-empsw,
                    el-rdinsw,
                    el-indx,
                    el-pindx;
```

```
DEFINE SET                                      set-prtbuf;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    buffer for lineot;
    SUBSET OF       set-dismpr;
    CONSISTS OF     el-prtbuf;


DEFINE SET                                      set-psagcm;
# LAST CHANGED - 13-Aug-79     21:53:34
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    global information for psa report programs;
    SUBSET OF       set-main;
    CONSISTS OF     el-numinm,
                    el-numnan,
                    el-notusw;


DEFINE SET                                      set-refrce;
# LAST CHANGED - 16-Aug-79     16:02:03
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                     this set contains the vectors used to sort
                     the lists of rows and columns when order
                     different of standard specified;
    SUBSET OF       set-disum,
                    set-disort,
                    set-diprrf;
    CONSISTS OF     el-refrow,
                    el-refcol,
                    el-oldrow,
                    el-oldcol;


DEFINE SET                                      set-rltcod;
# LAST CHANGED - 14-Aug-79     15:23:40
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    define every relation possible for each event type,
                    with indication of how the condition(third part)
                    is recorded;
    SUBSET OF       set-dibmat;
    CONSISTS OF     el-rlt,
                    el-evcorr,
                    el-rltn,
                    el-nbrtru,
                    el-rltfal,
                    el-nbrfal,
                    el-rltno,
                    el-third;


DEFINE SET                                      set-rowkey;
# LAST CHANGED - 17-Aug-79     16:17:08
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    storage for keys of rows of matrices;
    SUBSET OF       set-indi,
                    set-dibmat,
                    set-disum,
                    set-disort,
                    set-diprrf,
                    set-dismpr;
    CONSISTS OF     el-nrow,
                    el-row;


DEFINE SET                                      set-srtvec;
# LAST CHANGED - 16-Aug-79     16:02:03
    ATTRIBUTES ARE
                    attr-statut
                    common;
    DESCRIPTION;
                    vectors for sorting rows or columns;
    SUBSET OF       set-disort;
    CONSISTS OF     el-tag;
```

```
314  DEFINE SET                                    set-typcom;
315  # LAST CHANGED - 16-Aug-79      17:52:45
316       ATTRIBUIES ARE
317                         attr-statut
318                         common;
319       DESCRIPIION;
320                         storage of name types for conversion routines;
321       SUBSET OF         set-diprrf,
322                         set-dimess;
323       CONSISTS OF       el-plen,
324                         el-ptype;
325
```

Number of statements printed: 99

ARGUMENTS DES ROUTINES DU PROGRAMME
****

1 LISTE DES ARGUMENTS
++++++++++++++++++

Set Name: S1
Query: ? ATTRIBUTES ARE attr-statut argument
The Number of Element is: 9
The Elements are:

Object Name	Object Type
set-arg-diadhr	SET
set-arg-dibmat	SET
set-arg-dicond	SET
set-arg-difndr	SET
set-arg-dimess	SET
set-arg-dismpa	SET
set-arg-dismpr	SET
set-arg-disort	SET
set-arg-sucors	SET

2 DESCRIPTION DES ARGUMENTS CLASSES PAR ROUTINE

Set Name: S2
Query: set-arg-diadhr CONSISTS OF ?
The Number of Element is: 2
The Elements are:

Object Name	Object Type
el-evcod-diadhr	ELEMENT
el-idx-diadhr	ELEMENT


```
1  DEFINE ELEMENT                          el-evcod-diadhr;
2  # LAST CHANGED - 16-Aug-79     14:40:29
3
4  DEFINE ELEMENT                          el-idx-diadhr;
5  # LAST CHANGED - 16-Aug-79     14:40:29
6
```

Number of statements printed: 2

Set Name: S3
Query: set-arg-dibmat CONSISTS OF ?
The Number of Element is: 3
The Elements are:

Object Name	Object Type
el-ierr-dibmat	ELEMENT
el-key-dibmat	ELEMENT
el-type-dibmat	ELEMENT


```
1  DEFINE ELEMENT                          el-ierr-dibmat;
2  # LAST CHANGED - 14-Aug-79     15:23:40
3      ATTRIBUTES ARE
4                  attr-type
5                  integer;
6      ATTRIBUTES ARE
7                  attr-modified-by
8                  dibmat;
9      DESCRIPTION;
10                 error-code 0-everything O.K.
11                            1-matrix overflow
12                            2-stack-overflow
13                            3-levels overflow;
14
15 DEFINE ELEMENT                          el-key-dibmat;
16 # LAST CHANGED - 14-Aug-79     15:23:40
17     ATTRIBUTES ARE
18                 attr-type
19                 integer;
20     DESCRIPTION;
21                  D.B. key of the input name;
22
```

```
 23  DEFINE ELEMENT                                  el-type-diomat;
 24  # LAST CHANGED - 14-Aug-79      15:23:40
 25      ATTRIBUTES ARE
 26                      attr-type
 27                      integer;
 28      DESCRIPTION;
 29                      type of the input name;
 30
```

Number of statements printed: 10

Set Name: S4
Query: set-arg-dicond CONSISTS OF ?
The Number of Element is: 3
The Elements are:

Object Name	Object Type
el-cdval-dicond	ELEMENT
el-idx-dicond	ELEMENT
el-skey-dicond	ELEMENT


```
  1  DEFINE ELEMENT                                  el-cdval-dicond;
  2  # LAST CHANGED - 16-Aug-79      14:55:53
  3
```

```
  4  DEFINE ELEMENT                                  el-idx-dicond;
  5  # LAST CHANGED - 16-Aug-79      14:55:53
  6      ATTRIBUTES ARE
  7                      attr-type
  8                      integer;
  9      DESCRIPTION;
 10                      index of name bound by relation depending
 11                      on condition inserted in header matrix;
 12
```

```
 13  DEFINE ELEMENT                                  el-skey-dicond;
 14  # LAST CHANGED - 16-Aug-79      14:55:53
 15      ATTRIBUTES ARE
 16                      attr-type
 17                      integer;
 18      DESCRIPTION;
 19                      key of condition to insert in header matrix;
 20
```

Number of statements printed: 7

Set Name: S5
Query: set-arg-difhdr CONSISTS OF ?
The Number of Element is: 3
The Elements are:

Object Name	Object Type
el-evcod-difhdr	ELEMENT
el-idx-difhdr	ELEMENT
el-ival-difhdr	ELEMENT


```
1  DEFINE ELEMENT                          el-evcod-difhdr;
2  # LAST CHANGED - 14-Aug-79     18:55:52
3      ATTRIBUTES ARE
4                    attr-type
5                    integer;
6      DESCRIPTION;
7                    code of event(sensu lato) of input name;
8
9  DEFINE ELEMENT                          el-idx-difhdr;
10 # LAST CHANGED - 14-Aug-79     18:55:52
11     ATTRIBUTES ARE
12                   attr-type
13                   integer;
14     DESCRIPTION;
15                     index of D.B.( key of input name in list of rows;
16
17 DEFINE ELEMENT                          el-ival-difhdr;
18 # LAST CHANGED - 14-Aug-79     18:55:52
19     ATTRIBUTES ARE
20                   attr-type
21                   integer;
```

```
24
25
                    1-if action depends upon a false condition;


Number of statements printed: 9
```


University of Namur - DEC 20

Query System

Set Name: S6
Query: set-arg-dimess CONSISTS OF ?
The Number of Element is: 6
The Elements are:

Object Name	Object Type
el-j-dimess	ELEMENT
el-k-dimess	ELEMENT
el-lmes-dimess	ELEMENT
el-ltyp-dimess	ELEMENT
el-mes-dimess	ELEMENT
el-t-dimess	ELEMENT

```
1  DEFINE ELEMENT                                  el-j-dimess;
2  # LAST CHANGED - 16-Aug-79     17:52:45
3      ATTRIBUTES ARE
4                       attr-type
5                       integer;
6      DESCRIPTION;
7                       row/column number;
8
9  DEFINE ELEMENT                                  el-k-dimess;
10 # LAST CHANGED - 16-Aug-79     17:52:45
11     ATTRIBUTES ARE
12                      attr-type
13                      integer;
14     DESCRIPTION;
15                      key of name to print;
16
17 DEFINE ELEMENT                                  el-lmes-dimess;
18 # LAST CHANGED - 16-Aug-79     17:52:45
19     ATTRIBUTES ARE
20                      attr-type
21                      integer;
22     DESCRIPTION;
23                       length of message;
24
25 DEFINE ELEMENT                                  el-ltyp-dimess;
26 # LAST CHANGED - 16-Aug-79     17:52:45
27     ATTRIBUTES ARE
28                      attr-type
29                      integer;
30     DESCRIPTION;
31                       length of type;
32
33 DEFINE ELEMENT                                  el-mes-dimess;
34 # LAST CHANGED - 16-Aug-79     17:52:45
35     ATTRIBUTES ARE
36                      attr-type
37                      character;
38     DESCRIPTION;
39                      message to be printed;
```



```
40

41 DEFINE ELEMENT                                  el-t-dimess;
42 # LAST CHANGED - 16-Aug-79     17:52:45
43     ATTRIBUTES ARE
44                      attr-type
45                      integer;
46     DESCRIPTION;
47                      name type;
48
```

Number of statements printed: 18

Set Name: S7
Query: set-arg-disort CONSISTS OF ?
The Number of Element is: 1
The Elements are:

Object Name	Object Type
el-ierr-disort	ELEMENT


```
 1  DEFINE ELEMENT                          el-ierr-disort;
 2  # LAST CHANGED - 16-Aug-79     16:02:03
 3      ATTRIBUTES ARE
 4                     attr-type
 5                     integer;
 6      ATTRIBUTES ARE
 7                     attr-modified-by
 8                     disort;
 9      DESCRIPTION;
10                     error code =0 if everything OK
11                                 1 if overflow;
12
```

Number of statements printed: 4

Set Name: S8
Query: set-arg-dismpa CONSISTS OF ?
The Number of Element is: 5
The Elements are:

Object Name	Object Type
el-bufchp-dismpa	ELEMENT
el-lenchp-dismpa	ELEMENT
el-maxcol-dismpa	ELEMENT
el-maxrow-dismpa	ELEMENT
el-n-dismpa	ELEMENT


```
 1  DEFINE ELEMENT                          el-bufchp-dismpa;
 2  # LAST CHANGED - 16-Aug-79     18:05:38
 3      ATTRIBUTES ARE
 4                     attr-type
 5                     character;
 6      DESCRIPTION;
 7                     stuff to print;
 8


 9  DEFINE ELEMENT                          el-lenchp-dismpa;
10  # LAST CHANGED - 16-Aug-79     18:05:38
11      ATTRIBUTES ARE
12                     attr-type
13                     integer;
14      DESCRIPTION;
15                     number of characters per entry;
16
```

```
   17  DEFINE ELEMENT                              el-maxcol-dismpa;
   18  # LAST CHANGED - 16-Aug-79     18:05:38
   19      ATTRIBUTES ARE
   20                      attr-type
   21                      integer;
   22      DESCRIPTION;
   23                       maximum column number;
   24



   25  DEFINE ELEMENT                              el-maxrow-dismpa;
   26  # LAST CHANGED - 16-Aug-79     18:05:38
   27      ATTRIBUTES ARE
   28                      attr-type
   29                      integer;
   30      DESCRIPTION;
   31                       maximum row number;
   32



   33  DEFINE ELEMENT                              el-n-dismpa;
   34  # LAST CHANGED - 16-Aug-79     18:05:38
   35      ATTRIBUTES ARE
   36                      attr-type
   37                      integer;
   38      DESCRIPTION;
   39                       integer = matrix number;
1     Generalized Analyzer Version G1.0                29-Aug-79     16:18:54   Page  2
                            University of Namur - DEC 20

                                 Formatted Statements

   40



Number of statements printed: 15
```

```
1     Generalized Analyzer Version G1.0                29-Aug-79     14:12:53   Page  3
                            University of Namur - DEC 20

                                    Query    System

Set Name: S9
Query:      set-arg-dismpr CONSISTS OF ?
The Number of Element is:      9
The Elements are:

Object Name                                                          Object Type
-----------                                                          -----------

el-ce-dismpr                                                         ELEMENT
el-ie-dismpr                                                         ELEMENT
el-is-dismpr                                                         ELEMENT
el-je-dismpr                                                         ELEMENT
el-js-dismpr                                                         ELEMENT
el-n-dismpr                                                          ELEMENT
el-ncbv-dismpr                                                       ELEMENT
el-ncpe-dismpr                                                       ELEMENT
el-nrbh-dismpr                                                       ELEMENT


1     Generalized Analyzer Version G1.0                29-Aug-79     16:21:17   Page  1
                            University of Namur - DEC 20

                                 Formatted Statements

    1  DEFINE ELEMENT                              el-ce-dismpr;
    2  # LAST CHANGED - 17-Aug-79     16:35:47
    3      SYNONYMS ARE   el-buchp-dismpa;
    4



    5  DEFINE ELEMENT                              el-ie-dismpr;
    6  # LAST CHANGED - 17-Aug-79     16:35:47
    7      ATTRIBUTES ARE
    8                      attr-type
    9                      integer;
   10      DESCRIPTION;
   11                      number of last row to print;
   12
```

```
13  DEFINE ELEMENT                              el-is-dismpr;
14  # LAST CHANGED - 17-Aug-79     16:35:47
15      ATTRIBUTES ARE
16                      attr-type
17                      integer;
18      DESCRIPTION;
19                      number of first row to print;
20
21  DEFINE ELEMENT                              el-je-dismpr;
22  # LAST CHANGED - 17-Aug-79     16:35:47
23      ATTRIBUTES ARE
24                      attr-type
25                      integer;
26      DESCRIPTION;
27                       number of last column to print;
28
29  DEFINE ELEMENT                              el-js-dismpr;
30  # LAST CHANGED - 17-Aug-79     16:35:47
31      ATTRIBUTES ARE
32                      attr-type
33                      integer;
34      DESCRIPTION;
35                      number of first column to print;
36
```



```
37  DEFINE ELEMENT                              el-n-dismpr;
38  # LAST CHANGED - 17-Aug-79     16:35:47
39      SYNONYMS ARE    el-n-diampa;
40
41  DEFINE ELEMENT                              el-ncbv-dismpr;
42  # LAST CHANGED - 17-Aug-79     16:35:47
43      ATTRIBUTES ARE
44                      attr-type
45                      integer;
46      DESCRIPTION;
47                      number of columns to print before printing
48                      separation column;
49
50  DEFINE ELEMENT                              el-ncpe-dismpr;
51  # LAST CHANGED - 17-Aug-79     16:35:47
52
53  DEFINE ELEMENT                              el-nrbh-dismpr;
54  # LAST CHANGED - 17-Aug-79     16:35:47
55      ATTRIBUTES ARE
56                      attr-type
57                      integer;
58      DESCRIPTION;
59                      number of rows to print before printing
60                      separation row;
61
```

Number of statements printed: 23

Query System

Set Name: S10
Query: set-arg-sucors CONSISTS OF ?
The Number of Element is: 13
The Elements are:

Object Name	Object Type
el-cdval-sucors	ELEMENT
el-code-sucors	ELEMENT
el-evcod-sucors	ELEMENT
el-i-sucors	ELEMENT
el-idx-sucors	ELEMENT
el-inssw-sucors	ELEMENT
el-key-sucors	ELEMENT
el-numnms-sucors	ELEMENT
el-ovf-sucors	ELEMENT
el-rt-sucors	ELEMENT
el-set1-sucors	ELEMENT
el-set2-sucors	ELEMENT
el-set3-sucors	ELEMENT

Formatted Statements

```
1  DEFINE ELEMENT                          el-cdval-sucors;
2  # LAST CHANGED - 16-Aug-79     14:29:37
3      ATTRIBUTES ARE
4                    attr-type
5                    integer;
6      DESCRIPTION;
7                    value of condition 0-if true
8                                       1-if false;
9
10 DEFINE ELEMENT                          el-code-sucors;
11 # LAST CHANGED - 29-Aug-79     16:56:36
12     ATTRIBUTES ARE
13                   attr-type
14                   integer;
15     DESCRIPTION;
16     code of relation for which successors are to be found;
17
18 DEFINE ELEMENT                          el-evcod-sucors;
19 # LAST CHANGED - 16-Aug-79     14:29:37
20     ATTRIBUTES ARE
21                   attr-type
22                   integer;
23     DESCRIPTION;
24                    code of event(sensu lato) of name involved
25                    in relation;
26
27 DEFINE ELEMENT                          el-i-sucors;
28 # LAST CHANGED - 16-Aug-79     14:29:37
29     ATTRIBUTES ARE
30                   attr-type
31                   integer;
32     DESCRIPTION;
33                    code to be written on matrix;
34
35 DEFINE ELEMENT                          el-idx-sucors;
36 # LAST CHANGED - 16-Aug-79     14:29:37
37     ATTRIBUTES ARE
38                   attr-type
39                   integer;
40     DESCRIPTION;
41                    index of row where to insert code of relation;
42
43 DEFINE ELEMENT                          el-inssw-sucors;
44 # LAST CHANGED - 16-Aug-79     14:29:37
45     ATTRIBUTES ARE
46                   attr-type
47                   logical;
48     DESCRIPTION;
49                   whether D.B. key should be inserted in header
50                   matrix;
51
```

```
52  DEFINE ELEMENT                                  el-key-sucors;
53  # LAST CHANGED - 16-Aug-79      14:29:37
54      ATTRIBUTES ARE
55                      attr-type
56                      integer;
57      DESCRIPTION;
58                      D.B. key for which successrors are to be found;
59
60  DEFINE ELEMENT                                  el-numnms-sucors;
61  # LAST CHANGED - 29-Aug-79      16:56:36
62      ATTRIBUTES ARE
63                      attr-type
64                      integer;
65      ATTRIBUTES ARE
66                      attr-modified-by
67                      sucors;
68      DESCRIPTION;
69      number of elements successors;
70
71  DEFINE ELEMENT                                  el-ovf-sucors;
72  # LAST CHANGED - 16-Aug-79      14:29:37
73      ATTRIBUTES ARE
74                      attr-type
75                      sucors;
76      DESCRIPTION;
77                      error-code 0-ok
79                                 3-levels overflow;
80
81  DEFINE ELEMENT                                  el-rt-sucors;
82  # LAST CHANGED - 16-Aug-79      14:29:37
83      ATTRIBUTES ARE
84                      attr-type
85                      integer;
86      DESCRIPTION;
87                      relation type;
88
89  DEFINE ELEMENT                                  el-set1-sucors;
90  # LAST CHANGED - 16-Aug-79      14:29:37
91      ATTRIBUTES ARE
92                      attr-type
93                      integer;
94      DESCRIPTION;
95                       first set used;
96
97  DEFINE ELEMENT                                  el-set2-sucors;
98  # LAST CHANGED - 29-Aug-79      16:56:36
99      ATTRIBUTES ARE
100                     attr-type
101                     integer;
102     DESCRIPTION;
103     indicates where second part of relation is recorded;
104
105 DEFINE ELEMENT                                  el-set3-sucors;
106 # LAST CHANGED - 16-Aug-79      14:29:37
107     ATTRIBUTES ARE
108                     attr-type
109                     integer;
110     DESCRIPTION;
111                     indicates where third part (condition) is
112                     recorded;
113
```


Formatted Statements

Number of statements printed: 40

BUMP
0 0 6 5 0 3 7 9 4
*FM B16/1979/12/2