**MASTER EN SCIENCES INFORMATIQUES**

# Contribution à la modélisation et à l'évaluation du fonctionnement des systèmes d'information de bureau

Urbain, Annick; Van Belle, Philippe

*Award date:*
1987

*Awarding institution:*
Universite de Namur

# CONTRIBUTION A LA MODELISATION ET A L'EVALUATION DU FONCTIONNEMENT DES SYSTEMES D'INFORMATION DE BUREAU

Annick URBAIN et Philippe VAN BELLE

PROMOTEUR : Monsieur R. LESUISSE

Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique.

Année académique 1986 - 1987

Nous tenons à adresser nos plus sincères remerciements à Monsieur Roland Lesuisse pour l'attention toute particulière qu'il a accordée à notre travail ainsi que pour les précieux conseils qu'ils nous a prodigués tout au long de sa réalisation.

Que soit également remercié Monsieur Yves Pigneur pour son acceuil chaleureux, ainsi que l'aide, et le temps qu'il nous a consacrés à Lausanne.

Nos plus vifs remerciements vont également à Monsieur Jean-Marie Leheureux pour le temps et les précieuses critiques qu'il nous a prodigués.

Que soient enfin remerciés nos parents respectifs sans lesquels il nous aurait été impossible de mener à bien nos études et ce travail, ainsi que tous ceux qui de près ou de loin nous ont aidés.

Namur, le 26 août 1987

Annick Urbain
Philippe Van Belle

## ABSTRACT

Le but de ce mémoire est tout d'abord de concevoir un langage de spécification du fonctionnement d'une classe particulière de systèmes d'information : les systèmes d'information de bureau; ensuite de permettre au concepteur d'une application de bureau d'évaluer par simulation une modélisation effectuée au moyen de ce langage.

The aim of this work is first to define a language for the behaviour specification of a special kind of information systems : the office information systems; then, to permit an office analyst to evaluate by simulation a modelization expressed with this language.

# TABLE DES MATIERES

Annexes

1. Syntaxe du langage
2. Exemple de base
3. Implémentation
4. Modèles MIB

# CHAPITRE I

# INTRODUCTION

## 1.1. Preambule

Le but de ce mémoire est tout d'abord de concevoir un langage de spécification du fonctionnement d'une classe particulière de systèmes d'information : les systèmes d'information de bureau; ensuite de permettre au concepteur d'une application de bureau d'évaluer par simulation une modélisation exprimée au moyen de notre langage.

Dans un premier temps, cette introduction présente les concepts de bureau et de système d'information de bureau. Ensuite, nous examinons pourquoi et comment modéliser de tels systèmes. Enfin, nous introduisons la base de notre langage: les pré et postconditions.

## 1.2. Organisation et Bureau

### 1.2.1. Organisation

La structure d'une organisation peut être décrite, entre autre, au moyen d'un graphe appelé "diamant de Leavitt" [LEAVITT, 65] (cfr. fig. 1.1).

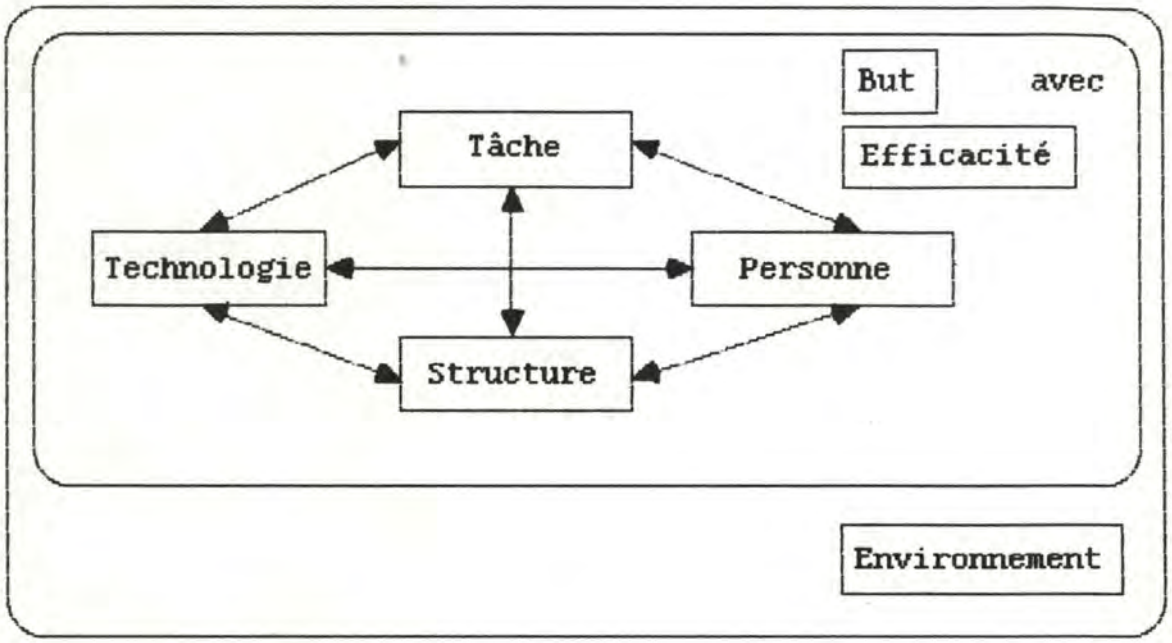



fig. 1.1. "Le diamant de Leavitt"

Ce diamant se compose de quatre facettes : les Personnes, les Tâches, la Technologie et la Structure.

Les personnes accomplissent certaines tâches en utilisant de la technologie.

Personnes, tâches et technologie sont intégrées dans une structure qui permet à l'organisation de fonctionner dans un environnement en poursuivant son but avec efficacité.

### 1.2.2. Bureau

Le bureau peut être considéré comme une organisation réduite à la taille d'un de ses composants (service, département ...). Il peut donc être, lui également, représenté au moyen du diamant de Leavitt, à la restriction que :

- Les tâches à exécuter, qu'elles soient opérationnelles ou décisionnelles, sont des tâches traitant de l'information.

- La technologie prend en considération les outils bureautiques ou informatiques.

- Les personnes (secrétaires, employés, cadres ...) sont soit les exécutants, soit les responsables des tâches.

Le fonctionnement d'une organisation, et donc à fortiori d'un bureau, nécessite l'existence d'un système d'information (SI) au sein de cette organisation ou de ce bureau.

## 1.3. Le système d'information ( de bureau )

### 1.3.1 Définition

Le fonctionnement d'une organisation engendre des besoins qu'elle satisfait au moyen de son système d'information (SI).

Dans [BOD_PIG, 83], le SI est défini comme suit : "Il s'agit d'une construction formée d'ensembles contenant des informations, des traitements et des ressources (...) Il englobe des traitements automatisés et manuels".

Vu le rapport entre organisation et bureau, on parlera de SI de bureau (SIB) quand l'organisation sera réduite à la taille du bureau (cfr 1.2.2.).

### 1.3.2. Décrire un SIB

La description d'un SI ( comme d'un SIB ) comprend deux parties. D'une part la structure, d'autre part, le fonctionnement.

La structure décrit les propriétés statiques d'un SIB, c-à-d les différentes entités du réel perçu ainsi que les relations existant entre ces entités.

Le fonctionnement ou comportement décrit les propriétés dynamiques, c-à-d les traitements ainsi que la manière dont se réalise leur enchaînement.

Il existe généralement deux manières d'aborder le fonctionnement d'un SI : l'approche par les traitements ou par les données. Le choix d'une de ces approches va déterminer l'étroitesse du lien entre description du fonctionnement et de la structure.

Par exemple, l'approche par les données nécessite une connaissance complète de la structure avant de pouvoir décrire le fonctionnement. L'approche par les traitements permet quant à elle, de concevoir le fonctionnement indépendamment de la structure.

Ces différentes approches seront étudiées en détail dans le chapitre II.

Examinons à présent l'enjeu de la modélisation du fonctionnement pour le concepteur d'une application de bureau.

## 1.4. Pourquoi décrire le fonctionnement d'un SI ?

Il existe plusieurs raisons de modéliser le comportement d'un SIB.

La première de ces raisons consiste à comprendre et expliquer le fonctionnement de celui-ci ou son éventuel dysfonctionnement. Une bonne modélisation permet également l'évaluation de la "faisabilité" du fonctionnement décrit en fonction des ressources dont l'organisation dispose.

De plus, si le fonctionnement tel qu'il est spécifié au moyen du modèle est irréalisable ou inefficace, il est possible de modifier la modélisation choisie et de tester ainsi à volonté de nouvelles hypothèses de fonctionnement.

La deuxième de ces raisons est de vouloir s'assurer plus ou moins fortement à l'avance de la faisabilité d'un projet de changement, c-à-d examiner sa cohérence, son efficacité et son ajustement aux besoins.

## 1.5. Comment décrire le fonctionnement d'un SI ?

La modélisation du comportement d'un SIB comprend les aspects suivants, qui sont dérivés de [PIGNEUR, 84].

A. Le comportement fonctionnel proprement dit, à savoir :

* Dans quelles conditions une tâche de bureau peut s'exécuter.

* Quelles sont les propriétés du résultat de cette exécution.

En connaissant ces deux éléments, il n'est pas nécessaire de spécifier de quelle manière va se réaliser l'enchainement des différentes tâches.

B. Les moyens requis par l'exécution des tâches et modélisés par le concept de ressource. Ce concept bien que lié à la notion de performance d'un système, est cependant nécessaire afin de décrire complètement un SIB. En effet une évaluation du fonctionnement de celui-ci ne peut se faire sans que l'on connaisse les ressources disponibles.

C. Si on considère qu'une activité est un ensemble de tâches concourantes, il faut spécifier l'échéancier des événements externes à cette activité, ceci afin de pouvoir estimer la charge du système.

## 1.6. Les besoins auxquels doit répondre notre modèle de spécification du fonctionnement d'un SIB

Comme nous l'avons vu au point 1.3.2, le but de ce travail est de définir un modèle de représentation du comportement d'un SIB.

Ce modèle doit permettre de calquer, de la manière la plus idéale possible, le travail tel qu'on voudrait qu'il s'accomplisse dans les bureaux.

La caractéristique principale du travail en bureau est celle-ci : les tâches sont exécutées par des individus différents qui connaissent leur travail, mais qui généralement n'ont aucune idée précise de l'activité globale. Il n'est donc pas aisé de définir un scénario de fonctionnement.

Le modèle que nous allons définir doit permettre à son utilisateur, à savoir le concepteur d'une application bureautique, de spécifier les tâches constitutives d'une activité indépendamment l'une de l'autre, et donc ainsi d'ignorer dans son détail le flux des informations circulant de tâche en tâche. L'approche du comportement est une approche "locale" à la tâche.

Notre hypothèse est la suivante : les responsables ou exécutants d'une tâche donnée, connaissent et peuvent exprimer les conditions pour que la tâche soit exécutée, ainsi que les résultats de son exécution.

Maintenant que le cadre de travail est fixé, examinons au moyen d'une analogie, une manière de spécifier le fonctionnement d'un système au moyen de pré et postconditions.

## 1.7. Première approche des pré et postconditions

Il existe une certaine analogie entre le travail de bureau ( où chacun connait la ou les tâches qu'il est censé exécuter ) et les méthodologies de la programmation qui préconisent la découpe des programmes en modules conçus indépendamment l'un de l'autre, quoique dans une stratégie globale.

En effet, dans une certaine mesure, l'exécutant de la tâche comme le programmeur du module effectue son travail sans se soucier de celui des autres, et donc indépendamment de tout scénario d'activité ou d'exécution.

Afin de permettre cela il faut pouvoir cacher tout scénario de comportement, qu'il s'agisse du bureau ou du programme. Dans [VANLAM1, 86] on trouve des règles aidant à la conception des modules constituant un programme. Si un module reçoit en entrée des arguments et produit en sortie des résultats, ces règles

veulent que :

- les arguments doivent vérifier certaines conditions afin que le module s'exécute correctement

- si le module s'est exécuté correctement, ses résultats possèdent certaines propriétés.

Les conditions sur les arguments sont appelées "préconditions" et les propriétés des résultats "postconditions". Ce type de spécification est appelée spécification par assertions.

Par exemple, voici la spécification par assertions d'un petit module appelé "EXISTE". La fonction de celui-ci est de vérifier si un nombre "N" appartient ou n'appartient pas à un ensemble ordonné de maximum 100 éléments ,"E". Le résultat de son exécution est un booléen "B" qui est vrai si "N" appartient à "E", faux sinon. En outre, la spécification utilise une fonction "nombre(K)" qui permet de consulter le " Kième " élément de l'ensemble.

EXISTE : Arguments : N -> entier,
E -> { nombre(k) t.q. 0<=K<=100 };

Précondition:

Pour tout nombre(i), nombre(j) appartenant à E : i<j => nombre(i) < nombre(j);

Résultat : B -> booléen;

postcondition :

(Il existe un K tel que : nombre(K) = N) et (B = vrai)
ou
(il n'existe pas de K tel que : nombre(K) = N) et (B =faux).

Nous verrons dans le deuxième chapitre plus en détail la base théorique des pré et postconditions.

Un tel type de spécification lors de l'analyse conceptuelle offre des avantages car cela facilite le passage de l'analyse au développement. En effet, si les traitements sont spécifiés de cette manière leur automatisation ( si elle s'avère nécessaire ) sera plus aisée, par le fait que l'on adopte à un niveau conceptuel des principes proches de ceux adoptés par les méthodologies de développement de logiciels.

L'inconvénient immédiat qui se dégage de ce type de spécification, est l'effort d'abstraction que doit fournir l'utilisateur.

## 1.8. Plan du travail

Tel est donc l'objectif et le cadre général de notre travail. Son plan sera le suivant :

Dans le deuxième chapitre, nous présenterons, comme il est usuel, la littérature existante en matière de représentation du fonctionnement des SI.

Dans le troisième chapitre, nous rappellerons les concepts de base du modèle descriptif MIB ( modèle de SIB ); en effet le fonctionnement du système est lié aux informations manipulées, aux ressources disponibles, à l'organisation, etc... Toutes ces propriétés sont représentées au moyen du modèle MIB [DACHOUF,86]. La seconde partie de ce chapitre présente les principes fondamentaux de notre modèle de représentation du fonctionnement d'un SIB.

Le quatrième chapitre décrira le langage adopté pour pouvoir exprimer le modèle et ainsi permettre qu'il soit exploité par un logiciel d'évaluation par simulation.

Le cinquième chapitre montrera, sur un exemple, comment on peut décrire le comportement d'un SI à l'aide du langage défini au chapitre précédent.

Le sixième chapitre présentera notre modèle d'implémentation qui nous permettra d'utiliser un outil de simulation existant; ce choix implique un processus de traduction de notre langage dans celui du simulateur.

Le septième chapitre décrira l'architecture logicielle de notre programme de traduction.

Enfin, les annexes contiendront :

-Un rappel du modèle MIB.

-Une description complète et formelle de la syntaxe de notre langage.

-Un exemple complet MIB, sa traduction dans le langage du simulateur, ainsi que les résultats de la simulation.

-L'architecture physique et le code du programme de traduction.

# CHAPITRE II

# ETUDE DES MODELES EXISTANTS

## 2.1. Introduction

Après avoir présenté le sujet de ce travail, il semble intéressant de situer les différentes approches de l'expression du comportement d'un SI dans la littérature et les axes de recherche existants.

Ce chapitre commence par une classification des modèles. Nous illustrons ensuite chacune des classes par des modèles représentatifs dont nous présentons les principes, un exemple si nécessaire, ainsi qu'une évaluation dans le cadre de notre recherche. Enfin, nous expliquons en quoi notre vision du fonctionnement se différencie ou se rapproche des modèles décrits.

## 2.2. Classification

Il existe deux manières générales d'aborder la représentation du fonctionnement d'un SI : l'approche basée sur les données et celle basée sur les traitements.

### 2.2.1. Modèles basés sur les traitements

L'élément central de cette approche est une vision globale de l'activité spécifiée. L'accent est mis sur la coordination entre les différents traitements.

Les modèles représentatifs de cette tendance sont :

- le modèle présenté par Bodart et Pigneur dans [BOD_PIG,83] et le modèle de Pigneur, [PIGNEUR, 84], tous deux basés sur les concepts d'événement et de processus

- le modèle proposé par Ellis dans [ELLIS, 80], caractérisé par la modularité de la découpe des activités de bureau et basé sur des fondements mathématiques.

### 2.2.2. Modèles basés sur les données

Les éléments centraux de ce type d'approche sont les données. Les traitements ( ou dans le cadre d'un SIB, les tâches de bureau ) ne sont considérés que comme une suite d'opérations sur les données. Ils ne sont donc envisagés qu'individuellement.

On ne peut déduire aucun comportement global du système à partir d'une telle spécification (sauf [TSICHRI,??]).

Les modèles représentatifs de cette tendance sont :

- le modèle de Brodie et Silva [BRO_SIL, 82] développé dans le cadre des bases de données

- le modèle REMORA de Rolland présenté dans [ROLLAND, 82]

- le modèle proposé par Tsichritzis [TSICHRI,??], Office Object Flow.

Présentons à présent de manière plus détaillée ces différents modèles.

## 2.3. Modèles basés sur les traitements

### 2.3.1. EVENEMENT ET PROCESSUS

#### 2.3.1.1. Le modèle de la "dynamique" de Bodart et Pigneur

Ce modèle est présenté dans [BOD-PIG, 83]. Il sera étudié de manière plus approfondie, car il y sera fait souvent référence par la suite.

##### A : Concepts de base

Ce modèle repose sur deux concepts fondamentaux : le processus et l'événement.

Un PROCESSUS est l'exécution d'une procédure de traitement de l'information dont la progression peut être représentée par son ETAT. Le cycle de vie d'un processus comporte les états : déclenché, actif, interrompu et terminé.

La notion de CHANGEMENT D'ETAT en découle, on a les changements : activation, interruption, redémarrage et terminaison.

Un EVENEMENT correspond à un changement d'état du système caractérisé dans le temps et dans l'espace. On distingue des événements INTERNES et EXTERNES.

Un événement est dit externe s'il correspond au franchissement de la frontière du SI par un message en provenance de son environnement et qui déclenche en réaction un processus du SI. Le "message" permet de véhiculer des informations.

Un événement est dit interne s'il correspond à un changement d'état interne au SI.

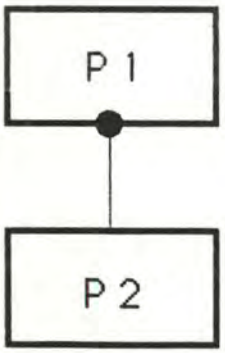
## B : Enchainements proposés

Ce modèle présente six structures élémentaires d'enchaînement de processus qui, combinées, permettent de décrire le fonctionnement d'un système :

- l'enchaînement séquentiel
- l'enchaînement éclaté
- l'enchaînement multiple
- l'enchaînement conditionnel
- l'enchaînement convergent
- l'enchaînement synchronisé.

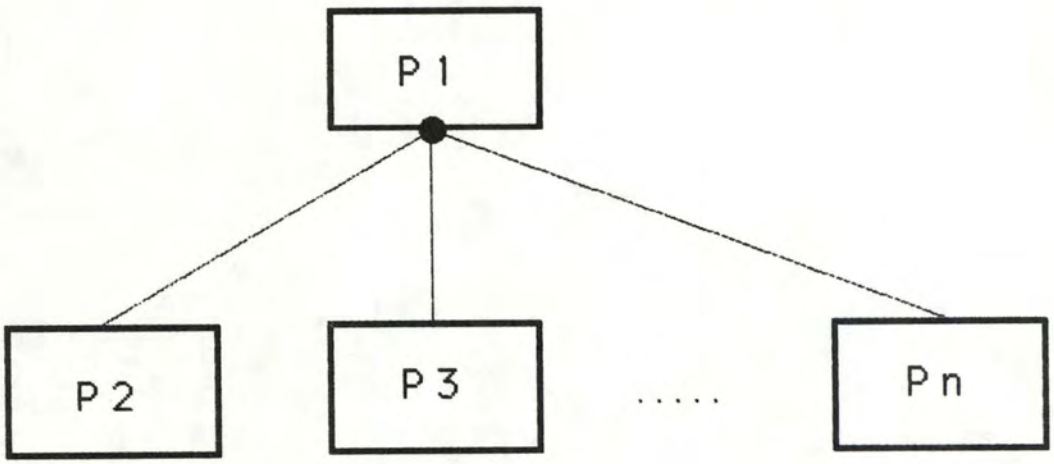
### -1- ENCHAINEMENT SEQUENTIEL

Un enchaînement est dit séquentiel lorsque la terminaison d'un processus P1, représentée par un point, provoque le déclenchement d'un processus P2. La représentation graphique de cet enchaînement est conventionnellement la suivante :



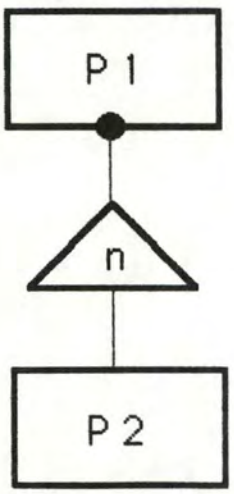

### -2- ENCHAINEMENT ECLATE

Un enchaînement est dit éclaté lorsque la terminaison d'un processus P1 provoque le déclenchement simultané de P2,P3,....Pn. Cet enchaînement peut être représenté au moyen du graphe suivant.



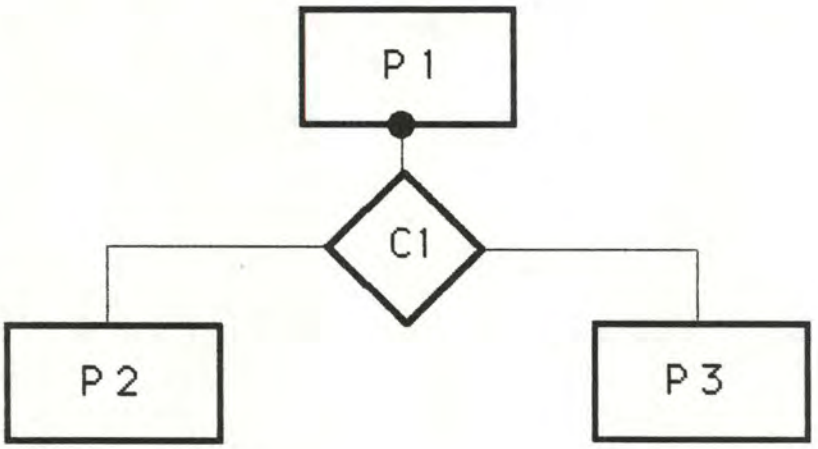

-3- ENCHAINEMENT MULTIPLE

Un enchaînement est dit multiple lorsque la terminaison d'un processus P1 provoque le déclenchement simultané de n ( n>1 ) processus de type P2, c'est-à-dire n exécutions du traitement P2. La représentation graphique de ce type d'enchaînement est :



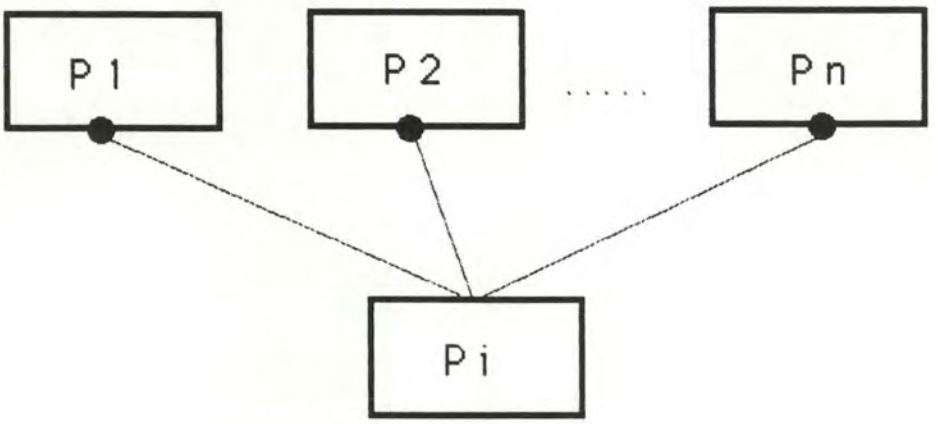
## -4- ENCHAINEMENT CONDITIONNEL

Un enchaînement est dit conditionnel lorsque la terminaison d'un processus Pi déclenche le processus P2 si la condition Ci est vraie et déclenche P3 si la condition Ci est fausse.
Cet enchaînement est représenté au moyen du graphe suivant.



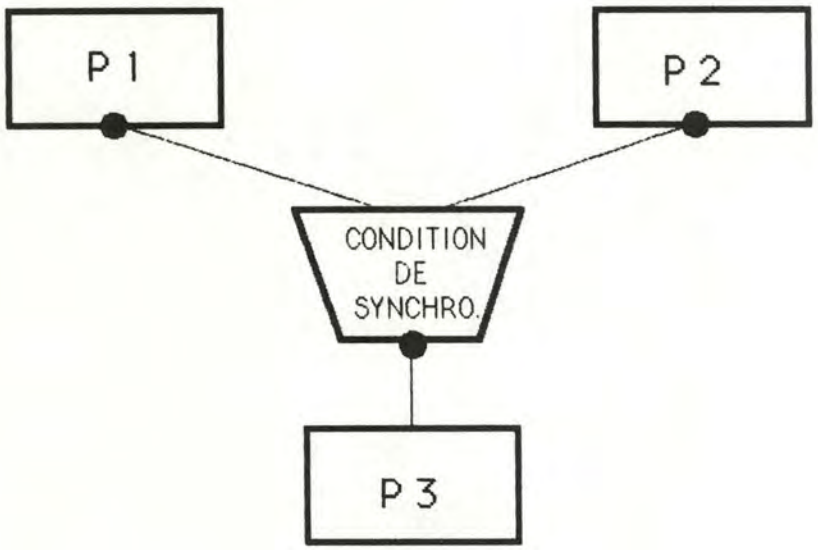

## -5- ENCHAINEMENT CONVERGENT

Un enchaînement est dit convergent si le déclenchement des processus Pi est provoqué par la terminaison d' un des processus P1,...Pn. On représente conventionnellement cet enchaînement de la manière suivante :



## -6- ENCHAINEMENT SYNCHRONISE

Si P1 et P2 sont des processus asynchrones travaillant à des rythmes différents, l'enchaînement synchronisé va permettre de déclencher P3 suivant la manière dont P1 et P2 se sont terminés. La représentation graphique de ce type d'enchaînement est la suivante :




La coordination de P1 et P2 est réglée par le point de synchronisation. Celui-ci définit une condition de synchronisation et donc : le processus P3 ne sera déclenché qu'à la terminaison de P1 et P2.

La réalisation de la condition de synchronisation est l'événement déclencheur de P3.

### C : Evaluation

Ce modèle, comme tous ceux qui sont basés sur l'approche par les traitements, montre bien le fontionnement du système de manière globale. Le scénario d'enchaînement des différents traitements, ou flux de contrôle, doit être connu avant d'être modélisé. De plus, un traitement ( dont l'exécution est modélisée par un processus ) est aveugle, c'est à dire qu'il

ignore les conditions ( état global du SI ) dans lesquelles il est déclenché et qu'il n'a aucun moyen de contrôle sur les noeuds de décision qui le précèdent dans les différents enchaînements du scénario décrit.

Examinons à présent un second modèle basé sur les notions d'événements et de processus.

### 2.3.1.2. Le modèle de Pigneur, [PIGNEUR, 84]

A : Concepts

La méthodologie proposée par Yves PIGNEUR "privilégie une approche par les traitements basée sur les concepts d'événements et de processus" ( cfr 2.3.1.1. ).

Cette approche ( tout comme la précédente ) assure l'autonomie de la modélisation du comportement par rapport à la description des données tout en attribuant aux processus un rôle intégrateur puisque

- -ils sont déclenchés par et causent la survenance d'événements (vue processus-événement)
- -ils utilisent des données (vue processus-données)
- -ils requièrent des ressources (vue processus-ressources).

Par opposition au précédent, ce modèle décrit systématiquement le role de l'événement et les modalités de déclenchement des processus par les événements.

Le schéma entité-association suivant décrit le modèle de la dynamique envisagé ici (cfr. fig.2.1).

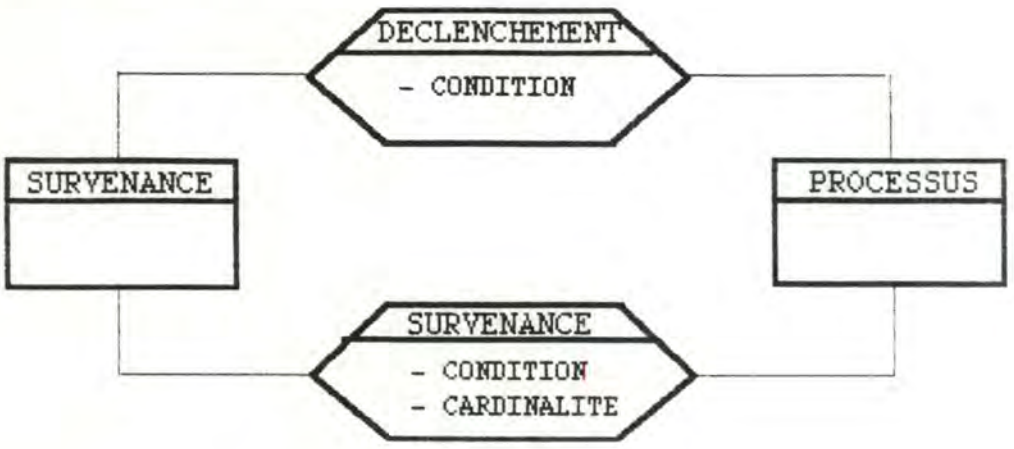



Fig. 2.1.

Association SURVENANCE :

L'association SURVENANCE représente la survenance d'un événement causée par un processus. Comme celui-ci peut causer la survenance de plusieurs événements ceci doit pouvoir être spécifié au moyen d'une contrainte de cardinalité.

La survenance de certains événements peut ne pas être systématiquement causée par un processus, il faut donc pouvoir spécifier dans quelles conditions un processus cause la survenance de certains événements.
Un prédicat spécifie alors la condition de survenance en fonction du problème.

La condition attachée à l'association SURVENANCE constitue une POSTCONDITION dynamique puisqu'elle donne une vue partielle de l'état dans lequel le SI se trouve lorsqu'un tel processus se termine.

Exemple :

Un processus "enregistrement-lettre-de-contact" cause la survenance d'un événement "lettre-à-traiter" si la "lettre-de-contact" qu'il reçoit est valide, ou d'un événement "lettre-à-renvoyer" si elle est invalide.

Ceci se traduit dans le contexte du langage adopté par Pigneur :

```
DEFINIR  PROCESSUS enregistrement-lettre-de-contact;
         CAUSE 1 lettre-à-traiter SI lettre-valide;
         CAUSE 1 lettre-à-renvoyer SI letter-invalide;

DEFINIR  CONDITION lettre-valide;
        VRAIE DANS "valeur de probabilité quelquonque";

DEFINIR  CONDITION lettre-invalide;
        VRAIE DANS "valeur de probabilité quelquonque";
```

Association DECLENCHEMENT :

"les événements sont des stimuli auxquels le SI réagit par

le déclenchement de processus. Une association de déclenchement représente la contribution d'un événement au déclenchement d'un processus."

L'existence d'une association de déclenchement peut être soumise à certaines contraintes exprimées sous forme de prédicats qui doivent être vérifiés pour que l'événement contribue au déclenchement d'un processus.

La spécification de la condition de déclenchement constitue une PRECONDITION dynamique au déclenchement de ces processus puisqu'elle traduit l'état dans lequel le SI doit se trouver pour qu'un tel processus commence son exécution.

Exemple :
---

Chaque événement "arrivée-lettre-de-contact" déclenche un processus "enregistrement-lettre-de-contact" si la valeur d'attribut "présence-signature" de l'événement est "vrai".

Ceci se traduit dans le langage adopté par Y.Pigneur par :

```
DEFINIR PROCESSUS enregistrement-lettre-de-contact;
      DECLENCHE PAR arrivée-lettre-de-contact
                  QUAND enregistrement-possible;


DEFINIR CONDITION enregistrement-possible;
      PREDICAT :
présence-signature DE arrivée-lettre-de-contact = VRAI;
```

B : Evaluation
---

Ce modèle est plus puissant que le précédent car il nécessite moins de concepts pour exprimer le fonctionnement d'un système (par exemple on ne retrouve plus le point de synchronisation).

En plus, cette modélisation introduit implicitement les notions de pré et de postcondition sous forme de conditions attachées aux associations de "déclenchement de processus" et de "survenance d'un événement"; ceci permet de cacher le flux de contrôle et donc le scénario dans ces conditions, ce qui constitue une première étape vers l'indépendance de la spécification des traitements vis-à-vis du scénario.

## 2.3.2 ICN : UN EXEMPLE DE MODELE PROCEDURAL

### A : Introduction

Le modèle d'Ellis [ELLIS, 80], appelé ICN (Information Control Nets), a été développé au "Palo Alto Research Center" de la compagnie XEROX aux USA dans le but de :

- décrire les procédures de bureau sous la forme de graphes
- analyser les performances et la régularité de ces procédures
- déterminer les points faibles de l'organisation et les corriger.

Trois principaux objectifs ont guidé les concepteurs de ce modèle lors de son développement :

(1) la définition d'un modèle mathématiquement exprimable,

(2) capable de décrire la plupart des "scénarios du bureau",

(3) et cependant suffisamment simple et compréhensible par les travailleurs de bureau.

### B : Le modèle

Nous nous contenterons de présenter ici une définition informelle du modèle, pour plus de détails, on consultera [ELLIS, 80].

Le modèle ICN est basé sur la définition du bureau comme un ensemble de PROCEDURES réalisées par un potentiel humain et matériel. Il décrit les procédures de bureau sous la forme de graphes (arcs, cercles, polygones, ...) où chacun des éléments a une signification propre.

Chaque procédure est constituée d'un ensemble interdépendant

d'ACTIVITES (représentées par des cercles) reliées par des CONTRAINTES DE PRECEDENCE (arcs).

Il est possible de considérer deux types d'activités :

- les activités de base : elles peuvent exprimer une décision (petits cercles vides) ou permettre la représentation de processus parallèles (petits cercles pleins)

- les activités composées : elles peuvent être analysées comme une procédure et donc redécomposées en différentes activités pour permettre une description plus détaillée. C'est la notion de modularité du modèle ICN.

Les information utilisées et engendrées par les activités sont stockées dans des moyens de rangement : REPOSITOIRES (documents, fichiers, ...). Un carré représente un répositoire permanent (stockage à long terme) et le triangle est un répositoire temporaire.

Le modèle ICN a été utilisé pour deux types d'analyse dans un environnement bureautique :

- analyse qualitative :

elle permet de détecter des incohérences dues à un manque de précision dans la description d'une procédure.

Exemple : le cas où un calcul de taxes sur salaire serait exécuté le même jour qu'une augmentation de salaire. Si une règle de précédence n'a pas été convenablement spécifiée, certains employés seraient taxés sur leur ancien salaire, d'autres sur leur salaire après augmentation.

Ce type d'analyse permet également le détection de contraintes fatales qui, sous certaines conditions, peuvent bloquer le déroulement normal d'une procédure.

Exemple : le cas où un étudiant devrait prouver qu'il jouit d'une situation financière stable pour pouvoir s'inscrire dans une université. Il y aurait bloquage si ce même étudiant ne pouvait obtenir de bourse sans qu'il soit inscrit au préalable dans cette université.

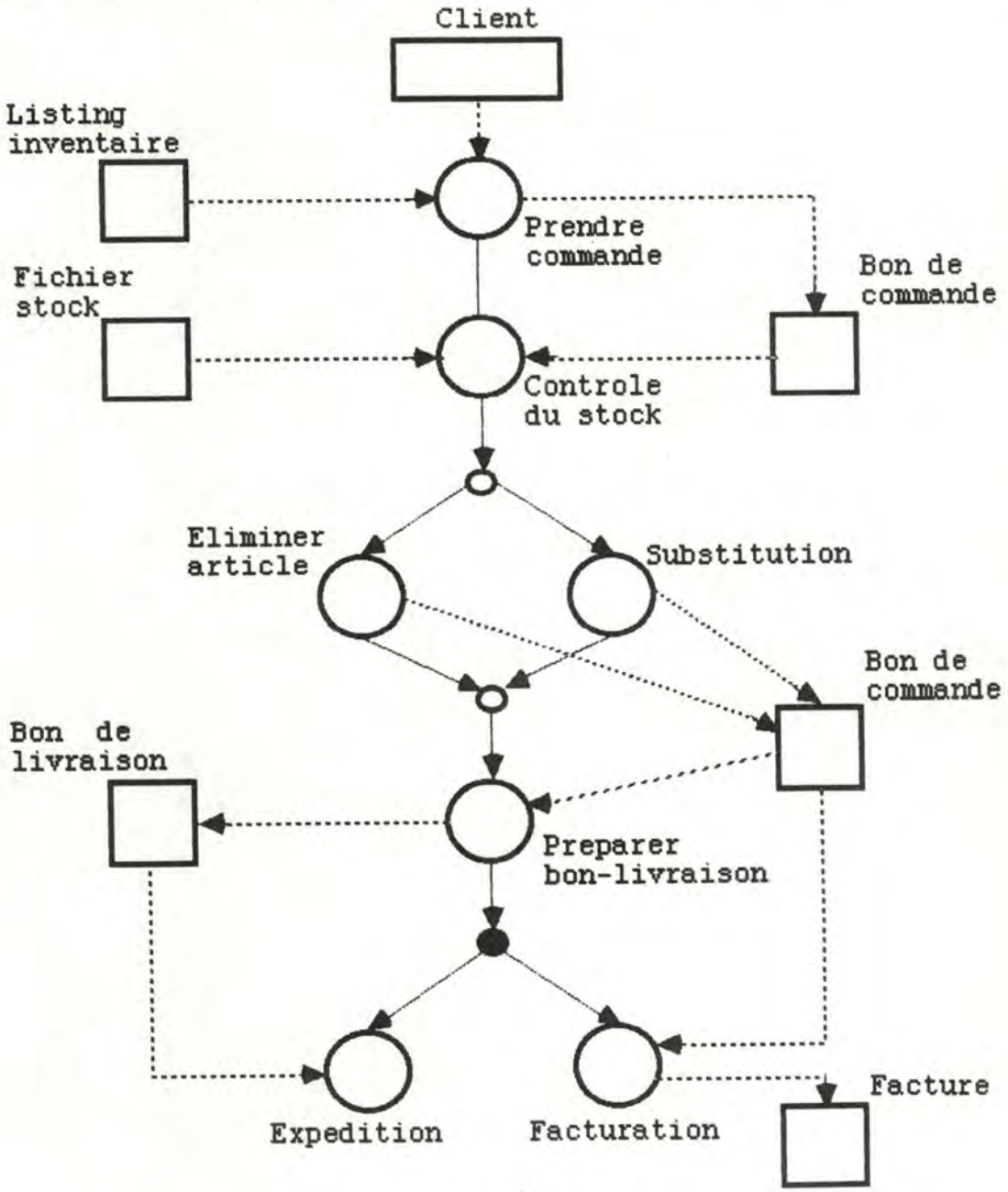
- analyse quantitative :

elle permet une évaluation des flux d'informations avec détection des goulots d'étranglement et de saturation au niveau de certaines activités.

De plus, cette analyse quantitative permet l'étude de différents scénarios d'automatisation. A ce niveau, nous dépassons la simple analyse de l'existant et aboutissons sur des propositions d'automatisation des moyens (ou outils) bureautiques utilisés.

C : Représentation graphique

La figure ci dessous représente le graphe ICN d'une procédure de vente. (cfr. fig.2.2).
A partir des besoins du client et des contraintes de l'inventaire hebdomadaire, le représentant définit un bon de commande. Un contrôle du stock effectivement disponible est effectué. A partir de ce contrôle, on prend une décision quant à la substitution d'un article par un autre, ou bien l'élimination pure et simple de cet article.
Ensuite, deux activités peuvent être exécutées en parallèle : la facturation et l'expédition.



D : Evaluation
---

Ce modèle permet de décrire une activité dans toute sa généralité (grâce au principe de la modularité) en évitant de prendre en considération certains détails occasionnant une plus grande complexité.

Mais l'hypothèse de base du modèle qui consiste à considérer l'organisation comme un ensemble de procédures, peut aboutir à une analyse incomplète qui ignorera certains éléments importants pour une bonne compréhension. Cette décomposition peut donc être une source d'oublis au niveau de certains aspects de l'organisation.

Cet inconvénient peut cependant être perçu comme un avantage si l'on considère que de nombreuses organisations manquent de structuration dans le déroulement des opérations à effectuer et que l'emploi d'un tel modèle, s'il est très utile et adaptable dans des organisations déjà bien structurées, permettra justement d'aider les décideurs à suivre une méthode de réorganisation qui les obligera à rationnaliser leurs services.

Nous retiendrons de ce modèle une caractéristique intéressante de " flux de précédence et de flux d'information ". Cette notion permet en effet, cette notion permet de rappeler le degré de complétude de ce modèle qui, s'il est de type procédural, comporte cependant de nombreux avantages à rattacher aux modèles basés sur l'étude des flux d'information.

### 2.3.3 CONCLUSION DE L'APPROCHE PAR LES TRAITEMENTS

Les avantages des modèles basés sur les traitements sont ceux :

-de perception globale de l'activité spécifiée.

-de modularité :

Il est généralement possible de réaliser une spécification de haut niveau d'abstraction pour obtenir une vue d'ensemble d'une activité en évitant certains détails qui peuvent être encombrants pour une compréhension simple et claire. Par contre, lorsqu'une description plus détaillée d'une activité est nécessaire, alors ces modèles permettent une spécification plus fine de l'activité considérée.

de parallélisme :

Les représentations graphiques qu'ils offrent généralement permettent de montrer de façon claire toutes les tâches qui peuvent être réalisées en parallèle.

Cependant, tous ces modèles nécessitent une connaissance à priori de l'activité globale; ce qui n'est pas toujours possible. De plus le caractère apparent des flux de contrôle ( par exemple : le point de synchronisation dans le modèle de Bodart et Pigneur ou les arcs de précédence du modèle ICN) rend ardu toute modification de scénario. En effet, un changement à un endroit du flux de contrôle peut en induire beaucoup d'autres si les répercussions d'un changement ne sont pas automatisées.

## 2.4. Modèles basés sur les données

### 2.4.1. INTRODUCTION

Avant d'examiner les modèles basés sur les données ( ou plus généralement sur les objets ), il est intéressant de situer la manière dont ces modèles vont concevoir le fonctionnement d'un SI. Les objets peuvent prendre différentes valeurs que nous appellerons "états". L'espace d'états d'un objet est l'ensemble des valeurs que celui-ci peut prendre.

Le fonctionnement d'un SI ne sera plus vu comme une interconnection entre les différents traitements constituant le système mais bien comme une suite d'états à travers lesquels le système va évoluer.

### 2.4.2. FONDEMENT THEORIQUE DES PRE ET POSTCONDITIONS

#### A : Introduction

Les différents modèles dont nous allons examiner les principes de spécification du fonctionnement, utilisent généralement des pré et postconditions; c'est pourquoi il nous semble important de rappeler les fondements théoriques de ces pré et postconditions. En effet, si nous avons déja abordé ces concepts dans l'introduction, ce ne fut pas de manière formelle. Faisons le donc maintenant à partir de la présentation générale de Dijkstra [DIJKSTR, 76].

Dans son livre "A discipline of programming", Dijkstra utilise des équations pour caractériser des ensembles d'états d'un système d'information. Ex: x = y.

Il appelle de telles équations des conditions, et réserve le terme prédicat à l'expression formelle dénotant la condition, c'est à dire :"x = y"et "y = x" sont deux prédicats différents mais, tous deux dénotent la même condition.

De cette manière, on dira que si un système arrive dans un état satisfaisant une condition P, le système établit de manière certaine que le prédicat P est vrai.

Dijkstra s'appuie sur une théorie plus ancienne qui est celle des automates finis et s'intéresse aux systèmes qui, démarrant dans un "état initial", se termineront dans un "état final" dépendant du choix de "l'état initial". Si on veut qu'un système réalise certains buts , il faut que l'état final dans lequel il se trouve après exécution satisfasse certaines conditions. Celles-ci sont appelées postconditions car elles imposent une condition sur l'état dans lequel le système doit se trouver après son activité.

" Le système produit une réponse " a la même signification que " le système atteint un état final satisfaisant une postcondition ". A présent, il reste à connaître l'ensemble des états initiaux tel que l'activation du système dans un de ces états initiaux, laissera celui-ci dans un état final satisfaisant la post-condition.

La condition qui caractérise l'ensemble de tous ces états initiaux, est appelée la "plus faible précondition". Celle-ci est bien la plus faible car plus la condition est faible , plus il y a d'états la satisfaisant. En effet, nous désirons pouvoir caractériser tous les états initiaux possibles qui mènent avec certitude au résultat final désiré.

Donc, de manière formelle, on peut dire que si on appelle le système S et la postcondition voulue R, alors on peut considérer la plus faible précondition correspondante comme une fonction de S et de R, appelée wp (weakest precondition) :

wp (S,R).

Si l'état initial satisfait wp(S,R), c-à-d la précondition, le système est certain d'établir finalement la véracité de R. Dans le cas contraire, il ne pourra jamais se trouver dans l'état final ( ou bien, cet état final peut ne pas correspondre à R ).

En guise de conclusion, on peut dire que le système agit comme un " transformeur de prédicats ". En effet, une fois l'exécution du système terminée, la véracité de la postcondition est établie.

Examinons à présent comment cette manière de voir les choses peut permettre de représenter le fonctionnement d'un SI.

B: les concepts "d'état" et de pré et postconditions

Dijkstra met clairement en évidence le concept d'état d'un système ; c-à-d à un momment donné l'état de tous les objets qui s'y trouvent.
La précondition d'un système caractérise l' état dans lequel doivent se trouver tous les objets afin que le système puisse démarrer.
La postcondition, quant à elle, caratérise l' état dans lequel se trouvent les objets après une exécution correcte du système.

Considérons à présent, différents systèmes ainsi que leurs pré et postconditions respectives; un système peut démarrer si sa précondition est vérifiée, de même quand celui-ci s'arrête, les états dans lesquels il laisse les objets peuvent vérifier la précondition d'un autre système qui pourra alors démarrer et ainsi de suite... .

On voit ainsi que sans qu'aucun lien explicite entre deux systèmes n'ait été défini, les pré et postconditions permettent de définir ce lien de manière implicite, nous verrons par la suite l'avantage dans le cadre d'un SIB de cette approche.

Enfin, cette manière de spécifier les systèmes permet de percevoir le fonctionnement de ceux-ci comme une suite d'états d'un SI (cfr. fig.2.3). Par opposition à l'approche par les traitements où on représente le fonctionnement du système sous la forme d'une interconnection entre les différents traitements (cfr. fig.2.4).

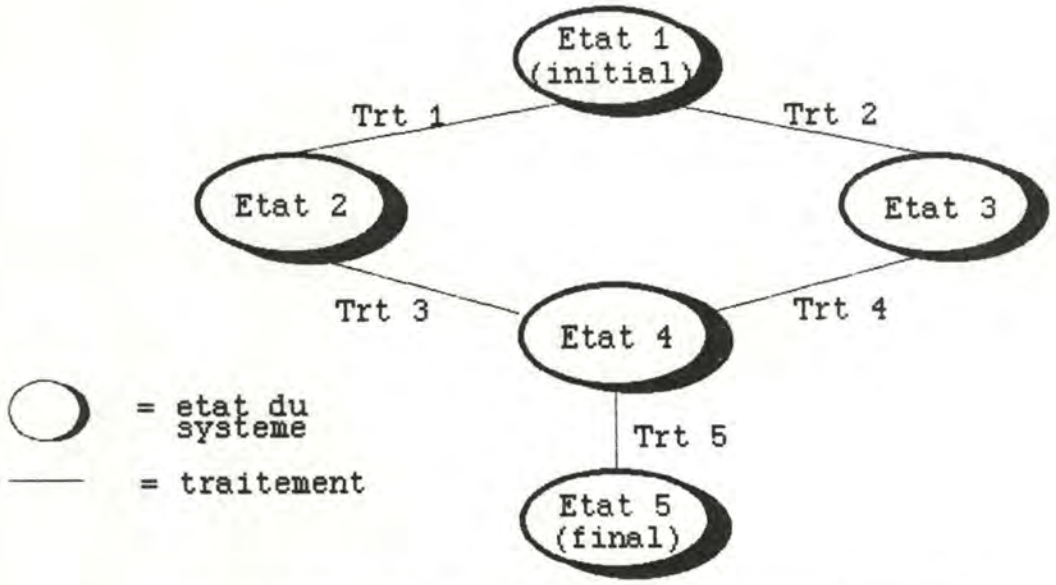



Fig.2.3. Schématisation par les données

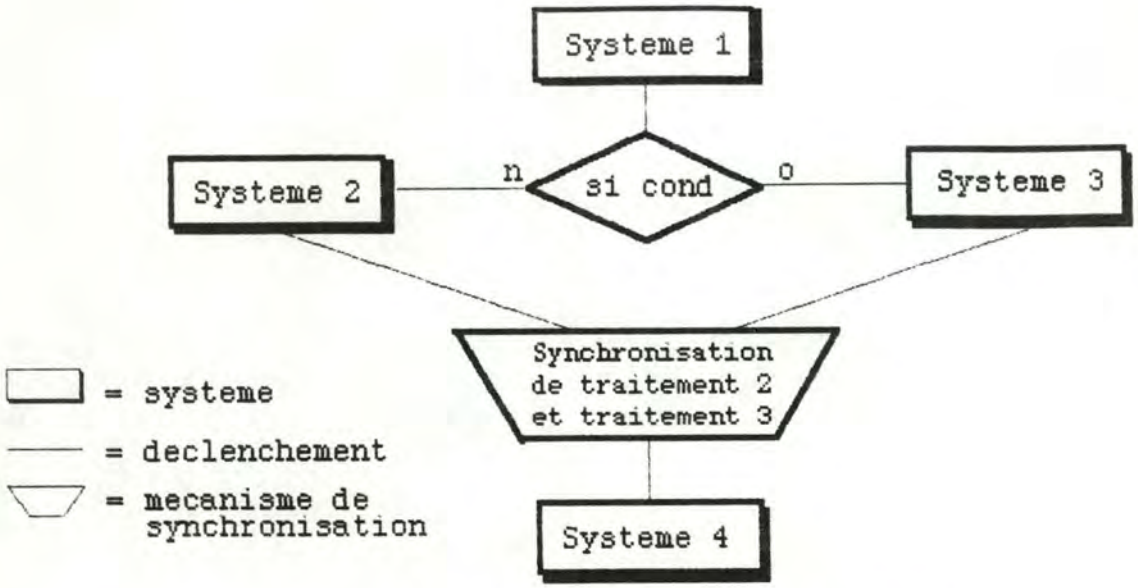



Fig. 2. 4. Schématisation par les traitements

Après avoir présenté les principes généraux de l'approche par les données, étudions quelques modèles représentatifs de cette catégorie.

## 2.4.3. SPECIFICATION DU FLUX D'OBJETS DANS UN BUREAU [TSICHRI, 8?]

### A : Introduction

Tsichritzis propose un modèle basé sur les données offrant la possibilité de modéliser l'activité de bureau au moyen de concepts aussi proches que possibles des objets réels manipulés dans le bureau.

Un SIB est, selon lui, composé de plusieurs stations de travail interconnectées; les messages circulant entre ces stations sont de tous les types : objets, procédures etc... L'automatisation d'un tel système permet aux événements se produisant d'en déclencher d'autres. Par exemple, la réception d'un message peut automatiquement déclencher une procédure qui répond à celui-ci.

Si dans un tel système, on spécifie exactement dans quelles conditions un événement peut survenir, l'utilisateur donne au système la responsabilité de causer sa survenance. Remarquons que dans ce cas le système peut ne jamais se stabiliser. En effet, si un événement cause la survenance d'un autre et ainsi de suite, le système peut indéfiniment cycler. Cette activité très parallèle peut être difficilement évaluable et comprise si on ne possède aucune vue globale de l'activité du système.

L'intérêt essentiel de ce modèle réside dans les outils qu'il offre pour décrire le comportement global du système et détecter ses propriétés anormales, à partir du modèle des données.

## B: Concepts

En résumé les concepts de ce modèle sont :

### 1. Objets

Les objets constituent tout ce que l'utilisateur manipule dans le bureau ( mémos, enregistrements, messages, procédures ). Il est possible de modéliser tout un système au moyen "d'objets". En effet, on peut considérer qu'un objet est une entité possédant un ensemble de propriétés que Tsichritzis appelle "variables" et un "comportement" qui décrit comment l'objet peut être manipulé.

Celui-ci connait donc son comportement et en est responsable. Les langages objets tels que SMALLTALK [GOL-ROB, 83] sont basés sur de tels principes; on envoit un message à l'objet, message dont le contenu permet à l'objet de déterminer quel comportement on attend de lui.

Un objet possède donc une série de "comportements" qui sont en fait des "règles" permettant sa manipulation . Ces règles peuvent être déclenchées implicitement quand l'objet remarque que leur condition de déclenchement est vraie; c-à-d quand les conditions qui portent sur les variables de l'objet sont vérifiées. Ces conditions dont la vérification est préalable au déclenchement de la règle sont des préconditions.

Si certains objets désirent communiquer au moyen de règles dont les conditions de déclenchement sont vérifiées, ils causent alors la survenance d'un événement.

## 2. Etat

On remarque qu'un objet change de valeur au cours de son existence dans une organisation.
Ces changements de valeur constituent en fait les différents états de l'objet. Certaines actions n'ont un sens que si l'objet se trouve dans un certain état; par exemple, on ne peut réparer une chaise que si celle-ci est cassée.
L'état est donc le concept qui permet de modéliser le cycle de vie d'un objet mais aussi de retenir quelles règles peuvent être actives à un moment donné.

L'état du système est déterminé par l'état dans lequel se trouvent tous les objets à un moment donné.

On peut à présent définir le cycle de vie d'un objet; c-à-d, la séquence d'états que celui-ci peut atteindre et la séquence de "règles" déclenchées d'état en état ( un état est un ensemble de valeurs qu'un objet peut atteindre ).

Exemple :

Considérons un objet "produit" dont les variables sont "inventeur" (personne qui a découvert le produit), "quantité disponible", "prix" et "date".

Son comportement consiste en les règles :
- "nouveau prix" (changer le prix du produit)
- "prélèvement" (prélever une quantité dans le disponible)
- "alpha" (création de l'objet)
- "oméga" (destruction de l'objet).

On remarque aisément qu'on ne peut déclencher certaines règles que suivant l'état de l'objet. Par exemple, on ne peut prélever que si la quantité disponible est plus grande que zéro.

Il est également possible de spécifier qu'un objet ne peut exécuter une de ces règles que si cela lui est demandé par un autre objet privilégié; par exemple seul l'objet "directeur" a le droit de demander à l'objet "produit" d'exécuter la règle "nouveau prix".

## 3. Faisabilité et comportement global

La description d'un système au moyen des objets laisse certains problèmes en suspens.
Par exemple, on ne connait jamais à priori tous les objets participants à la survenance d'un événement.
En effet, dans le cas où un objet souhaite communiquer avec un autre, il invoque une "régle" de celui-ci. Cette règle peut, de la même manière, communiquer avec un troisième objet pour pouvoir s'exécuter correctement.
En conséquence, la spécification du comportement global pourrait ne pas correspondre à ce que nous voulions au départ.

Dans ce cas, un processus automatique d'analyse semble requis, afin de donner à l'utilisateur une vue globale du comportement de son système.
Ceci peut se réaliser au moyen des "expressions de déclenchement" ( firing expressions ) qui représentent l'ensemble de toutes les séquences possibles de déclenchement de règles et de changement d'état pour un objet.
Ces expressions sont utiles pour générer des expressions de flux.

Par exemple, examinons le cas d'un objet document dont une des variables est propriétaire, cette variable indique qui peut modifier ce document. Dans ce cas, certains états de celui-ci correspondront à sa possession par une station de travail donnée. L'expression de déclenchement qui permet de connaître tous les changements d'états d'un objet pourra alors être utilisée pour déterminer quelles séquences de stations peuvent posséder l'objet, c-à-d quel est le flux de l'objet dans le réseau.

### C : Evaluation

Nous retiendrons de ce modèle que, bien que orienté données, il permet une estimation du comportement global du système.
Il montre également l'importance :

- du concept "d'état" d'un objet dans ce genre d'approche
- des expressions permettant de caractériser les flux d'information.

Examinons à présent comment les modèles orientés bases de données conçoivent le fonctionnement des SI.

## 2.4.4. LA SPECIFICATION DU COMPORTEMENT DE SYSTEMES ORIENTES BD

### A : Concepts

Brodie et Silva [BROD-SIL, 82] insistent sur le fait que peu de concepts ou d'outils existent pour modéliser le comportement d'un SI, et intégrer comportement et structure, alors que la conception des bases de données exige une telle démarche.

La modélisation du comportement d'un SI proposée par Brodie et Silva est basée sur les axiomes et les techniques de transformation de prédicats, et est réalisée au moyen de contraintes exprimées par ces prédicats. Ceci permet tout à la fois l'abstraction et la précision.

Nous ne nous intéressons qu'à la façon de spécifier le comportement ou plus précisément les "transactions" de systèmes orientés BD.
Une transaction est un traitement qui constitue la réponse à un besoin de l'utilisateur. Elle se décompose en "actions" qui sont en fait des primitives sur les objets de la base de données.

Examinons comment Brodie et Silva proposent de spécifier les transactions.

### B : Méthode

On spécifie une transaction de la manière suivante :

" N (V) I, L: P{A} Q ", où :

N est le nom de l'opération,

V est la liste de paramètres,

I,L définissent la portée de l'opération en nommant :

I : les objets accédés par les actions et les accès BD

L : les objets locaux à l'opération

P,A,Q spécifient l'effet de l'opération :

P : est la pré-condition sous forme d'une liste de prédicats

A : est le role actif (0,1 ou plusieurs actions)

Q : est la postcondition sous forme d'une liste de prédicats.

I et L définissent la portée de la transaction; P est la pré-condition qui détermine quand A peut s'exécuter . A est une suite d'actions dont le but est de réaliser les objectifs de la transaction qui peuvent être ensuite vérifiés par la post-condition Q.

Remarquons que les effets de la transaction sont complètement définis par P{ }Q.
A, qui définit comment les effets sont réalisés, est un détail inapproprié pour la spécification.

Exemple :

Supposons que l'on désire spécifier une transaction "enregistrer_pièce" qui permette d'enregistrer une nouvelle pièce dans le stock. Dans le langage proposé par Brodie, cela se fait de la manière suivante :

```
TRANSACTION enregistrer-pièce (p)
(* scope permet de définir la portée de la transaction *)
            SCOPE (p:pièce)
            PRECONDITION pièce_n'existe_pas(p) ?
            POSTCONDITION piéce_existe(p) ?

pièce-n'existe_pas : NO p IN pièce
pièce_existe  :  p  IN  pièce
```

C : Evaluation
---

Les effets d'une opération sont définis en termes de simples prédicats dans P et Q qui peuvent être précis, abstraits, et formels.

Ce modèle dans sa démarche de spécification des traitements fait usage de pré et postconditions portant sur les objets contenus dans la base de données. Une telle méthode permet de spécifier indépendamment l'un de l'autre les différents traitements et garantit l'indépendance vis à vis du scénario d'exécution de ceux-ci.

## 2.4.5. LE MODELE "REMORA" BASE SUR LES CATEGORIES

### A: Concepts

Le modèle REMORA présenté dans [ROLLAND, 82] est basé sur les éléments observés dans une organisation. Ces éléments sont regroupés en trois catégories :

#### 1. La catégorie OBJET

les objets sont des constituants abstraits ou concrets de l'organisation étudiée, durables dans leur spécificité. A chaque objet est associé le concept d'état; ceci permet de spécifier la "dynamique" de l'objet en décrivant les causes des différents changements d'état. L'état de l'organisation est défini par l'état de tous les objets qui s'y trouvent.

#### 2. La catégorie OPERATION

La catégorie OPERATION reprend toutes fonctions que l'on peut effectuer sur les objets.

L'exécution des opérations a pour résultat soit la création de nouveaux objets, soit la modification d'objets déja existants. Ceux-ci ne disparaisssent jamais du SI, ils changent simplement d'état ce qui permet d'intégrer l'aspect temporel dans le cycle de vie des objets en mémorisant les dates de changement d'état.

#### 3. La catégorie EVENEMENT

Un événement est tout ce qui peut survenir dans le SI et déclencher l'exécution des opérations. Un événement est donc soit lié au temps, soit lié au changement d'état d'un objet.

Le fonctionnement d'un SI est décrit par trois types d'interactions entre les catégories que nous venons de voir :

MODIFIE (opération, objet) : exprime la modification d'un objet par l'exécution d'une opération

CONSTATE (objet, événement) : la constatation du changement d'état d'un objet par un événement

DECLENCHE (événement, opération) : le déclenchement d'un opération par un événement.

Tout ceci est exprimé par la figure suivante.

Celle-ci montre qu'un EVENEMENT CONSTATE le changement d'état d'un OBJET et DECLENCHE l'exécution d'OPERATIONS qui MODIFIENT l'état d'autres objets provoquant ainsi parfois de nouveaux événements (cfr.fig.2.5).

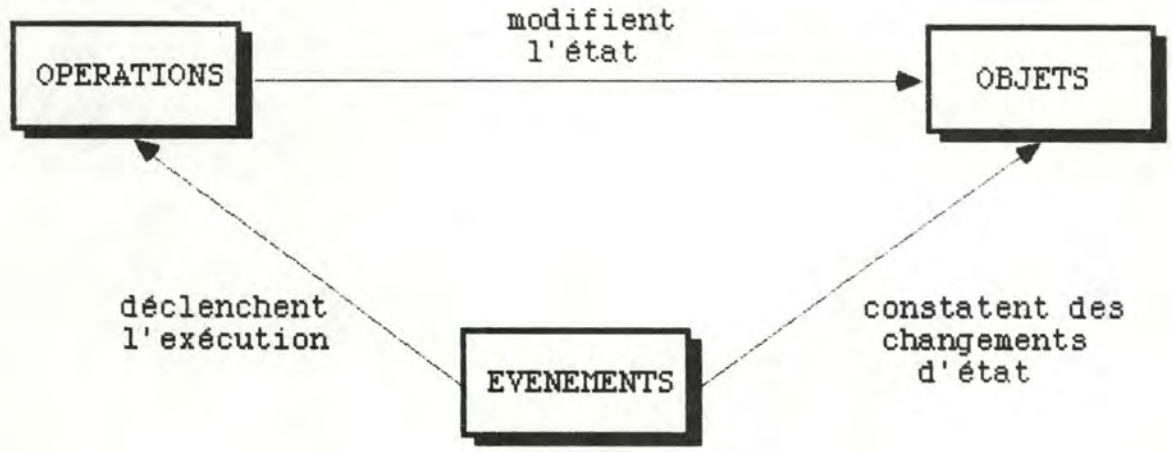



Fig. 2.5.

B: Evaluation

Nous retenons de REMORA que :

- Dans un même modèle, on retrouve la spécification de la structure et du comportement.

- il permet une modélisation du cycle de vie des objets grâce au concept d'état.

## 2.4.6. CONCLUSION DE L'APPROCHE CENTREE SUR LES DONNEES

En guise de conclusion de l'approche centrée sur les données, voyons quels en sont les avantages et les inconvénients dans le cadre de notre étude.

### Désavantages

Tout d'abord, la modélisation du fonctionnement ne permet pas à priori de voir l'activité globale du SI décrit.

Ensuite, puisque la modélisation du fonctionnement repose entièrement sur les objets et leurs états respectifs, il faut préalablement à toute modélisation de fonctionnement connaître la modélisation de la structure du SI.

### Avantages

Les différents traitements sont uniquement modélisés par rapport aux objets informationnels. Ils peuvent donc être conçus et spécifiés indépendamment les uns des autres (c-à-d : sans qu'aucune interaction préalable entre eux ne doive être connue). Ceci permet de concevoir le fonctionnement sans se soucier d'un scénario précis d'exécution.

Le flux de contrôle n'est pas apparent mais caché dans les conditions (portant sur les objets) qui doivent être vérifiées pour qu'un traitement puisse s'exécuter. Ce flux est modifiable aisément en changeant ces conditions car les répercussions d'un changement sont locales au traitement.

Après avoir présenté et illustré par des exemples les deux approches générales de modélisation du fonctionnement des SI, il reste à situer notre travail par rapport à ces deux courants littéraires.

## 2.5. Notre approche

Tous les modèles présentés, qu'ils soient centrés sur les traitements ou sur les objets, existent et présentent certains avantages et inconvénients suivant le contexte dans lequel on les utilise.

Nous allons uniquement privilégier une approche en fonction des besoins particuliers auxquels doit satisfaire notre modèle de spécification du fonctionnement d'un SIB.

Il semble normal que l'approche que nous allons choisir dans notre démarche de conception d'un modèle de comportement pour un SIB doit respecter la caractéristique essentielle dont nous avons parlé dans l'introduction, à savoir, l'aspect local des tâches de bureau.
Nous souhaitons, dans ce but, disposer d'un modèle permettant de spécifier chacune des tâches indépendamment l'une de l'autre.

De plus, l'activité principale des tâches de bureau est de traiter de l'information, ou plutôt les objets informationnels qui véhiculent celle-ci. En conséquence, les tâches reçoivent en entrée des objets dans un certain état et produisent en sortie ces mêmes objets ou d'autres dans un état différent.

Ceci nous oblige à écarter l'approche par les traitements pour deux raisons;
tout d'abord, elle préconise une interconnection entre les différents traitements au niveau contrôle ( c-à-d : séquentialisation des traitements ). Cependant cette séquentialisation est à priori ignorée par le concepteur d'une application de bureau.
La seconde raison est que les modalités de déclenchement des tâches de bureau ne portent que sur les objets informationnels.

Par contre, une modélisation par les objets, combinée à la spécification par pré et post conditions, permettra de rencontrer :

- "l'approche locale" des traitements (car les flux d'information et de contrôle seront cachés dans les pré et postconditions)
- le fait que ces traitements ne manipuleront que des objets informationnels.

Etant donné l'approche envisagée, le fonctionnement global du SI sera décrit à partir du fonctionnement de chacun des traitements qui le composent. Il n'est donc pas visible à priori. Ceci constitue le défaut de notre choix.

Pour y remédier et afin que la modélisation choisie soit la plus cohérente et la plus adéquate possible en fonction des ressources dont une organisation dispose, nous définirons un langage formel (cfr. chapitre III et IV) et implémenterons un outil de simulation.

Pour ce faire nous devons passer de notre représentation à une autre où le fonctionnement global est visible, en l'occurence le modèle de Bodart et Pigneur. Il nous faut donc concevoir et implémenter un processus de traduction de notre langage dans celui de Bodart et Pigneur afin de pouvoir utiliser leur simulateur (cfr. chapitre VI et VII).
Ce simulateur fournira au concepteur un moyen d'évaluer la cohérence et la faisabilité de sa modélisation.

Etudions à présent dans le chapitre suivant les principes de notre langage.

# CHAPITRE III

# CONCEPTS MIB

# ET

# APPROCHE DES PRE/POSTCONDITIONS

## 3.1. Introduction

La première partie de ce chapitre va présenter les concepts MIB (Modèle de SIB ) qui sont nécessaires à la compréhension de ce travail.
En effet, notre modèle d'expression du comportement s'inscrit dans le cadre du modèle descriptif MIB tel qu'il a été défini et implémenté par [DACHOUF, 86], auquel le lecteur se référera s'il souhaite plus de détails.

La seconde partie de ce chapitre montre comment il est possible de spécifier le fonctionnement d'un SIB au moyen de pré et postconditions.

## 3.2. Le MIB

### 3.2.1. Description

Le projet MIB, ( Modèle de système d'Information de Bureau ) dont le fondement théorique est le "diamant de Leavitt" (cfr introduction 1.1), est un ensemble de modèles et d'outils.
Sa particularité est d'être dédicacé à la spécification des S.I.B. ainsi qu'à la description et l'évaluation de leur comportement.

MIB comprend trois composants :

- une base de données permettant d'enregistrer les spécifications des utilisateurs concernant leurs S.I.B.
- un langage, MIBLangage, qui permet à l'utilisateur d'enregistrer ses spécifications.
- un ensemble d'outils automatisés qui sont de deux types:
  - < des outils génériques de documentation permettant d'extraire en tout ou en partie les spécifications des utilisateurs
  - < des outils spécifiques de simulation et d'aide à la conception par animation graphique [VANPEVE, 87].

### 3.2.2 Les modèles proposés par le MIB

MIB propose au concepteur d'applications de bureau toute une série de modèles orientés vers le travail de bureau.
On trouvera les schéma entités/associations des modèles en annexe, nous reprenons simplement ici la description des objets qui ont le plus d'intérêt pour notre étude.

## A. Le modèle de structuration de l'organisation :

il permet de définir la facette "structure" du diamant de Leavitt, c'est-à-dire l'organigramme de l'organisation, les responsabilités et le rôle des différentes personnes dans la réalisation d'un but fixé.

Nous retiendrons de ce modèle que l'information circulant dans le système est originaire des différents composants de la structure (PERSONNE ou UNITE ORGANISATIONELLE), et que ces composants sont soit les responsables, soit les exécutants des différentes TACHES à accomplir.

### Entité UNITE ORGANISATIONNELLE

#### Définition

Une organisation ( qui est l'unité organisationnelle la plus importante ) est un groupe permanent de personnes réunies afin de poursuivre un objectif spécifique.

L'organisation est divisée en unités organisationnelles. Chacune a un responsable ainsi que des membres

Une unité organisationnelle peut :
- soit être décomposée en d'autres unités organisationnelles
- soit être constituée de personnes qui en sont les membres.

On distinguera une unité organisationnelle particulière : l'environnement de l'organisation.

### Entité PERSONNE

#### Définition

Toute personne physique est une PERSONNE.
Chaque personne joue un rôle dans l'organisation. En bureautique, on peut distinguer 4 types d'acteurs en fonction des rôles : administrateur, professionnel, clerc, sécrétaire.

Entité TACHE

Définition

Une tâche remplit un objectif, une fonction. C'est une suite d'actions élémentaires à exécuter dans un ordre convenu pour résoudre un problème.

Deux attributs intéressants sont à mentionner : ce sont les attributs qui permettront d'exprimer les pré et postconditions de la TACHE.

Ces attributs sont de type "texte libre" pour un utilisateur restant à un niveau descriptif du fonctionnement de son SI, mais devront respecter la syntaxe et la sémantique du langage que nous allons définir au chapitre suivant, si l'utilisateur désire évaluer par simulation sa modélisation.

- PRE-TACHE

Condition qui doit expliciter les propriétés des arguments qui doivent être vérifiées avant toute exécution de la tâche pour que celle-ci s'exécute correctement. Ces propriétés porteront sur les objets informationnels manipulés par la tâche, ainsi que sur l'ETAT dans lequel ceux-ci se trouvent.

- POST-TACHE

Condition qui doit expliciter les propriétés des résultats de la tâche qui doivent être satisfaites à la fin de toute exécution si celle-ci s'est exécutée correctement.

## B. Le modèle de structuration de l'information :

il permet de définir l'information traitée dans le bureau. Les TACHES manipulent de l'information et la représentation de cette information se fait au moyen de six objets. Ces six objets sont : le MESSAGE, le DOCUMENT, le FORMULAIRE, le DOSSIER, le FICHIER, la PILE.

MESSAGE, DOCUMENT et FORMULAIRE sont des objets élémentaires.

DOSSIER, FICHIER et PILE sont des objets structurants qui permettent d'organiser l'information, ce qui signifie qu'ils peuvent contenir des objets simples.

Les TACHES reçoivent des objets en entrée qu'elles modifient ou dont elles se servent pour en générer d'autres.

Les objets sont caractérisés par des attributs qui sont de deux types : statiques et dynamiques.

- Les attributs statiques décrivent les caractéristiques de l'objet qui ne changent pas au cours de la vie de l'objet.
  ex : le TYPE de l'objet ... etc,
  un objet MESSAGE de TYPE = mess-télex.

- L'attribut dynamique essentiel considéré ici est l'ETAT d'un objet. Cet attribut peut évidemment varier durant la vie de l'objet et permet de décrire son évolution dynamique.
  ex : un objet MESSAGE dont l'ETAT peut évoluer de "reçu" à "consulté" puis à "traité".

En bref, l'état est modifié lorsqu'une tâche utilise l'objet auquel il est associé.

Voici la description des entités intéressantes de ce modèle :

Entité TACHE

C'est celle décrite dans le modèle organisationnel.

Entité OBJET

L'entité représentant un objet informationnel sera mise en relation avec les objets du modèle organisationnel afin de décrire l'origine et la destination de l'information dans l'organisation.

On distingera parmi les entités objets :

le MESSAGE : c'est une information sous forme écrite ou orale, qui a les caractéristiques suivantes :
- elle est en général brève;
- le corps n'obéit à aucune structure particulière;
- le moyen de communication utilisé est généralement synchrone ( téléphone, réunion ).

le DOCUMENT : c'est une information spécifique, généralement représentée sous forme écrite, graphique ou mixte. Les documents demandent à être produits, sont généralement classés ou archivés et leur distribution se fait selon des moyens de communication asynchrones ( poste interne ou externe ).

le FORMULAIRE : c'est une information écrite standardisée. Il est composé d'un squelette textuel et d'éléments auxquels peuvent être associés des ensembles de valeurs.

le DOSSIER : c'est un groupe de plusieurs types différents de formulaires, messages, documents, reliés entre eux par une relation logique.

le FICHIER : c'est un groupe d'occurrences d'un même type de formulaires (quelquefois de messages ou de documents), univoquement idendifiés et arrangés selon un certain ordre (alphabétique, chronologique). Tout fichier porte un identifiant.

la PILE : c'est un groupe d'occurrences de divers types de documents, messages, formulaires, sans lien logique entre eux, sauf généralement l'ordre chronologique (mais cet ordre n'est pas intentionnel).
Ce sont des informations en attente de classement, de traitement.

### C. Le modèle des ressources :

il décrit les facettes "technologie" et "personnes" du diamant de Leavitt.

Nous retenons de ce modèle les ressources nécessaires à l'exécution des tâches de bureau. Ces ressources sont : les PERSONNES, les RESSOURCES CONSOMMABLES, et la TECHNOLOGIE. Celles-ci sont REQUISES à un certain TAUX par les TACHES et sont disponibles en une certaine quantité (DISPONIBILITE).

Ce modèle n'intervient pas dans la spécification du fonctionnement d'un SI mais est utile pour évaluer la faisabilité de celui-ci.

Entité TACHE

C'est celle décrite dans le modèle organisationnel.

Entité TECHNOLOGIE

Définition

C'est l'ensemble du matériel (informatique ou non) et des logiciels qui exécutent totalement ou partiellement une tâche.
La technologie est une ressource réutilisable.

Entité PERSONNE

Définition

Ici, on considère les personnes en tant que facteur de production et non plus comme membre de l'organisation.
La personne est une ressource réutilisable.

Entité RESSOURCE-CONSOMMABLE

Définition

C'est une ressource qui n'est pas réutilisable lorsqu'elle a été consommée lors de l'exécution d'une tâche.
Exemple : papier, énergie, argent.

### D. Le modèle de structuration des traitements :

il permet une décomposition des traitements en différentes tâches et une découpe élémentaire des tâches en opérations primitives. Il décrit donc la facette "tâches" du diamant de Leavitt.

Ce modèle n'est pas utilisé dans notre étude.

E. Le modèle de la dynamique des traitements :

il permet de décrire deux types de fonctionnement; tout d'abord celui que nous envisageons dans ce travail qui est le problème des enchaînements entre tâches, et ensuite pour chaque tâche, le problème de sa décomposition en opérations primitives [VANPEVE, 87].

## 3.3. Les Pré-Post conditions

Puisque nous avons choisi de spécifier le comportement d'un SI à l'aide de pré et postconditions, il convient à présent d'expliquer comment cette approche peut s'appliquer au système d'information de bureau qui nous intéresse.

Nous nous intéressons à un système d'information qui évolue, passant d'un ETAT à un autre.

L'état du SIB à un moment donné est caractérisé par l'état des objets qu'il manipule, en l'occurrence, les objets informationnels du MIB. Comme l'aproche choisie est orientée objets, ce sont ces objets informationnels ainsi que l'état dans lequel ils se trouvent qui vont déterminer quelles tâches vont pouvoir être exécutées.

Les événements de type "calendrier" auront également été intégrés dans le modèle des données sous forme d'objets informationnels.

ex : un événement tel que la rentrée académique peut par exemple être spécifié dans le MIB comme un objet informationnel MESSAGE de type "rentrée-académique".

De plus, notre modèle de fonctionnement doit permettre de spécifier chaque tâche de manière indépendante l'une de l'autre. Ceci peut être modélisé dans le MIB en complétant les attributs des tâches, PRE-TACHE et POST-TACHE dont le contenu porte sur les objets informationnels et leur état.

Ces pré et postconditions dont le format est un texte libre tant que l'utilisateur reste à un niveau purement descriptif, devront être exprimées dans un langage formel lorsqu'on voudra évaluer le comportement d'un SIB.

Examinons à présent de manière plus précise en quoi consistent ces pré et postconditions et de quelle manière elles vont permettre l'agencement des tâches lors de l'exécution d'un programme de simulation.

### 3.3.1. La précondition

Une précondition est une condition qui doit être vérifiée avant toute exécution de la tâche à laquelle elle est attachée.

L'expression de cette condition est un prédicat portant sur

- les objets informationnels (cfr.3.2.2.B) que la tâche va manipuler,

  les objets simples (document, formulaire, message)
  et / ou
  les objets structurants (pile, dossier, fichier)

- l'état de ces objets (attribut qui ne peut être modifié que lors de l'exécution d'une tâche et qui retrace donc l'évolution dynamique de l'objet).

exemple :

si la spécification d'une tâche est de classer un formulaire de demande d'inscription (préalablement vérifié) dans un fichier (préalablement ouvert) reprenant les demandes d'inscription de tous les étudiants, la précondition à l'exécution de cette tâche sera la suivante :

il faut que . le FORMULAIRE de type "demande d'inscription" soit dans l'état "vérifié"

et que . le FICHIER reprenant toutes les inscriptions (de type "inscriptions") soit dans l'état "créé".

### 3.3.2. La postcondition

Une POSTCONDITION est une condition explicitant les propriétés des résultats de la tâche, qui doivent être satisfaites à la fin de toute exécution de cette tâche.

Cette condition sera elle aussi exprimée par un prédicat portant sur les objets informationnels du MIB et l'état dans lequel ils se trouvent après qu'ils aient été éventuellement modifiés par l'exécution de la tâche.

Exemple :

La tâche "classer un formulaire" aura pour postcondition :

. le FICHIER de type "inscriptions" est maintenant dans l'état "modifié"

et

. le FORMULAIRE de type "demande d'inscription" est dans l'état "classé dans fichier".

Si la précondition d'une tâche est vérifiée et que l'exécution de cette dernière s'est effectuée correctement, alors la postcondition est également vérifiée, de par sa définition.

### 3.3.3. L'enchaînement

Le concept d'enchaînement permet de représenter l'agencement des tâches lors d'une exécution par simulation du fonctionnement du SI décrit.

Un enchaînement pourra s'effectuer lorsque la précondition d'une tâche sera "contenue" dans la postcondition d'une ou de plusieurs autres, c'est-à-dire :

> si dans la précondition d'une tâche, toutes les conditions ( portant sur les objets informationnels dans un certain état qu'elle va manipuler ) correspondent exactement aux résultats d'une ou plusieurs autres tâches.

Il est nécessaire que la condition d'une précondition et le résultat d'une postcondition se correspondent exactement, c'est-à-dire, qu'ils portent tous les deux sur un même objet informationnel dans le même état.

Une condition d'une précondition est "contenue" dans une postcondition si elle correspond exactement à un des résultats de la tâche à laquelle cette dernière est attachée.
L'enchaînement sera effectué lorsque toutes les conditions de la précondition correspondront exactement à l'un ou l'autre résultat apparaissant dans l'une ou l'autre postcondition.

En effet, si une tâche "classer demande inscription" a pour postcondition que :

> . le FICHIER de type "inscription " est dans l'état "mis à jour"
>
> et . le FORMULAIRE de type "demande d'inscription" est dans l'état "classé dans fichier";

et si on considère une tâche "consulter demande inscription" dont la précondition spécifie que

> . un FORMULAIRE de type "demande d'inscription" doit être dans l'état "classé dans fichier"

on voit que la deuxième tâche va pouvoir s'exécuter après la première.

Si la tâche "classer demande inscription" s'est correctement exécutée, sa postcondition est vérifiée. Or, la tâche "consulter demande inscription" reconnait qu' une des propriétés des résultats de "classer demande inscription" correspond à sa propre précondition, elle peut donc s'exécuter.

Ce processus peut être généralisé à l'ensemble des tâches constituant l'activité (cfr.1.5.C).

Remarquons que le flux décrit peut aisément être modifié en changeant uniquement la précondition de "consulter demande inscription" et que la manière de spécifier le fonctionnement possible est bien locale à une tâche, puisque celle-ci connait elle-même ses conditions de déclenchement.

Cette manière de décrire le fonctionnement d'un système permet également d'introduire la notion de parallélisme.

En effet, si une tâche T1 s'est exécutée correctement, sa postcondition est vérifiée.

Considérons maintenant une tâche T2 dont la précondition correspond à une propriété des résultats de T1,
et une tâche T3 dont la précondition correspond également à une des propriétés des résultats de T1 ;
T2 et T3 pourront être exécutées en parallèle, puisque leurs préconditions respectives sont toutes deux vérifiées.

Si les traitements effectués en parallèle modifient l'état d'un même objet, il est possible qu'une autre tâche veuille travailler sur cet objet à l'issue du traitement en parallèle.
On retrouvera donc dans la précondition de cette dernière tâche, une condition composée portant sur l'état de l'objet en question.

exemple :

Si une tâche de "vérification_correct" a pour postcondition :
- le FORMULAIRE de type "demande d'inscription" est dans l'état "correct"

si une tâche de "vérification_complet" a pour postcondition :
- le FORMULAIRE de type "demande d'inscription" est dans l'état "complet",

en admettant que ces deux types de vérification peuvent s'exécuter en parallèle pour un même formulaire,

on peut spécifier qu'une tâche de "classement_des_formulaires" ne peut être exécutée que si elle dispose d'un formulaire de demande d'inscription qui ait subi les deux contrôles, c'est-à-dire, qui soit non seulement complet, mais aussi correct.

On retrouvera dans la précondition de cette tâche :
. le FORMULAIRE de type "demande d'inscription" est dans l'état "complet" ET "correct".

Il peut sembler étrange que des tâches différentes travaillent en concurrence sur un même objet, mais c'est néanmoins réalisable lorsque les traitements effectués sur les données sont automatisés.

## 3.4. Conclusion

Après avoir rappelé le modèle MIB, ce chapitre a exposé de manière informelle comment exprimer le fonctionnement d'un SIB au moyen de pré et postconditions. Nous avons vu de manière générale leur contenu et la signification de celui-ci.

Nous allons à présent définir un langage formel d'expression de ces conditions qui permettra au concepteur d'une application de bureau de décrire le fonctionnement de son système et par la suite d'évaluer sa modélisation.

# CHAPITRE IV

# DESCRIPTION DU LANGAGE

## 4.1. Introduction

Nous avons vu de manière informelle comment il était possible de spécifier le fonctionnement d'un SIB au moyen de pré et postconditions.

Ce chapitre se propose d'exposer la syntaxe du langage attaché à notre modèle de représentation de la dynamique sous forme de pré et postconditions. Ce langage sera un sous-ensemble du langage MIBL (cfr. chapitre III 2.2.1.).

En effet, si l'on veut simuler le comportement du SI à partir des spécifications conceptuelles, il est indispensable de formaliser l'expression de ces conditions.
L'utilisateur souhaitant décrire, mais surtout dans ce cas, évaluer une activité de bureau doit spécifier le comportement dynamique des différentes tâches qui la composent : c'est-à-dire donner à leurs pré et postconditions un contenu syntaxiquement correct.

De plus, le langage d'expression de la dynamique doit être clair et facile à utiliser. Nous devons, pour ce faire, éliminer des pré et postconditions les caractéristiques "techniques" relatives à l'outil qui permettra l'évaluation (valeur probabilistique des conditions, valeurs des paramètres ... etc).

Nous allons d'abord préciser les concepts et la terminologie employés, ensuite la syntaxe et la sémantique des pré et postconditions seront étudiées de façon progressive. Après avoir modélisé le comportement fonctionnel proprement dit, nous examinerons comment on peut quantifier ces conditions afin qu'une modélisation de fonctionnement puisse être évaluée par simulation.

On trouvera en annexe un document récapitulatif de la syntaxe du langage.

## 4.2. Expression générale du langage

Rappelons que nous nous intéressons à un système d'information qui se trouve dans un état initial et qui doit atteindre un certain état final après l'exécution d'une ou plusieurs tâches.

L'expression de cet état initial est en fait une précondition à l'exécution de la tâche et décrit l'état du SI à un moment donné.

L'expression de l'état final est une postcondition puisqu'elle impose une condition sur l'état dans lequel le système doit se trouver après une activité.

Après avoir montré que les pré et postconditions expriment toutes deux une condition il est nécessaire de préciser la terminologie utilisée dans ce contexte.

### 4.2.1. Terminologie

Un PREDICAT est l'expression formelle d'une condition. On peut dire, par exemple, que les deux prédicats "X = Y" et "Y = X" expriment la même condition.

Un prédicat est toujours défini : sa valeur est à tout moment soit "vrai", soit "faux" et le PREDICAT est utilisé pour caractériser l'ensemble des états pour lesquel il a la valeur "vrai".

Les pré et postconditions sont des prédicats.

## 4.2.2. Expression d'une précondition

L'expression d'une précondition peut se faire au moyen de prédicats simples ou composés.

### A : PREDICAT SIMPLE

Un prédicat simple exprime une condition portant sur un objet informationnel simple ( document, message, formulaire) ou structurant ( fichier, dossier, pile ).

Il dénote le fait que cet objet doit se trouver dans un certain état.

Un même objet peut se trouver dans un état donné et également dans un autre état, uniquement s'il a atteint ces différents états lors de traitements effectués en parallèle. Cette introduction du parallélisme dans les traitements est due à l'automatisation des SIB.

Les différents états seront séparés par le symbole "ET".
Les symboles '(' et ')' sont les séparateurs de prédicats, le nombre de parenthèses ouvrantes est toujours égal au nombre de parenthèses fermantes.

Exemple :

Pour exprimer qu'un DOCUMENT de type "lettre-contact" doit être dans l'état "complet" et dans l'état "correct" avant toute exécution d'une tâche donnée, on spécifiera la précondition de cette tâche de la manière suivante :

( DOCUMENT lettre-contact : complet ET correct )

Le symbole ":" signifie "est dans l'état".

### B : PREDICAT COMPOSE

Un prédicat composé est une composition

- de prédicats simples ou composés

avec

- des opérateurs ET, OU.

Les opérateurs ET et OU sont binaires et utilisés en notation préfixée, ce qui permet de déterminer les priorités d'évaluation des opérateurs.

Un prédicat composé contenant un opérateur ET est encore un prédicat qui aura la valeur "vrai" quand les deux prédicats qui le composent ont également la valeur "vrai".

Un prédicat composé contenant un opérateur OU est encore un prédicat qui aura la valeur "vrai" quand au moins un des deux prédicats qui le composent a la valeur "vrai".

Si PS1, PS2 et PS3 sont des prédicats simples, PC1 ET PC2 des prédicats composés, on peut exprimer :

```
PC1 = ET
          PS1
          PS2
```

Ceci signifie que PC1 aura pour valeur "vrai" si PS1 est "vrai" et PS2 est "vrai" également.

De même, on aura

```
PC2 = OU
          PC1
          PS3
```

équivalent à

```
PC2 = OU
          ET
               PS1
               PS2
          PS3
```

encore équivalent à

```
PC2 = OU
          PS3
          ET
               PS1
               PS3
```

Exemple :

Si on veut exprimer dans la précondition d'une tâche qu'elle sera effectuée si on a :

. SOIT un document de type "lettre-contact" dans l'état "reçu"
. SOIT un message de type "entretien-contact" dans l'état "reçu"
AINSI QUE
un fichier de type "contact" dans l'état "créé",
on écrira :

OU
( DOCUMENT lettre-contact : reçu )
ET
( MESSAGE entretien-contact : reçu )
( FICHIER contact : créé )

### 4.2.3. Expression d'une postcondition

L'expression d'une postcondition peut se faire au moyen de prédicats simples ou composés.

A : PREDICAT SIMPLE

Tout comme pour une précondition, un prédicat simple exprime une condition portant sur un objet informationnel simple (document, message, formulaire), ou structurant ( fichier, dossier, pile ).

Ce prédicat dénote le fait qu'après exécution de la tâche à laquelle est rattachée la postcondition, les objets informationnels doivent se trouver dans un certain état.

### A.1 les objets simples

Un prédicat simple portant sur un objet informationnel simple s'exprime comme une condition sur l'état de cet objet.

Exemple :

Pour exprimer qu'un DOCUMENT de type "lettre-contact" doit être dans l'état "traité" après l'exécution d'un traitement, on notera :

( DOCUMENT lettre-contact : traité )

### A.2 Les objets structurants

Une postcondition peut également porter sur le fait que la tâche a mis à jour un objet structurant (FICHIER, PILE, DOSSIER) au moyen d'un autre objet.

On peut effectuer trois types de mise à jour :

- MAJ+ = ajout d'un objet à un objet structurant (ex : ajouter un DOCUMENT à une PILE )
- MAJ- = suppression d'un objet de l'objet structurant qui le contenait (ex : supprimer un MESSAGE d'un DOSSIER )
- MAJ = modification d'un objet contenu dans un objet structurant.

On peut également émettre une condition sur l'état dans lequel se trouve un objet structurant :

Exemple :

( DOSSIER étudiant : créé )

Ce prédicat est vérifié si le dossier de type "étudiant" est dans l'état "créé".

Un prédicat simple portant sur un objet structurant offre les possibilités suivantes :

1 - l'objet structurant change SIMPLEMENT d'état
ex : ( FICHIER refus : en-mise-à-jour )

2 - l'objet structurant ne change pas d'état et est mis à jour par un objet
ex : ( FICHIER acceptation MAJ
PAR DOCUMENT lettre-contact : accepté )

3 - l'objet structurant change d'état et est mis à jour par un objet
ex : ( FICHIER inscription : modifié MAJ-
PAR FORMULAIRE demande : refusé )

Ceci exprime le fait que le fichier de type "inscription " est maintenant dans l'état "modifié" et que l'on y a supprimé un formulaire de type "demande" qui est dans l'état "refusé".

Tout objet se trouvant dans une postcondition a été manipulé par la tâche et a donc changé d'état.

On trouve cependant une exception à cette règle :

dans le cas (2-) ci-dessus, lorsque l'objet structurant a été mis à jour sans changer d'état
ex : ( FICHIER acceptation MAJ+
PAR DOCUMENT lettre-contact : accepté )

L'état du fichier "acceptation" n'a pas été modifié par la tâche. Il peut sembler anormal qu'un objet mis à jour par un autre ne change pas d'état.
C'est cependant possible si l'utilisateur spécifie (par exemple) qu'un fichier ne peut passer que dans deux états ("créé" et "clôturé"); dans ce cas à partir du moment ou le fichier est "créé", il ne changera plus d'état (même s'il est mis à jour) avant d'être "clôturé".

Ceci évite à l'utilisateur de prévoir un état différent pour chaque mise-à-jour effectuée sur l'objet structurant.

B : PREDICAT COMPOSE
---

Tout comme pour les préconditions, un prédicat composé est une composition

- de prédicats simples ou composés

avec

- des opérateurs ET, OU.

Les opérateurs ET et OU sont binaires et utilisés en notation préfixée.

### B.1 L'opérateur ET dans une postcondition

Un ET dans une postcondition exprime la coordination entre des résultats d'un traitement.

### B.2 L'opérateur OU dans une postcondition

Un OU dans une postcondition exprime une disjonction entre les résultats d'un traitement.

Nous sommes cependant obligé dans ce cas de changer la sémantique du OU par rapport aux préconditions. En effet, nous avons déja dans la conclusion du chapitre II, abordé le problème de la traduction de notre langage, les mécanismes de traduction ne nous permettent pas d'utiliser le OU exclusif; nous comprendrons dans le chapitre VI les raisons de cette entorse aux possibilités d'expression du langage.

Si on considère les deux prédicats reliés par cet opérateur, on peut dire que "soit l'un est vrai, soit l'autre, et peut-être les deux".

Si P1, P2 et P3 sont des prédicats simples ou composés, et que l'on retrouve dans la postcondition :

```
OU
      P1
      OU
            P2
            P3
```

Cela signifie que la tâche à laquelle est attachée la postcondition peut avoir les résultats suivants :

- seul P1 est vrai,
- seul P2 est vrai,
- seul P3 est vrai,
- P1 et P2 sont vérifiés,
- P1 et P3 sont vérifiés,
- P2 et P3 sont vérifiés,
- P1, P2 et P3 sont vérifiés.

Après avoir présenté la spécification du comportement proprement dit, nous examinons à présent comment préciser et quantifier les pré et postconditions en vue d'une évaluation par simulation.

## 4.3. Précision et quantification des prédicats

Si l'utilisateur désire évaluer sa modélisation du fonctionnement, il va devoir préciser et quantifier les prédicats.

Nous présentons une série d'informations supplémentaires au moyen desquelles l'utilisateur va devoir compléter ses pré et postconditions. Ces informations sont nécessaires au simulateur que nous allons employer (cfr.chapitre II 2.5.). La forme syntaxique et la sémantique qui y sont associées sont donc très proches de celles du langage du simulateur. Nous verrons par ailleurs dans le chapitre suivant à quoi correspondent exactement ces informations dans ce langage.

Examinons maintenant quelles sont les informations nécessaires pour compléter la précondition et la postcondition d'une tâche.

### 4.3.1 Précondition

Dans une précondition, lorsqu'un PREDICAT SIMPLE est vérifié (il a la valeur "vrai"), il peut contribuer de différentes manières à l'évaluation du prédicat composé qui le contient. Cette modalité de contribution n'est en aucune manière liée à la spécification conceptuelle du fonctionnement mais est introduite dans le but de pouvoir évaluer celui-ci.

Si l'on veut simuler le comportement d'un SI à partir des spécifications conceptuelles, il est indispensable de déterminer les modalités de contribution d'un prédicat à la vérification d'une précondition.

Nous voulons maintenant exprimer dans quelle mesure il intervient par rapport à l'évaluation de la précondition toute entière. C'est-à-dire, sous quelles modalités (quand, combien, durée) il peut influencer le nombre de "réalisations" de cette précondition, et par conséquent également le nombre d'exécutions de la tâche à laquelle il est associé.

On va donc attacher aux prédicats des propriétés (optionnelles et cumulatives) permettant d'exprimer ces modalités.

A : SURVENANCE

La tâche dont on veut exprimer la précondition peut se trouver à la frontière de l'activité que l'on désire simuler, c'est-à-dire que ses conditions de déclenchement portent sur des objets "externes", se trouvant dans un état "initial" pour l'activité considérée.

Ces objets ne sont les résultats d'aucune des tâches de l'activité;
lors de l'évaluation par simulation, il est cependant nécessaire de fournir des renseignements supplémentaires sur ces objets informationnels qui sont destinés à être "injectés" les premiers dans le SIB, puisque situés à la frontière de l'activité.
Jusqu'à présent, on ignore leurs modalités de survenance, leur période de survenance et leur origine, si ce n'est quelle est externe.
Il est donc nécessaire d'associer à chaque objet externe sa provenance, sa loi de distribution et son "calendrier".

La SURVENANCE est soit une valeur numérique,
soit une distribution
Ex :
DISTRIBUTION EXPNEG AVEC PARAMETRE "1H".

La PROVENANCE figure dans le modèle MIB sous la forme d'occurrence de PERSONNE ou ORGANISATION.

La PERIODE DE SURVENANCE est en fait une occurrence du CALENDRIER appartenant au MIB.

Pour exprimer la propriété de survenance relative à un objet, c'est à dire la fréquence à laquelle une condition sur cet objet est vérifiée, on notera qu'il :

SURVIENT x FOIS DE nom_source PENDANT nom_calendrier

où "x" est le nombre de survenances
"nom_source" représente l'origine de l'objet
"nom_calendrier" est un type de période.

Exemple :

( DOCUMENT lettre_contact : reçu,
SURVIENT 1 FOIS DE étudiant PENDANT rentrée )

Le symbole "," est considéré comme un séparateur de propriétés.

B : CONTRIBUTION

Le nombre de contributions indique le nombre d'occurrences de vérification de la précondition auxquelles peut participer la vérification d'une condition; cette propriété indique si un objet dans un certain état peut participer à une ou plusieurs vérifications de la précondition dans laquelle il se trouve.

Il existe deux cas particuliers, la contribution unique ou multiple.

Pour exprimer cela, on retrouvera dans la précondition une propriété attachée à un objet informationnel dans un certain état, indiquant si celui-ci contribue :

- une fois (valeur par défaut si la propriété est absente)
ou - plusieurs fois

à la vérification de la précondition.

Exemple :

Considérons une tâche qui, si le FICHIER contact est créé, et si elle reçoit un DOCUMENT lettre-contact dans l'état reçu, enregistre ce DOCUMENT.

La propriété de contribution va permettre à l'utilisateur d'exprimer le fait que le FICHIER "contact" peut contribuer plusieurs fois à la vérification de la précondition.
En effet, une vérification du prédicat exprimant que le FICHIER "contact" est créé pourra participer à plusieurs réalisation de la précondition.
Par contre, la vérification du prédicat exprimant que le DOCUMENT lettre-contact est dans l'état reçu, ne contribuera qu'à une seule réalisation de la précondition.

La précondition s' exprimera de la manière suivante :

ET
( FICHIER contact : ouvert,
CONTRIBUE PLUSIEURS FOIS )
( DOCUMENT lettre-contact : reçu )

## C : TEMPORISATION

Lors de la simulation, on peut exprimer le fait que la vérification d'un prédicat portant sur un objet informationnel peut avoir un effet sur la vérification de la précondition toute entière uniquement pendant un laps de temps bien déterminé.

En effet, lors de l'exécution de la simulation, la précondition d'une tâche peut être évaluée plusieurs fois. Si l'on a attaché une propriété de temporisation à l'un des prédicats qui la constituent, lorsque ce prédicat est vérifié pour la première fois, il peut encore contribuer aux évaluations ultérieures de la précondition, mais uniquement pendant une période considérée. Lorsque ce laps de temps est écoulé, il n'a plus aucun effet sur la vérification de la précondition.

La temporisation indique si la vérification d'une condition doit être retenue ou pas; c-à-d si le fait qu'un prédicat (portant sur un objet informationnel et son état) soit vérifié peut avoir des effets au delà du moment où il à été vérifié. Si c'est le cas, il faut spécifier le terme de cette temporisation.

Nous utiliserons pour décrire le terme de la mémorisation, la fin d'un délai.

De la même manière, on trouve dans la précondition une propriété indiquant pendant combien de temps un objet dans un certain état est capable de contribuer à la réalisation de la précondition

ex : ( FICHIER contact : créé,
RETENU PENDANT "120d" )

Il peut y contribuer pendant 120 jours

Rem : si aucune temporisation n'est spécifiée, l'objet et son état contribueront indéfiniment à la vérification de la précondition. Un objet pourrait alors avoir des effets alors qu'il est périmé depuis longtemps. Par exemple, un message annonçant la rentrée académique ne peut avoir d'effets que pendant l'année dans laquelle il est survenu.

D : LE LOT
---

Cette propriété spécifie le fait que la tâche ne s'exécute que si une condition sur un objet informationnel et son état est vérifiée un certain nombre de fois.

- Si l'on désire que la précondition ne soit vérifiée qu'une seule fois (une seule exécution de la tâche) pour tout le lot, on notera

PAR LOT DE nombre

- Si par contre, cette tâche doit être exécutée de manière individuelle pour chaque objet du lot, on notera

PAR LOT DE nombre,
TRAITEMENT INDIVIDUEL

Exemple :

(1) si on désire exprimer dans une précondition que la tâche à laquelle elle est attachée, sera exécutée une seule fois alors que 10 documents de type "lettre_contact" sont dans l'état "reçu", on écrira :

```
( DOCUMENT lettre_contact : reçu,
           PAR LOT DE 10 )
```

Exemple :

(2) si on désire spécifier dans une précondition que la tâche à laquelle elle est attachée sera exécutée lorsque on aura 10 documents de type "lettre_contact" dans l'état "reçu", mais qu'en plus, une fois ces dix documents rassemblés elle sera exécutée indviduellement pour chacun d'entre eux, dans ce cas, on écrira :

```
( DOCUMENT lettre_contact : reçu,
           PAR LOT DE 10,
           TRAITEMENT INDIVIDUEL )
```

Dans l'exemple (1), la tâche n'est exécutée qu'une seule fois pour le lot; par contre dans l'exemple (2), la tâche est exécutée pour chaque élément du lot.

## E : LA CONDITION

Dans une précondition, on peut associer une condition aux prédicats qui la composent.

Dans le contexte de la simulation, la vérification d'un prédicat n'aura d'effet que si la condition qui lui est (éventuellement) associée est également vérifiée. L'évaluation de cette condition ne portera jamais sur une comparaison réelle de deux valeurs; la valeur "vrai" de la condition correspondra à un tirage aléatoire.

Cettre propriété n'est pas nécessaire à l'exécution d'une simulation, mais peut s'avérer utile si l'utilisateur désire émettre une restriction sur la vérification d'un prédicat et sa collaboration vis-à-vis de l'évaluation de la précondition toute entière.

La condition est attachée à un prédicat simple et s'exprime sous la forme syntaxique suivante :

"SI nom-condition"
ou bien
"SI PAS nom-condition".

On sépare cette expression de l'objet informationnel, ou de ses autres propriétés, par le caractère ",".

Exemple 1 :

Soit le prédicat simple
( DOCUMENT lettre-contact : envoyé, SI condition )

Ce prédicat prendra la valeur "vrai"
- si un document de type "lettre-contact" est dans l'état "envoyé"
- et si le prédicat "condition" est vérifié.

Exemple 2 :

Si une tâche de "vérification" est exécutée lorsqu'elle dispose d'un formulaire de demande d'inscription dans l'état "rempli", et que l'administration a transmis un message stipulant que la rentrée académique est survenue, cette tâche a pour précondition :

ET

( MESSAGE rentrée-académique : survenu,
SURVIENT 1 FOIS DE administration PENDANT rentrée,
RETENU PENDANT "60đ",
CONTRIBUE PLUSIEURS FOIS )

( FORMULAIRE demande-inscription : rempli ) .

Si l'on désire maintenant exprimer le fait que la tâche ne doit plus vérifier tous les formulaires de demande d'inscription dont elle dispose, mais seulement une partie de ceux-ci; on joint au prédicat portant sur le formulaire une condition exprimant la proportion de formulaires à prendre en compte (ex: 70%).

On définit une condition "cond-inscription" qui doit prendre la valeur "vrai" dans 70% des cas et la valeur "faux" dans 30% des cas.
Si le prédicat portant sur le formulaire est vérifié, l'exécution de la tâche dépendra de la valeur "vrai" ou "faux" de la condition "cond-inscription" à cet instant.
Si la condition est vérifiée, c'est-à-dire dans 70% des cas, la tâche s'exécutera.

La précondition devient :

ET
( MESSAGE rentrée-académique : survenu,
SURVIENT 1 FOIS DE administration PENDANT rentrée,
RETENU PENDANT "60đ",
CONTRIBUE PLUSIEURS FOIS )

( FORMULAIRE demande-inscription : rempli,
SI cond-inscription ) .

Voyons maintenant quels sont les renseignements à fournir au niveau de la postcondition.

## 4.3.2 La postcondition

Pour évaluer la modélisation du fonctionnement, il faut disposer d'informations supplémentaires dans une postcondition.

On distingue deux types d'informations :
- celles qui se rapportent à l'opérateur "OU", ce sont les conditions,
- et celles qui sont relatives à la quantification des résultats d'une tâche, elles sont exprimées par le lot.

### A : LE "OU" DANS UNE POSTCONDITION

Lors de l'exécution de la simulation, les prédicats (portant sur un objet informationnel et son état) contenus dans le texte de la postcondition de la tâche simulée, sont toujours vérifiés lorsqu'ils sont reliés par l'opérateur 'ET'.

S'ils sont séparés par 'OU', cet opérateur binaire exprime le fait que "soit l'un des prédicats est vérifié, soit l'autre, ou bien les deux". Pour exprimer ceci, le programme de simulation a besoin de renseignements supplémentaires afin de savoir quels seront effectivement les résultats de l'exécution d'une tâche.

Ces renseignements apparaissent sous la forme d'une restriction associée à la production de tel ou tel résultat.
Cette restriction prend la forme d'une condition dont le résultat sera "vrai" ou "faux" selon une valeur statistique.

En effet, dans le contexte de la simulation, l'évaluation d'une condition ne portera jamais sur une comparaison réelle de deux valeurs; la valeur "vrai" de la condition correspondra à un tirage aléatoire.

On distingue deux types de conditions :

- celles qui portent uniquement sur un objet informationnel et son état, c'est-à-dire sur un prédicat simple; elles sont appelées "conditions simples";

- celles qui portent sur un groupe d'objets informationnels et leur état, reliés par l'opérateur "ET", c'est-à-dire sur un prédicat composé; elles sont appelées "conditions composées".

A1 : La condition simple

Elle est attachée à un prédicat simple et s'exprime sous la forme syntaxique suivante :

"SI nom-condition"
ou bien
"SI PAS nom-condition".

On sépare cette expression de l'objet informationnel et de son état par le caractère ",".

Exemple 1 :

Soit le prédicat simple
( DOCUMENT lettre-contact : envoyé, SI condition )

Ce prédicat prendra la valeur "vrai"
- si un document de type "lettre-contact" est dans l'état "envoyé"
- et si le prédicat "condition" est vérifié.

Exemple 2 :

Si P1 et P2 sont des prédicats simples portant sur un objet informationnel dans un certain état,
si "condition" est une expression probabilistique,
et que l'on désire exprimer une exclusion mutuelle au niveau de la production des résultats,
alors on écrira :

OU
( P1, SI condition )
( P2, SI PAS condition )

Ceci signifie que l'on aura :
- P1 si "condition" est vérifiée
- P2 sinon,
mais en aucun cas, on ne pourra trouver P1 et P2.
Ceci est une manière d'exprimer un OU exclusif dans une postcondition.

Exemple 3 :

Si P1, P2 et P3 sont des prédicats simples portant sur un objet informationnel dans un certain état,
si "cond1" et "cond2" sont des expressions probabilistiques,
et que l'on désire exprimer une disjonction non exclusive

au niveau de la production des résultats,
alors on peut écrire :

```
ET
   OU
       ( P1, SI cond1 )
       ( P2, SI cond2 )
   ( P3 )
```

Ceci signifie que l'on aura :
- P1, P2 et P3 si "cond1" et "cond2" sont vérifiées,
- P1, P3 si seule la condition "cond1" est vérifiée,
- P2, P3 si seule la condition "cond2" est vérifiée,
- P3 si "cond1" et "cond2" ne sont pas vérifiées.

### A2 : La condition composée

Elle est attachée à un prédicat composé et s'exprime sous la forme syntaxique suivante :

"SI nom-condition,
[ prédicat-composé ]"
ou bien "SI PAS nom-condition,
[ prédicat-composé ]".

Les symboles "[" et "]" montrent quelle est la portée de cette condition.

On sépare l'expression de la condition et le prédicat composé sur lequel elle porte par le caractère ",".

Une condition composée ne peut porter que sur un prédicat composé contenant des prédicats simples reliés par l'opérateur "ET".
On ne trouvera donc jamais d'opérateur "OU" entre les symboles "[" et "]".

Exemple 1 :

Si PC1 et PC2 sont des prédicats composés,
si "condition" est une expression probabilistique,
et que l'on désire exprimer une exclusion mutuelle au niveau de la production des résultats,
alors on écrira :

```
OU
    SI condition, [ PC1 ]
    SI PAS condition, [ PC2 ]
```

Ceci signifie que l'on aura :
- les prédicats simples de PC1 sont vérifiés si "condition" est vérifiée

- les prédicats simples de PC2 sont vérifiés sinon, mais en aucun cas, ceux de PC1 et de PC2 ne seront vérifiés en même temps.

Exemple 2 :

Si P1, P2, P3 et P4 sont des prédicats simples portant sur un objet informationnel dans un certain état,
si "cond1" et "cond2" sont des expressions probabilistiques,
et que l'on désire exprimer une disjonction non exclusive au niveau de la production des résultats,

alors on peut écrire :

OU

SI cond1,
[ ET
( P1 )
ET
( P2 )
( P3 ) ]

SI cond2,
[ ( P4) ]

Ceci signifie que l'on aura pour résultat :

- P1, P2, P3 et P4 si "cond1" et "cond2" sont vérifiées,
- P1, P2 et P3 si seule la condition "cond1" est vérifiée,
- P4 si seule la condition "cond2" est vérifiée,
- aucun résultat si "cond1" et "cond2" ne sont pas vérifiées.

Exemple 3 :

Si l'on veut exprimer qu'une tâche a pour résultats :

- un message de type "telex" dans l'état "envoyé";

- un formulaire de type "inscription" dans l'état "correct" ainsi qu'un document de type "demande-rens" dans l'état "traité", uniquement si la condition "cond-correct" est vérifiée;

- un formulaire de type "inscription" dans l'état "incorrect" ainsi qu'un document de type "demande-rens" dans l'état "en-attente", si la condition "cond-correct" n'est pas vérifiée;

on écrira :

```
ET
   ( MESSAGE telex : envoyé )

   OU
       SI  cond-correct,
       [  ET
            ( FORMULAIRE inscription : correct )
            ( DOCUMENT demande-rens : traité ) ]

       SI  PAS  cond-correct,
       [  ET
            ( FORMULAIRE inscription : incorrect )
            ( DOCUMENT demande-rens : en-attente ) ]
```

La condition "cond-correct" porte à la fois sur le formulaire et le document.

## B : LA QUANTIFICATION DANS UNE POSTCONDITION

Dans une postcondition, il est possible de quantifier les résultats d'un traitement, c'est à dire, de spécifier le nombre d'objets informationnels dans un certain état qui sont les résultats de l'exécution d'une tâche.

Exemple :

( DOCUMENT lettre_contact : traité,
PAR LOT DE nombre )

est la postcondition d'une tâche qui aura pour résultats plusieurs (nombre) documents de type "lettre_contact" dans l'état "traité" alors qu'elle n'a été exécutée qu'une seule fois.

## 4.3.3 Caractéristiques techniques propres à la simulation

### A : LA CONDITION

Nous avons déjà expliqué (cfr. 4.3.2.A) que lorsque l'on retrouve un opérateur "OU" dans une postcondition, on peut attacher aux prédicats qu'il relie, une condition relative à la vérification de ces prédicats.
De même, dans une précondition (cfr. 4.3.1 E), il est possible d'attacher des conditions aux prédicats qui la composent.

Cette condition est une caractéristique technique propre à la simulation qui n'est pas exploitée au niveau du MIB.
On trouvera sa définition en dehors de la base de données du MIB, dans un endroit que nous appelons "annexe technique au MIB" et qui sera uniquement créé dans le but d'évaluer le comportement du SIB.

Exemple : pour exprimer dans la postcondition d'une tâche que celle-ci aura pour résultats :

- soit l'envoi d'un "dossier réponse" et un "document lettre-réponse" si la condition "cond-envoi" est vérifiée,

- soit uniquement l'envoi d'un "document lettre-réponse" si la condition "cond-envoi" n'est pas vérifiée, on écrira :

```
OU
    ( DOSSIER réponse : envoyé, SI cond-envoi )
    ( DOCUMENT lettre-réponse : envoyé )
```

La condition "cond-envoi" sera définie dans une "annexe technique au MIB" à l'aide d'un langage très proche du du langage descriptif MIBL :

```
DEFINIR CONDITION cond-envoi ;
        PROBABILITE VRAI : 0.80 ;
```

Ceci signifie que la condition "cond-envoi" prendra la valeur "vrai" dans 80% des cas et la valeur "faux" dans 20% des cas.
Lors de la simulation, l'exécution de la tâche aura pour résultats : un "dossier réponse : envoyé" ainsi qu'une "lettre-réponse : envoyé" dans 80% des cas, et uniquement un "document lettre-réponse envoyé" dans 20% des cas.

B : LE PARAMETRE

1. La propriété de survenance dans une précondition (cfr. 4.3.1 A) peut être exprimée au moyen d'un mécanisme appelé "paramètre".

Cette notion de paramètre est en fait une caractéristique technique uniquement nécessaire à la simulation et n'est pas exploitée au niveau du MIB car celui-ci est uniquement descriptif.

Pour cette raison, la définition d'un paramètre ne se retrouvera pas dans le texte des pré-postconditions, mais dans une "annexe technique du MIB" qui sera uniquement exploitée par le simulateur.

Les paramètres y sont définis exactement comme le "paramètre-du-système" en MIBL.
On trouvera par exemple :

```
DEFINIR PARAMETRE arrivée-lettres-contact ;
        DISTRIBUTION EXPNEG AVEC PARAMETRE "1H" ;
```

pour exprimer la distribution des arrivées de type "lettre-de-contact" (objet externe).

2. De même, la quantification dans une postcondition (cfr. 4.3.2.B) peut s'exprimer

- directement par un nombre
  ex : ( DOCUMENT lettre-contact : traité,
  PAR LOT DE 10 )

- ou bien à l'aide d'un paramètre
  ex : ( DOCUMENT lettre-contact : traité,
  PAR LOT DE nombre-lettre ).

Ce paramètre est déclaré dans une "annexe technique au MIB" sous la même forme qu'en MIBL :

```
DEFINIR PARAMETRE nombre-lettre,
          DOMAINE 3 JUSQUE 6 AVEC PROBABILITE 0.3;
          DOMAINE 7 JUSQUE 10 AVEC PROBABILITE 0.7;
```

Il exprime le fait qu'à la fin de l'exécution de la tâche spécifiée, celle-ci aura pour résultats :

de 3 à 6 lettres de contact dans l'état traité dans 30% des cas
et de 7 à 10 lettres de contact dans l'état traité dans 70% des cas.

## 4.4. CONCLUSION

Nous n'avons pas donné à notre modèle un langage formel d'expression du fonctionnement uniquement pour permettre à l'utilisateur de spécifier de manière simple et concise le comportement d'un SIB, mais essentiellement pour lui permettre d'évaluer sa spécification au moyen d'un outil de simulation.

Il nous reste donc à implémenter notre modèle de fonctionnement en le dotant d'un tel outil.

Le chapitre suivant va illustrer par un exemple les différentes notions qui viennent d'être étudiées.
Il sera suivi d'un chapitre montrant les choix que nous avons effectués lors de l'implémentation du modèle dans une optique orientée vers la simulation. Ce chapitre nous permettra notamment de mieux comprendre certains choix ou entorses effectués lors de la définition du langage.

# CHAPITRE V

# EXEMPLE DE SPECIFICATION DU COMPORTEMENT D'UN S. I. PAR PRE ET POSTCONDITIONS

## 5. 1. Introduction

Afin de faciliter la compréhension de la description de la spécification du comportement d'un SIB à l'aide de notre langage, nous allons présenter un exemple simplifié du flux administratif lié à l'inscription des étudiants à l'institut d'informatique.

Nous ne considérerons que le cas des étudiants ayant effectué leurs candidatures aux FNDP et ayant pris contact avec le sécrétariat, jusqu'à l'ouverture de leur dossier.

Après une brève présentation de l'exemple de base, nous décrirons les tâches constitutives de cette activité ainsi que les objets informationnels qu'elles manipulent.
Cette description sera suivie d'une étude des enchaînements possibles entre les tâches, tels que nous les avons étudiés au chapitre III(3. 3. Les pré et postconditions : enchaînement).

## 5.2. Exemple de base

### 5.2.1. Présentation

Les étudiants ayant effectué leurs candidatures en sciences économiques et sociales ou en mathématique aux FNDP, passent directement en première licence.
Ils peuvent introduire une demande soit par lettre, soit par téléphone ou encore en se présentant au secrétariat de l'institut d'informatique.

Au fur et à mesure que les lettres des étudiants demandant à s'inscrire arrivent, elles sont rangées dans un classeur "demandes de renseignements".
Ensuite, le secrétariat leur envoie un formulaire de demande d'inscription à remplir en deux exemplaires, ainsi qu'une lettre contenant des renseignements relatifs à l'inscription.

Quand l'étudiant a renvoyé un bulletin d'inscription dûment complété, on sort sa lettre du classeur de demande de renseignement. Il ne restera finalement dans ce classeur que les demandes d'étudiants qui n'ont pas renvoyé leur bulletin d'inscription.

A la rentrée académique, la secrétaire constitue un dossier contenant le courrier déjà échangé avec tous les étudiants admis. Ces dossiers sont classés par ordre alphabétique.
Chaque année, ils seront complétés. Lorsque l'étudiant a terminé ses études, son dossier sera classé dans le fichier des archives.

Pour décrire le SIB sous-jacent, nous procéderons en deux phases :

- la description des objets
- la description des tâches.

## 5.2.2. Description des objets

La définition des objets MIB a été donnée au chapitre III, nous donnons donc une description succinte, avec pour chaque objet : son nom et la liste de ses états possibles ( car seuls ces renseignements nous sont utiles ici ).

### A : Description des documents

- le document lettre contact

Pour s'inscrire, l'étudiant peut prendre contact avec le secrétariat en envoyant une lettre.

NOM : lettre-contact

ETATS POSSIBLES :

- reçu-secr, lorsque la lettre de contact est arrivée au sécrétariat;
- traité, lorsqu'elle a été prise en considération par la sécrétaire;
- classé-fich-contact , lorsqu'elle est classée dans le classeur des demandes de renseignements;
- classé-dossier-etd, lorsqu'elle est classée dans le dossier de l'étudiant.

### B : Description des messages

- le message téléphone contact

L'étudiant peut prendre contact avec le secrétariat par téléphone.

NOM : téléphone-contact

ETATS POSSIBLES :

- reçu-secr, lorsque l'étudiant a téléphoné au sécrétariat;
- traité, lorsque ce coup de téléphone a été pris en compte par la sécrétaire.

- le message entretien-contact

L'étudiant peut se présenter lui-même au secrétariat.

NOM : entretien-contact

ETATS POSSIBLES :

- reçu-secrétariat, lorsque l'étudiant s'est présenté au sécrétariat;
- traité, lorsque cette entrevue a été prise en compte par la sécrétaire.

- le message rentrée académique

Il indique un événement temporel.

NOM : rentrée-académique

ETATS POSSIBLES :

- survenu, lorsque l'administration fait part au sécrétariat que la rentrée académique a débuté.

C : Description des formulaires

- le formulaire lettre d'acceptation

Quand l'étudiant a pris contact avec le secrétariat, la secrétaire lui envoie une lettre décrivant les modalités d'inscription.

NOM : lettre-acceptation

ETATS POSSIBLES :

- envoyé-étudiant, lorsque la sécrétaire l'a rédigée et envoyée à l'étudiant.

- le formulaire demande d'inscription

Pour s'inscrire, l'étudiant doit remplir un formulaire de demande d'inscription en deux exemplaires.

NOM : demande-inscription

ETATS POSSIBLES :

- à-remplir, lorque la sécrétaire a envoyé une demande d'inscription à l'étudiant pour qu'il la remplisse;
- rempli, lorsque l'étudiant l'a renvoyée;
- correct, lorsqu'elle est correcte;
- à-corriger, lorsqu'elle est incorrecte;
- complet, lorsqu'elle est complète;
- à-compléter, lorsqu'elle est incomplète;
- classé-fich-inscr, lorsqu'elle est classée dans le fichier des inscriptions;
- classé-dossier-etd, lorsqu'elle est rangée dans le dossier de l'étudiant.

D : Description des dossiers

- le dossier réponse

Lorsque l'étudiant a pris contact avec la sécrétaire, elle lui envoie une réponse comprenant :
- le formulaire lettre-acceptation dans l'état "envoyé-étudiant" et
- deux formulaires demande-inscription dans l'état "envoyé-étudiant".

NOM : réponse

ETATS POSSIBLES :

- envoyé-étudiant, lorque la sécrétaire a constitué une réponse et l'a envoyée à l'étudiant.

- le dossier étudiant

A la rentrée académique, pour chaque étudiant inscrit, la secrétaire ouvre un dossier où elle range une demande d'inscription remplie et, éventuellement, la lettre de prise de contact envoyée par l'étudiant.

NOM : étudiant

ETATS POSSIBLES :

- ouvert, lorsqu'il est créé;
- archivé, lorsque l'étudiant a terminé ses études, son dossier est classé dans les archives.

E: Description des fichiers

Pour chaque fichier, on donnera son nom, la liste de ses états possibles et le nom de l'objet constituant.

- le fichier contact

On y range les lettres de prise de contact envoyées par les étudiants. Elles sont retirées quand l'étudiant est effectivement inscrit aux facultés.

NOM : fichier-contact

ETATS POSSIBLES :

- créé, lorsque le fichier est "ouvert" et que l'on peut y classer des éléments;
- clôturé, ce fichier est "fermé" et on ne peut plus le modifier.

CONSTITUANT : document lettre-contact.

- le fichier inscription

On y range les demandes d'inscription remplies correctement et complètement par l'étudiant.

NOM : inscription

ETATS POSSIBLES :

- créé, lorsque le fichier est "ouvert" et que l'on peut y classer des éléments;
- clôturé, ce fichier est "fermé" et on ne peut plus le modifier.

CONSTITUANT : formulaire demande-inscription.

## 5.2.3. Description des tâches

Décrivons maintenant les tâches constitutives de l'activité, ainsi que le contenu de leurs pré et postconditions.

TACHE 1: traitement des contacts avec l'étudiant

La secrétaire reçoit un contact de l'étudiant (écrit ou oral). Elle prépare une réponse, consistant en une lettre d'acceptation et une demande d'inscription à remplir en double exemplaire.
Elle lui envoie cette réponse.

- NOM : trt_cont_etd

- PRECONDITION :

OU

( MESSAGE entretien-contact : reçu-secr,
SURVIENT 5 FOIS DE étudiant PENDANT journée )

OU

( MESSAGE téléphone-contact : reçu-secr,
SURVIENT 5 FOIS DE étudiant PENDANT journée )

( DOCUMENT lettre-contact : reçu-secr,
SURVIENT CHAQUE distrib-let-cont DE étudiant
PENDANT journée )

La propriété de survenance est exprimée dans la précondition et est attachée à l'objet informationnel sur lequel elle porte :
"journée" est le nom d'un calendrier défini dans le MIB;
"étudiant" indique la provenance de cet objet et peut ne pas être décrit explicitement dans le MIB;
"distrib-let-cont" est un paramètre exprimant la distribution statistique des arrivées de lettres de contact, cette caractéristique technique (propre à la simulation) sera définie dans une "annexe technique au MIB" de la façon suivante :
DEFINIR PARAMETRE distrib-let-cont;
DISTRIBUTION EXPNEG AVEC PARAMETRE "1H";

- POSTCONDITION :

ET

( DOSSIER réponse : envoyé-étudiant )

( DOCUMENT lettre-contact : traité )

TACHE 2: classement des lettres de contact

Si le contact de l'étudiant était un contact écrit, la secrétaire le classe dans le fichier des contacts si celui-ci a été préalablement créé.

- NOM : classement_lettre_contact

- PRECONDITION :

ET

( DOCUMENT lettre-contact : traité )

( FICHIER fichier-contact : créé,
SURVIENT 1 FOIS DE administration PENDANT rentrée,
CONTRIBUE PLUSIEURS FOIS,
RETENU PENDANT "120d" )

La propriété de contribution attachée au "fichier-contact" stipule qu'une vérification du prédicat "un fichier-contact dans l'état créé" peut contribuer plusieurs fois à la vérification de la précondition toute entière.

De même, la propriété de temporisation exprime que la vérification du prédicat portant sur le "fichier-contact" peut encore avoir des effets sur la vérification de la précondition, et celà uniquement pendant 120 jours.

- POSTCONDITION :

( FICHIER contacts MAJ+ PAR
DOCUMENT lettre-contact : classé-fich-contact )

TACHE 3: réception de la demande d'inscription

Cette tâche ne fait pas partie intégrante de l'activité décrite, puisqu'elle n'est pas exécutée au sécrétariat des facultés. Elle sert à simuler l'activité de l'étudiant qui remplit sa demande d'inscription.
Comme toute autre tâche, elle est décrite dans le MIB et sa durée d'exécution est très utile lorsque l'on veut évaluer le comportement du système décrit.

Elle simule le fait que l'étudiant remplit sa demande d'inscription lorsqu'on lui a envoyé la lettre de contact et le formulaire à remplir, c'est à dire le dossier réponse.
Elle permet également d'exprimer que le nombre de formulaires remplis renvoyés par les étudiants peut être inférieur au nombre de formulaires à remplir envoyés par le serétariat.
En effet, certains étudiants (ex : 10 %) peuvent changer d'avis lors de la réception du dossier réponse et ne jamais renvoyer le formulaire de demande d'inscription rempli.

- NOM : reception_demande_inscription

- PRECONDITION :

( DOSSIER réponse : envoyé-étudiant )

- POSTCONDITION :

( FORMULAIRE demande-inscription : rempli,
SI cond-réception )

La condition "cond-réception" permet d'émettre une restriction quant à la vérification du prédicat portant sur le formulaire rempli. Elle exprime le taux de formulaires effectivement renvoyés par les étudiants par rapport au nombre de ceux qu'on leur a envoyés à remplir.
Si "cond-réception" est définie de la façon suivante dans le "annexe technique au MIB" :

```
DEFINIR  CONDITION cond-réception;
         PROBABILITE VRAIE : 0.90;
```

elle exprime que l'étudiant renverra une demande d'inscription remplie dans 90% des cas.
En effet, pour chaque dossier réponse envoyé, il y a une exécution de la tâche "réception_demande_inscription", et celle-ci aura pour résultat :

- une demande d'inscription remplie (avec une probabilité de 0.90)

ou

- aucun résultat (avec une probabilité 0.10).

## TACHE 4: vérification demande inscription correcte

La secrétaire vérifie si le formulaire de demande d'inscription a été correctement rempli. Si non, elle le renvoie au secrétariat central pour qu'il le corrige.

- NOM : vérification_correct

- PRECONDITION :

( FORMULAIRE demande-inscription : rempli )

- POSTCONDITION :

OU

( FORMULAIRE demande-inscription : correct,
SI cond-correct )

( FORMULAIRE demande-inscription : à-corriger,
SI PAS cond-correct )

TACHE 5: vérification demande inscription complète

La secrétaire vérifie si le formulaire de demande d'inscription est complet. Dans le cas contraire, elle le renvoie à l'étudiant pour qu'il le complète.

- NOM : vérification-complet

- PRECONDITION :

( FORMULAIRE demande-inscription : rempli )

- POSTCONDITION :

OU

( FORMULAIRE demande-inscription : complet,
SI cond-complet )

( FORMULAIRE demande-inscription : à-compléter,
SI PAS cond-complet )

Remarquons que l'emploi du OU avec les formes syntaxiques SI et SI PAS, permet d'exprimer un OU exclusif.

TACHE 6: classement de demande d'inscription

Quand la demande d'inscription est remplie correctement et complètement, la secrétaire la classe dans le fichier des inscriptions. Elle effectue uniquement cette opération lorsqu'elle a reçu 5 réponses correctes.

Si l'administration à l'occasion de la rentrée ouvre le FICHIER inscription et qu'un lot de 5 FORMULAIREs demande-inscription dans l'état complet-et-correct est disponible, cette tâche procédera au classement dans le FICHIER inscription des 5 FORMULAIREs en une seule fois.

- NOM : classer_demande_inscription

- PRECONDITION :

ET

( FICHIER inscription : créé,
SURVIENT 1 FOIS DE administration PENDANT rentrée,
CONTRIBUE PLUSIEURS FOIS,
RETENU PENDANT "60d" )

( FORMULAIRE demande-inscription : complet ET correct,
PAR LOT DE 5 )

La précondition spécifie le fait que la tache s'exécute pour un lot de 5 formulaires. La clause "PAR LOT DE 5" permet d'exprimer ceci et signifie qu 'on attend 5 formulaires avant toute exécution de la tâche.

On exprime également le fait que la tache ne s'exécute qu'une seule fois pour ce lot au moyen de la clause "TRAITEMENT PAR LOT".
Par contre, l'utilisateur aurait pu vouloir spécifier que la tache ne s'exécutait pas une fois pour tout un lot, mais une fois par élément constitutif du lot; il aurait alors ajouté la clause " TRAITEMENT INDIVIDUEL ".

- POSTCONDITION :

( FICHIER inscription MAJ+ PAR
FORMULAIRE demande-inscription : classé-fich-inscr,
PAR LOT DE 5 )

La postcondition spécifie simplement le fait que comme on effectuait un traitement par lot, le fichier inscription a été mis à jour par 5 formulaires demande-inscription et que l'état de ces formulaires est devenu : classé-fich-inscr.
Au lieu du nombre "5", on aurait pu employer un paramètre défini dans "l'annexe technique au MIB".

TACHE 7: ouverture du dossier

A la rentrée académique, la secrétaire prend tous les formulaires de demande d'inscription ainsi que les lettres de contact (rangées dans le fichier "contact ") et ouvre un dossier pour l'étudiant.

Sur base de tous les FORMULAIREs demande-insciption complets et corrects reçus jusqu'à l'arrivée du MESSAGE rentrée-académique et pendant les deux mois qui suivent la rentrée, cette tâche ouvre un DOSSIER étudiant et y classera pour chaque étudiant le FORMULAIRE demande-inscription. Elle s'exécute une fois par formulaire reçu.

- NOM : ouverture_dossier

- PRECONDITION :

ET

( FORMULAIRE demande-inscription : complet ET correct )

( MESSAGE rentrée-académique : survenu,
SURVIENT UNE FOIS DE administration PENDANT rentrée,
RETENU PENDANT "60d",
CONTRIBUE PLUSIEURS FOIS )

Les formulaires arrivant après le message calendrier mais durant la période de temporisation de celui-ci occasionneront chacun une exécution de la tâche; après le terme de la temporisation, il n'y aura plus d'exécution.

Si aucune durée de temporisation n'était spécifiée, il y aurait toujours exécution de la tâche après la survenance du message calendrier (pour chaque arrivée de formulaire).

- POSTCONDITION :

ET

( DOSSIER étudiant : ouvert )

ET

( FORMULAIRE demande-inscription : classé-dossier-etd )

( DOCUMENT lettre-contact : classé-dossier-etd )

## 5.2.4. Enchaînements possibles

Nous avons expliqué au chapitre III (3.3.) comment se réalisait l'enchaînement entre les tâches composant l'activité. Voyons maintenant comment pourrait se dérouler l'activité décrite dans l'exemple de base.

- Si la tâche1 de traitement des contacts avec les étudiants s'est correctement exécutée, elle a pour résultats :
un document lettre-contact dans l'état traité
et un dossier réponse dans l'état envoyé-étudiant.

On remarque que la tâche2 de classement des lettres de contact pourrait éventuellement s'exécuter, pour autant que l'administration des facultés ait transmis un fichier contact dans l'état créé.

La tâche3 réception de la demande d'inscription peut s'exécuter également puisque sa précondition porte sur un dossier réponse dans l'état envoyé-étudiant et est contenue dans la postcondition de la tâche1.

- Si la tâche3 de réception de demande d'inscription s'est correctement exécutée, elle a pour résultat que le formulaire demande d'inscription est dans l'état rempli (sous la condition cond-réception) ou aucun résultat(si cond-réception n'est pas vérifiée).

Dans le cas où cette condition est vérifiée, on remarque que la tâche4 de vérification si demande inscription correcte peut s'exécuter. En effet, sa précondition correspond à la postcondition de la tâche3.

De la même manière, la tâche5 de vérification si demande inscription complète ayant pour précondition que le formulaire demande d'inscription doit se trouver dans l'état rempli, pourra s'exécuter.

On peut constater que ces deux tâches peuvent s'effectuer en parallèle.

- La tâche6 de classement des demandes d'inscription a pour précondition que :
le fichier des inscriptions doit se trouver dans l'état créé et que 5 formulaires de demande d'inscription sont dans l'état correct et complet.

On remarque qu'elle ne pourra s'exécuter que si les tâches de vérification si demande inscription complète et de vérification si demande d'inscription correcte se sont effectuées correctement.

En effet, la première peut avoir pour résultat que le formulaire de demande d'inscription est dans l'état complet (sous la condition cond-complet), et la seconde que ce même formulaire est dans l'état correct (sous la condition cond-correct).

Rappelons que les différents scénarios que nous venons de passer en revue sont uniquement des enchaînements possibles et que le déroulement de l'activité d'inscription des étudiants peut différer selon les valeurs de certains paramètres :

- la survenance des objets externes
- la vérification des conditions attachées aux pré et postconditions
- la disponibilité des ressources
- la possibilité d'effectuer certains traitements en parallèle ou non.

Cet exemple montre bien que la simulation peut s'avérer très utile dans l'étude des différents scénarios d'exécution, surtout avec le type de spécification du comportement que nous avons choisi.

En effet, une fois le modèle des données spécifié, et à partir de la description par pré et postconditions, la simulation permettra peut-être de découvrir qu'il existe d'autres manières d'organiser ou de réorganiser le travail, ou que la performance du SIB ne répond pas aux attentes du concepteur.

## 5.3. Conclusion

Nous avons montré sur un exemple de base comment on peut décrire le comportement d'un SIB à l'aide de pré et postconditions, et surtout exprimer ces conditions dans un langage formel ( le MIBL ) afin de permettre l'évaluation de ces spécifications au moyen d'un outil de simulation.

Nous devons à présent implémenter notre modèle en le dotant d'un tel outil. Le chapitre suivant explique les choix effectués dans cette optique, ainsi que les mécanismes de transformation qui en découlent.

# CHAPITRE VI

# LE MODELE D'IMPLEMENTATION

## 6.1. Introduction

Le but de ce mémoire n'est pas uniquement de doter le MIB d'un langage d'expression du comportement sous forme de pré et postconditions, mais aussi de permettre l'évaluation par simulation d'une spécification de fonctionnement d'un SIB exprimée au moyen de ce langage.
Evaluer nécessite un outil qui le permette. Cet outil est appelé un simulateur.

Nous avons décidé d'utiliser le simulateur DSL_SIM existant dans l'environnement logiciel IDA.

IDA ( Interactive Design Approach ) est un atelier logiciel d'aide à l'analyse fonctionnelle dont DSL ( dynamic specification language ) est le langage de spécification. Ce langage permet entre autres choses de décrire le fonctionnement d'un SI en se basant sur les concepts d'événement et de processus. Une description complète des concepts et du langage se trouve dans [BOD_PIG, 83] et [DSL_SIM, 83].

Le simulateur que nous allons employer est un outil spécifique à DSL ou du moins au sous-ensemble de DSL relatif à l'expression de la dynamique. Les concepts de base de ce langage (orienté traitement) sont différents de ceux véhiculés par MIBL (orienté objets).

Pour montrer que l'on peut retrouver les mêmes structures d'enchaînement élémentaires entre traitements que celles utilisées en DSL, nous consacrerons la première partie de chapitre à l'étude des enchaînements réalisables, bien que l'on ne spécifie pas explicitement le concept d'enchaînement en MIBL. Ceci nous permettra de montrer que nous pouvons exprimer au moins les mêmes enchaînement qu'en DSL et donc que nous pouvons traduire une spécification MIBL en DSL afin qu'elle puisse être simulée.

Dans une deuxième partie, nous présenterons les principes et le processus de traduction.

Ce chapitre se terminera par une application du principe de traduction sur l'exemple de base décrit au chapitre 5.

## 6.2. Comparaison des structures d'enchaînement de MIBL et de DSL

Rappelons d'abord les principaux concepts de notre approche et de celle adoptée par Bodart et Pigneur dans leur conception de la simulation.

### 6.2.1. Concepts fondamentaux de MIBL

Le langage que nous avons défini (cfr. chapitre IV) est orienté objets et utilise des pré et postconditions.

La PRECONDITION d'une tâche est une condition explicitant les propriétés qui doivent être vérifiées avant toute exécution de la tâche pour que celle-ci s'exécute correctement.

Ces propriétés porteront sur les objets informationnels du MIB et l'état dans lequel ceux-ci se trouvent.

La POSTCONDITION est une condition explicitant les propriétés des résultats de la tâche, qui doivent être satisfaites à la fin de toute exécution de la tâche.

Ces propriétes ( tout comme dans la précondition ) portent sur

les objets informationnels ainsi que l'état dans lequel ceux-ci se trouvent.

Nous avons déja examiné comment il était possible de spécifier le fonctionnement d'un SI au moyen de pré et postconditions ( cfr. chapitre III), resituons à présent les concepts sur lesquels repose le langage DSL.

### 6.2.2. Concepts DSL

Une autre manière de décrire l'enchaînement des tâches, est basée sur les notions de processus et d'événement, utilisées dans DSL [BOD_PIG, 83]. Une description détaillée de cette approche a été faite au chapitre II, nous n'en reprenons ici que les éléments les plus importants.

Cette approche repose sur deux concepts :

1. Un PROCESSUS est l'exécution d'une procédure de traitement de l'information.

2. Un EVENEMENT correspond à un changement d'état du système caractérisé dans le temps et dans l'espace. On distingue des événements INTERNES et EXTERNES.

Un événement est dit externe s'il correspond au franchissement de la frontière du SI par un message en provenance de son environnement et qui déclenche en réaction un processus du SI.

Un événement est dit interne s'il correspond à un changement d'état interne au SI. Ces événements sont :

-un changement d'état d'un processus
-la réalisation d'un point de synchronisation
-la génération d'un message par un processus.

## 6.2.3. Comparaison

Nous allons montrer que l'on peut au moins retrouver dans une spécification effectuée au moyen de MIBL, les mêmes structures d'enchaînement élémentaires entre traitements que celles que l'on retrouve dans une spécification dynamique en DSL. Le fonctionnement d'un SI se retrouve en DSL, en combinant certaines structures élémentaires (cfr. chapitre II 3.1.1.). Il s'agit des structures :

- d'enchaînement séquentiel
- d'enchaînement éclaté
- d'enchaînement multiple
- d'enchaînement conditionnel
- d'enchaînement convergent
- d'enchaînement synchronisé.

Avant d'aborder ces différents enchaînements, remarquons encore qu'en DSL ce sont les événements qui déclenchent les processus, par contre en MIBL ce sont les tâches qui vérifient si les conditions nécessaires à leur déclenchement sont remplies. Ces conditions portent sur l'état d'objets informationnels, et non sur l'état dans lequel pourraient se trouver d'autres tâches (déclenchées, terminées ...).

Etant donné les propriétés de notre approche, le flux de contrôle et donc les structures d'enchaînement entre les tâches sont invisibles à priori, c-à-d, au moment de la spécification du fonctionnement. Le concept d'enchaînement ou d'agencement de tâches a donc uniquement un sens pour nous soit lors de l'exécution du système réel, soit lors de la simulation de sa description.

### -1- ENCHAINEMENT SEQUENTIEL

Un enchaînement est dit séquentiel lorsque la terminaison d'un processus P1 provoque le déclenchement d'un processus P2.

Exemple : la terminaison d'un processus "d'Enregistrement d'une commande" déclenche un processus "Mise à jour du stock".

Cette structure d'enchaînement se retrouve dans notre approche lorsque la précondition de la tâche T2 est contenue dans la postcondition de la tâche T1.
La tâche T2 pourra s'exécuter après la tache T1.

Exemple : si la postcondition d'une tâche "Enregistrement des contacts" est :
ET
( MESSAGE téléphone-contact : traité )
( FORMULAIRE demande-inscription : rempli )

et que la précondition de la tâche "traitement des contacts" est :
( FORMULAIRE demande-incription : rempli ),

le traitement des contacts pourra s'effectuer après leur enregistrement.

-2- ENCHAINEMENT ECLATE

Un enchaînement est dit éclaté lorsque la terminaison d'un processus P1 provoque le déclenchement simultané de P2, P3, .... Pn.

Exemple : la terminaison d'un processus de "Réception des documents d'une livraison" déclenche un processus de "Contrôle de qualité" et de "Contrôle de quantité".

Dans notre approche, cette structure est réalisée quand les preconditions des tâches T2, T3 .... et de Tn (prises individuellement) sont contenues (cfr. chapitre III 2.3.3) dans la postcondition de T1.

Exemple : si la postcondition d'une tâche "d'Enregistrement des contacts" est :
ET
( MESSAGE téléphone-contact : traité )
( FORMULAIRE demande-inscription : rempli ) ,

que la précondition d'une tâche de "Vérification-correct" est :
( FORMULAIRE demande-inscription : rempli )

et que la précondition d'une tâche de "Vérification-complet" est également :
( FORMULAIRE demande-inscription : rempli ),

ces deux tâches de contrôle pourront s'effectuer après l'enregistrement des contacts des étudiants.

## -3- ENCHAINEMENT MULTIPLE

Un enchaînement est dit multiple lorsque la terminaison d'un processus P1 provoque le déclenchement simultané de n (n>1) processus de type P2, c'est-à-dire n exécutions du traitement P2.

Exemple : la terminaison d'un processus "d'Enregistrement d'une livraison" déclenche autant de "Mise à jour du stock" qu'il y a de produits réapprovisionnés.

Dans notre approche, cet enchaînement est réalisé quand la précondition de la tâche T2 est contenue dans la postcondition de la tâche T1. C'est dans la postcondition de T1 que l'on retrouve le fait que cette tâche à "n" résultats.
Il en résultera donc "n" exécutions de T2 qui retrouve sa précondition dans chacun des "n" résultats pris individuellement.

Exemple : considérons une tâche de "Sélection des demandes d'inscription" qui, à partir du fichier contenant des demandes d'inscription, sélectionne celles qui sont relatives aux étudiants effectivement admis.

Cette tâche a pour résultats (postcondition) :
( nombre-etd-admis FORMULAIRE demande-inscription : correct ).

Une autre tâche de "Création de dossier", crée un dossier sur base du formulaire de demande d'inscription correct, et ceci pour chaque étudiant admis. Elle a pour précondition :

( FORMULAIRE demande-inscription : correct )

et pourra être exécutée après le tri autant de fois qu'il y a d'étudiants admis.

## -4- ENCHAINEMENT CONDITIONNEL

Un enchaînement est dit conditionnel lorsque la terminaison d'un processus P1 déclenche le processus P2 si la condition C1 est vraie et déclenche P3 si la condition C1 est fausse.

Exemple : la terminaison d'un processus de "vérification de signature d'une commande "déclenche un processus de "vérification de l'identité du client" si la commande est signée, sinon, elle déclenche un processus de "refus de la commande".

Dans notre approche, l'enchaînement conditionnel est réalisé dans la postcondition de la tâche T1 où on retrouve deux propriétés des résultats différentes et à chacune est attachée une condition qui définira laquelle des deux propriétés sera vérifiée après l'exécution de la tâche T1.

La précondition de la tâche T2 se retrouve dans l'une de ces propriétés et la précondition de la tâche T3 est contenue dans l'autre.

Exemple : si la postcondition d'une tâche de "Vérification-correct" est la suivante :

OU
( FORMULAIRE demande-inscription : correct, SI cond )
( FORMULAIRE demande-inscription : à-corriger )

et la précondition d'une tâche de "Classement d'une demande d'inscription" est :
( FORMULAIRE demande-inscription : correct )

et enfin, la précondition d'une tâche de "renvoi de formulaire à l'étudiant" est :
( FORMULAIRE demande-inscription : à-corriger ),

après la vérification du formulaire, on pourra exécuter soit un classement, soit un renvoi à l'étudiant selon la valeur de "cond".

## -5- ENCHAINEMENT CONVERGENT

Un enchaînement est dit convergent si le déclenchement des processus Pi est provoqué par la terminaison d' un des processus P1,...Pn.

Exemple : L'"Impression d'un extrait de compte" peut être déclenchée par la terminaison de l'"Enregistrement d'un payement" ou par la terminaison de l'"Enregistrement d'un versement".

Pour l'approche pré-post comme pour l'approche DSL, cette structure ne se différencie pas de l'approche séquentielle.
En effet, cet enchaînement sera réalisé lorsque la precondition de la tâche Ti sera contenue dans l'une des postconditions des tâches T1, T2,... Tn ou bien tout simplement lorsque l'on retrouve un opérateur "OU" dans la précondition de Ti.

Exemple : Une tâche de "Traitement des contacts étudiants" peut être effectuée sur base d'un message d'entretien reçu à la réception par la tâche "Réception" ou bien à partir d'une lettre de contact réceptionnée au sécrétariat central par la tâche "Réception sécrétariat".

La tâche "Réception" a pour postcondition :
( MESSAGE entretien-de-contact : reçu-secr )

et la tâche "Réception sécrétariat" :
( DOCUMENT lettre-contact : reçu-secr ).

La précondition de la tâche de "Traitement des contacts" sera :
OU
( MESSAGE entretien-de-contact : reçu-secr )
( DOCUMENT lettre-contact : reçu-secr ) .

## -6- ENCHAINEMENT SYNCHRONISE

P1 et P2 sont des processus asynchrones travaillant à des rythmes différents. Le déclenchement de P3 va dépendre de la terminaison de P1 et P2.

La coordination de P1 et P2 est réglée par le mécanisme DSL du point de synchronisation. Celui-ci définit une condition de synchronisation permettant d'exprimer que : le processus P3 ne sera déclenché qu'à la terminaison de P1 et P2.
La réalisation de la condition de synchronisation est l'événement qui va déclencher P3.

Exemple : les processus "Contrôle de qualité" et "Contrôle de quantité" travaillent à des rythmes différents. Le déclenchement d'un processus "d'Enregistrement de la livraison" dépend de la terminaison de ces deux processus. Pour régler la coordination, on utilise le mécanisme du point de synchronisation auquel est attaché une condition.
C'est la réalisation de la condition d'attente (les deux processus sont teminés) qui va provoquer le

déclenchement d'un enregistrement de livraison.

Dans notre approche, l'évaluation de la précondition de la tâche T3 correspond à la vérification des predicats qu'elle contient. Si chacun de ces prédicats se retrouve dans les postconditions de l'une des tâches T1 ou T2, la vérification de la précondition de T3 représente une synchronisation des tâches T1 et T2.

Exemple : une tâche de "Vérification-correct" vérifie si un formulaire de demande d'inscription est correct ou non, sa postcondition est :

OU
( FORMULAIRE demande-inscription : correct )
( FORMULAIRE demande-inscription : à-corriger ).

De même, une tâche de "Vérification-complet" a pour postcondition :

OU
( FORMULAIRE demande-inscription : complet )
( FORMULAIRE demande-inscription : à-compléter ).

Une tâche de "Classement des demandes d'inscription" ayant pour précondition :
( FORMULAIRE demande-inscription : complet et correct )
ne pourra s'exécuter que lorsque l'on aura vérifié que le formulaire est non seulement complet, mais aussi correct.

### 6.2.4. Evaluation

Nous pouvons remarquer qu'en DSL le processus est "aveugle", c'est à dire qu'il ignore les conditions dans lesquelles il est déclenché et qu'il n'a aucun moyen de contrôle sur les noeuds de décisions qui le précèdent dans les différents enchaînements constituant le scénario du SI décrit.

Par contre la tâche spécifiée au moyen de pré et postconditions "sait" quand elle peut s'exécuter et les noeuds de décision (flux de contrôle) du scénario de l'activité sont "cachés" dans les pré et postconditions des tâches qui constituent cette activité.

En tout état de cause, les six enchaînements de l'approche événement-processus peuvent se retrouver si on considère l'activité globale de plusieurs tâches spécifiées au moyen de pré

et postconditions, ce qui est nécessaire pour utiliser l'outil DSL-SIM existant.

Maintenant que nous savons que nous pouvons reproduire les mêmes enchaînements que DSL, examinons comment traduire notre langage en DSL.

## 6.3. Principes de traduction de MIBL en DSL

Examinons à présent comment on peut traduire notre langage en DSL afin d'utiliser le simulateur DSL-SIM.

Trois aspects de la modélisation d'un SI sont nécessaires pour utiliser DSL-SIM.

- le comportement fonctionnel
- les ressources
- les quantifications.

Nous allons étudier ces trois aspects dans notre langage et en DSL, en citant pour chacun les principes de traduction.

### 6.3.1. Le comportement fonctionnel

Notre langage permet d'exprimer le fonctionnement d'un SIB au moyen de trois concepts : la tâche, la précondition et la postcondition.

A : La tâche

Dans le MIB, l'entité TACHE a pour attributs :

- PRECONDITION et POSTCONDITION. Le format de ces conditions est un texte libre tant que l'utilisateur reste à un niveau descriptif mais elles doivent être exprimées dans un langage formel répondant à la syntaxe et sémantique présentées dans le chapitre IV, s'il désire évaluer le comportement de son S.I.B.

- DESCRIPTION, dont le format est également un texte libre qui exprime la dynamique interne de la tâche, c'est-à-dire la procédure de traitement de l'information effectuée par des opérations primitives; ceci est l'objet d'une autre étude [VANPEVE, 87].

On pourrait penser à priori traduire la TACHE MIB en un PROCESSUS DSL. Ce n'est cependant pas tout à fait vrai.
En effet, le PROCESSUS est "l'exécution" d'un traitement alors que la tâche est le traitement en lui-même, avec ses conditions de déclenchement et les propriétés de ses résultats.
Ce qui correspond en fait au PROCESSUS , c'est la DESCRIPTION de la TACHE, car cette DESCRIPTION indique le procédé de traitement de l'information au niveau d'une TACHE.

La traduction à exécuter sera donc :

| MIB | | DSL |
|---|---|---|
| DESCRIPTION (de la TACHE) | <-----> | PROCESSUS |

Comme le processus correspond à l'exécution d'une TACHE dans le MIB, nous donnerons au processus le nom de la tâche dont il représente l'exécution.

La "phrase" DSL utilisée sera :

DEF PROCESSUS nom-processus;

B : La précondition

Une précondition est constituée d'un ou plusieurs prédicats interreliés par des opérateurs ( et, ou ). Chacun de ces prédicats porte sur un objet informationnel ainsi que sur l'état de cet objet.

Exemple :

- le document de type " lettre_de_contact" dans l'état " reçu_sécrétariat "
- le fichier de type "contacts" dans l'état "ouvert" est mis à jour par un document de type "lettre_de_contact" dans l'état "classé_fichier".

L'objet DSL MESSAGE permet de représenter les informations échangées et sa génération constitue un changement d'état du système, donc un événement.

Un PREDICAT portant sur un objet et son état, sera traduit en un MESSAGE DSL. Ce message signalera qu' un objet du MIB se trouve dans un état particulier.

La génération de différents messages pour un même objet, illustrera la notion de cycle de vie des objets informationnels à travers l'activité décrite.

Exemple :

| MIB | <----------> | DSL |
|---|---|---|
| DOCUMENT : lettre-de-contact; ETAT : reçu | | MESSAGE : m1 |
| DOCUMENT : lettre-de-contact; ETAT : traité | | MESSAGE : m2 |
| DOCUMENT : lettre-de-contact; ETAT : classé-dossier | | MESSAGE : m3 |

Comme un objet dans un certain état est traduit en message, et que la génération d'un message est un événement, le changement d'état d'un objet correspond donc également à un événement.

Il en découle que la précondition, qui est une interrelation de prédicats portant sur des objets dans un certain état, équivaut à une coordination d'événements.

Le mécanisme DSL permettant d'exprimer une coordination d'événements est le POINT-DE-SYNCHRONISATION et sa réalisation correspond à la vérification de la précondition.

La forme syntaxique utilisée pour le définir est :

DEF POINT-DE-SYNCHRONISATION nom-point-de-synchronisation;

La spécification d'un point de synchronisation requiert les les quatre informations suivantes :

la condition de synchronisation,
l'événement causé par la réalisation de la condition,
les événements coordonnés par la condition,
les modalités de contribution d'un événement.

(1) - la condition de synchronisation;
La condition de synchronisation définit la coordination entre les événements qui contribuent au point-de-synchronisation. Cette condition va nous permettre de traduire la coordination de prédicats exprimée dans une précondition. Comme les

prédicats portant sur des objets informationnels sont traduits en MESSAGEs DSL, les événements coordonnés par le point de synchronisation seront des GENERATIONs de MESSAGEs.

La condition de synchronisation s'exprime au moyen d'un langage formel manipulant des opérandes et des opérateurs.

(a) Les opérandes prises en considération, sont les générations de messages. La forme syntaxique utilisée pour exprimer de telles opérandes est :

GEN DE nom-message [ SI [ PAS ] nom-condition ]

Ceci permet d'exprimer le fait qu'un objet MIB traduit en message DSL doit être coordonné dans la précondition. L'emploi de la clause conditionnelle pour ce message permettra de traduire une restriction sur les objets participants à la précondition.

(b) les opérateurs utilisés sont : ET, OU, COMPTER

ET est un opérateur binaire employé en notation préfixée pour exprimer une condition "et". Son format est :

ET
opérande-1
opérande-2

OU est un opérateur binaire employé en notation préfixée pour exprimer une condition "ou". Son format est :

OU
opérande-1
opérande-2

COMPTER est un opérateur unaire utilisé pour compter un certain nombre d'événements.

Nous utilisons cet opérateur pour traduire la propriété de "lot" que l'on peut retrouver dans une précondition. On peut en effet exprimer en MIBL qu'une précondition n'est vérifiée que si un lot d'un certain nombre d'objets informationnels est constitué; ceci sera traduit par l'opérateur COMPTER en DSL.

La forme syntaxique employée est :

COMPTER nom-attribut
GEN nom-message

où nom-attribut est le nom d'un attribut DSL auquel est affecté une valeur lors de la simulation; cette valeur représente la traduction de la taille du "lot" spécifié en MIBL.

(2) - l'événement causé par la réalisation de la condition;
Tout comme la vérification de sa précondition entraine le déclenchement de la tâche, la réalisation du point-de-synchronisation (traduisant la précondition ) entrainera le déclenchement d'un processus qui a pour nom celui de la tâche dont il représente l'exécution.
L'événement résultant de sa réalisation sera donc le déclenchement d'un processus. La phrase DSL permettant d'exprimer cela est :

EN REALISATION DECLENCHE nom-processus [POUR CHAQUE nom-attribut]

Si l'utilisateur a spécifié un lot dans sa précondition, il a également mentionné si (une fois ce lot constitué), la tâche s'exécutait pour chaque élément à l'intérieur de ce lot ou globalement pour tout le lot (TRAITEMENT INDIVIDUEL ou PAR LOT ). La clause DSL "POUR CHAQUE nom-attribut" nous permet d'exprimer cela.
En effet, si l'utilisateur désire une exécution de la tâche pour chaque élément constitutif du lot, nous traduirons cela par un "POUR CHAQUE nom-attribut" où nom-attribut est le nom de celui que nous avons utilisé pour traduire en DSL la taille du lot.
Par contre, si l'utilisateur ne désire qu'une seule exécution de la tâche pour tout le lot, nous n'emploieront pas cette clause.

(3) - les événements coordonnés par la condition;

Comme nous l'avons vu au point (2), ces événements sont des GENERATIONs DE MESSAGEs qui traduisent des objets informationnels dans un certain état.

(4) - les modalités de contribution d'un événement ;

La condition de synchronisation (cfr. (1)) définit la coordination d'événements contribuant à la précondition, il reste à présent à définir les modalités ( quand, combien,

durée ) selon lesquelles seront créées et mémorisées les occurrences des contributions à un point-de-synchronisation.

On distinguera deux modalités de contribution :
- (a) le nombre de contributions
- (b) la mémorisation de ces contributions

Certaines propriétés définies en MIBL reposent sur des concepts inspirés directement de DSL, ceci afin de disposer des informations nécessaires à la simulation.
C'est pourquoi, la traduction de ces propriétés en DSL est fortement facilitée.

(a) le nombre de contributions
En DSL, le nombre de contributions indique le nombre d'occurrences d'un point-de-synchronisation auquelles un événement (dans notre cas, la génération d'un message ) peut contribuer. La contribution peut être soit unique, soit multiple.

- Si la contribution est unique, une occurrence de l'événement contribue une seule fois au point-de-synchronisation; c-à-d, qu'une fois la condition de synchronisation satisfaite, il ne reste plus de traces de la contribution de l'événement.

- Si la contribution est multiple, une occurrence de la génération d'un message contribuera à un nombre illimité de réalisations du point-de-synchronisation.

La forme syntaxique utilisée est :

EST CONTRIBUE PLUSIEURS FOIS PAR GENERATION DE nom-message;

Ceci nous permet de traduire directement la propriété de contribution du MIBL. En effet, celle-ci indique le nombre d'occurrences de vérification de la précondition auxquelles peut participer la vérification d'un prédicat ( de cette précondition ) ou encore, le nombre de contributions de l'occurrence d'un OBJET dans un certain ETAT à une PRECONDITION en MIBL.

(b) la mémorisation
Cette modalité indique si une occurrence d'une contribution est mémorisable ou non. Dans le cas d'une contribution mémorisable, il faut spécifier le terme de la mémorisation. Cette modalité nous permet de traduire directement la propriété de temporisation du MIB.

La forme syntaxique DSL que nous utilisons est la rubrique MEMORISATION attachée au point-de-synchronisation. Cette rubrique contiendra des phrases DSL de la forme :

MEMORISE GEN DE nom-message PENDANT const-temps;

Cas particuliers

a) Il se peut que la précondition d'une tâche ne porte que sur un seul objet informationnel. Comme il n'y a rien à coordonner dans cette précondition, la traduction d'une telle tâche ne nécessite aucun point-de-synchronisation. Dans ce cas, le processus représentant l'exécution de la tâche sera déclenché directement par la génération du message (au moyen duquel on a traduit l'objet informationnel du MIB se trouvant dans la précondition).

Nous utiliserons alors la phrase DSL suivante dans la section processus :

EST DECLENCHE PAR GENERATION DE nom-message;

b) Un deuxième cas particulier est celui où, dans une précondition, on trouve une propriété de lot portant sur un objet dans plusieurs états.

Exemple : si la précondition d'une tâche contient le prédicat :

( DOCUMENT lettre-contact : complet ET correct,
PAR LOT DE 10 )

Dans ce cas, nous devons utiliser un autre point de synchronisation que celui utilisé pour traduire la précondition toute entière. Ce point-de-synchronisation coordonnera les messages qui sont la traduction d'un même objet mais dans des états différents. Il sera réalisé lorsque les différents messages seront générés et contribuera au point-de-synchronisation de la précondition. De plus l'opérateur COMPTER utilisé pour traduire la propriété MIB du "lot" aura pour opérande non pas la génération d'un message, mais bien la réalisation du premier point-de-synchronisation.

Si le document lettre-contact dans l'état complet a été traduit en un message DSL "m1" et que la traduction du document lettre-contact dans l'état correct, a donné lieu en DSL à la génération d'un message "m2", on peut comprendre la traduction de l'exemple

qui se fera en trois temps; d'abord la définition du point-de-synchronisation intermédiaire :

```
DEF POINT_DE_SYNCHRONISATION s1;
   CONTRIBUE PAR GEN m1;
   CONTRIBUE PAR GEN m2;
   REALISE-QUAND;
   ET
        GEN m1
        GEN m2
   ;
```

ensuite la définition du point-de-synchronisation représentant la précondition :

```
DEF POINT_DE_SYNCHRONISATION precondition;
   CONTRIBUE PAR REALISATION DE s1;
   REALISE-QUAND;
   COMPTER n_attribut
        REALISATION s1
   ;
```

enfin, la définition de l'attribut exprimant la taille du lot :

```
DEF ATTRIBUT n_attribut;
   VALEUR POUR LA DYNAMIQUE 10
                  EN REALISATION DE précondition;
```

### C : La postcondition

La postcondition exprime les propriétés des résultats après l'exécution correcte d'une tâche.

La tâche durant son exécution a consulté, manipulé, ou même créé certains objets informationnels. La plupart de ces objets ont changé d'état. Or, nous avons vu que nous utilisons l'objet DSL MESSAGE pour représenter un objet dans un certain état. Donc pour chacun des objets dont l'état a été modifié il nous faut créer un nouveau message DSL.

La "phrase" DSL utilisée sera :

```
DEF MESSAGE nom-message;
```

Contrairement à la précondition, la traduction d'une postcondition en DSL ne nécessite cependant pas de point de synchronisation. En effet, celui-ci serait automatiquement réalisé à la sortie de la tâche, car la postcondition est un prédicat qui est toujours réalisé si la tâche s'est exécutée correctement.

Pour exprimer les résultats de la tâche et donc les objets modifiés on utilisera la forme syntaxique suivante dans la section processus :

GENERE nombre nom-message [ SI [PAS] nom-condition ]

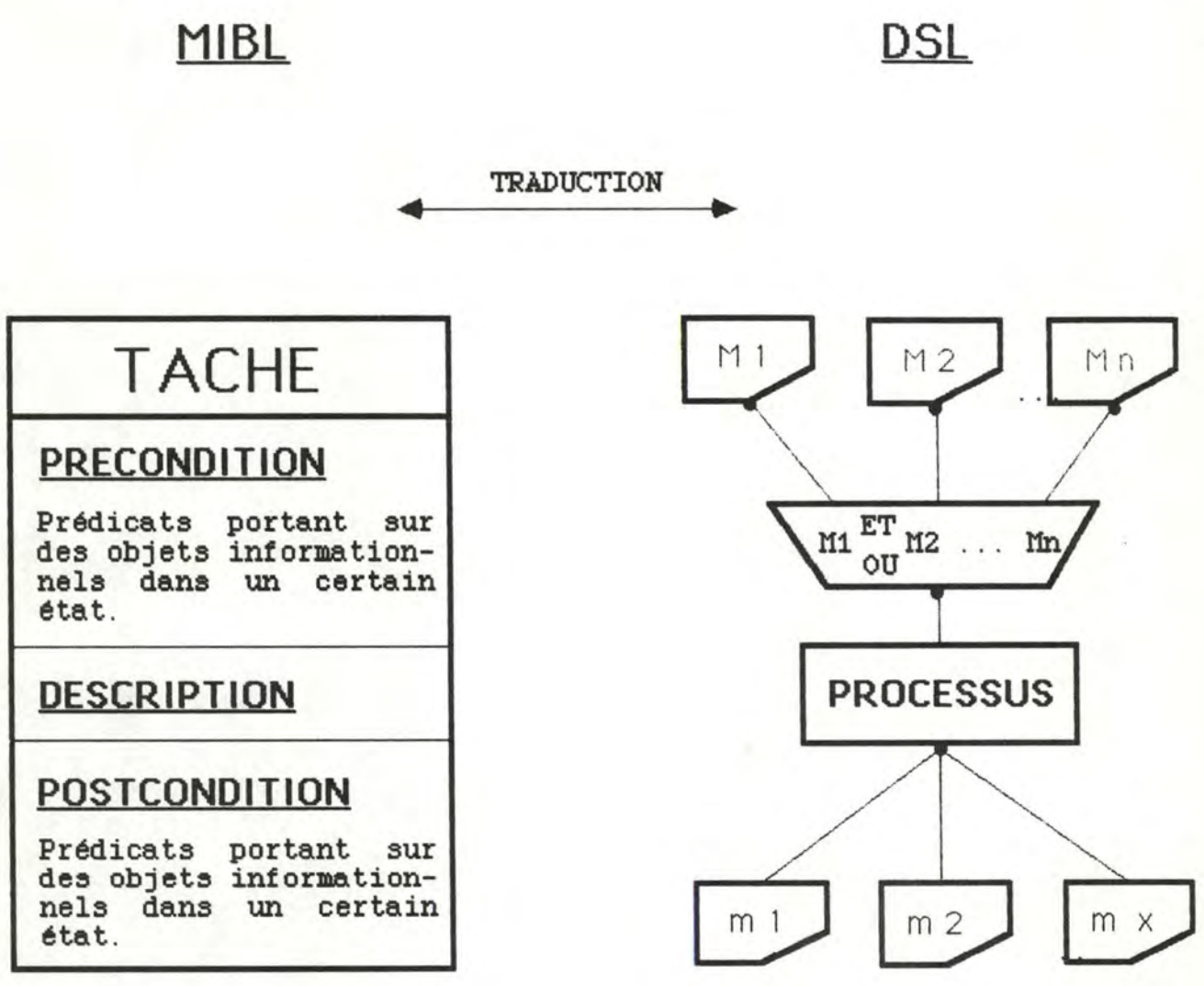
Cette relation permet d'exprimer qu'un processus génère un certain nombre de messages, et que cette génération ne s'effectue que si la condition est vérifiée.
Chacun de ces messages sera la traduction d'un objet informationnel qui a changé d'état.

Dans la phrase DSL, "nombre" sera utilisé si l'utilisateur a spécifié un lot dans sa postcondition c-à-d, la production de résultats par "paquet".

La clause optionnelle en DSL, [ SI [PAS] condition ] nous permet de traduire le fait que l'utilisateur a associé une condition à la production d'un des résultats, c'est-à-dire une restriction à la vérification du prédicat exprimant ce résultat.

Cette restriction qui s'exprime sous la forme d'une condition est régie par les mêmes principes que les conditions dans les préconditions. Sa valeur sera statistique, car une simulation au moyen de DSL-SIM, n'effectue aucun traitement sur les données.

La figure suivante présente le mécanisme général de traduction entre les concepts de base de notre dynamique et ceux de DSL :



## 6.3.2. Les ressources

En MIBL, on considère trois types de ressources : les personnes, la technologie et les ressources consommables.
On les retrouve dans le modèle des ressources du MIB. Ce modèle n'intervient pas directement dans la spécification du fonctionnement d'un SI, mais est nécessaire pour évaluer la faisabilité de celui-ci.

La traduction de ces ressources en DSL s'effectuera aisément, puisque les informations s'y rapportant en MIBL sont très proches de celles développées en DSL. En conséquence, on trouvera directement dans le MIB les éléments nécessaires au processus de traduction.

Nous présentons ici de manière synthétique les éléments du modèle des ressources de DSL nécessaires pour utiliser le simulateur.

### A : LES RESSOURCES

#### (1) Le PROCESSEUR

Le concept de ressource DSL recouvre l'ensemble des moyens ( hommes, machines ...) nécessaires au déroulement des processus. Ces moyens sont modélisés indifféremment au moyen de l'objet DSL PROCESSSEUR. Les PROCESSEURS sont réutilisables.

C'est à ce niveau qu'il existe une différence entre DSL et MIBL, car pour ce dernier, les ressources réutilisables sont scindées en deux concepts : PERSONNEs et TECHNOLOGIE. Une fois traduit en DSL ces objets deviennent des PROCESSEURS.

#### (2) Les Requetes

Les processeurs peuvent être requis soit par des processus, soit par d'autres processeurs, de la même manière que dans le MIB, les personnes ou la technologie peuvent être requises soit par des tâches, soit par d'autres personnes ou une autre technologie.

La forme syntaxique que nous utilisons est :

REQUIERT nombre UNITES DE nom-processeur;
(dans les sections PROCESSUS et PROCESSEUR)

Le principe de la réquisition partagée n'a pas été retenu dans le MIBL.

(3) propriétés des ressources

les deux propriétés envisagées et retenues sous la même forme syntaxique que dans le MIB sont : les calendriers de disponibilité, la capacité.

-Qu'ils soient associés aux processeurs en DSL ou aux personnes et à la technologie en MIBL, les calendriers permettent de spécifier les plages de disponibilité des ressources.

La forme syntaxique utilisée est :

DISPONIBLE PENDANT nom-calendrier;
(dans les sections PROCESSEURS)

-La capacité est le concept qui permet d'exprimer la limite maximum de la somme des taux des requêtes susceptibles d'être satisfaites simultanément.

La forme syntaxique DSL que nous utilisons est :
CAPACITE nombre;
(dans les sections PROCESSEURS)

B : L'ASPECT PROCESSEUR DES PROCESSUS

L'aspect processeur des processus concerne :
- la durée des processus,
- leur priorité,
- les réquisitions de processeurs.

(1) La durée d'activité d'un processus

La durée d'activité du processus est la durée pendant laquelle il s'exécute; tout comme la durée d'une tâche est la durée pendant laquelle elle s'exécute.

La phrase DSL utilisée est :

EST EXECUTE PENDANT durée;

(2) La priorité des processus

La priorité des processus concerne la priorité d'allocation des processeurs qui leur seront alloués. Elle peut être simple, ou préemptive. La priorité est un entier compris entre 0 et 100. Le même concept existe en MIBL.

La phrase DSL utilisée est :

PRIORITE valeur-priorité; (si la priorité est simple)
PRIORITE ABSOLUE valeur-priorité; (si la priorité est préemptive)
(Dans les sections PROCESSUS)

(3) les requisitions de processeurs par les processus

Cet aspect a déja été envisagé au point 6.3.2.A(1).

## 6.3.3. Les quantifications

Ce sont les quantifications qui vont permettre d'évaluer les performances du système décrit. Elles concernent surtout l'affectation de valeurs utiles à la simulation.

### A : CALENDRIER, INTERVALLE-JOUR, INTERVALLE-HEURE

Toutes les notions permettant d'exprimer les périodes d'activité des CALENDRIERS ainsi que la durée des différents INTERVALLES sont définies en DSL de la même manière que dans le MIB.
La traduction est donc immédiate.

### B : LES MESSAGES EXTERNES

Le flux des messages externes est le nombre de messages qui franchissent la frontière du SI par unité de temps. Cette information est nécessaire afin de pouvoir injecter des données dans le système à simuler. Il faut donc spécifier la survenance d'objets externes au système ou du moins à l'activité décrite. C'est pourquoi la propriété de survenance en MIBL est inspirée directement de DSL.

En DSL, la clause permettant de spécifier la survenance d'un MESSAGE externe est:

```
SURVIENT     CHAQUE  période     DE  nom-interface
              nombre     FOIS

                        PENDANT   nom-calendrier;
                                  (Dans la section MESSAGE)
```

C : ATTRIBUT

Bien que l'on retrouve en MIBL l'objet ATTRIBUT, il n'y sera cependant utilisé que dans un rôle descriptif, afin de pouvoir compléter le système décrit à l'aide de caractéristiques non prévues initialement par le MIB.

Nous utiliserons l'objet DSL ATTRIBUT lors de la traduction d'une propriété MIBL de "lot". En effet, pour traduire la propriété de "lot" nous utilisons l'opérateur DSL COMPTER qui nécessite l'emploi d'un attribut afin de spécifier le nombre d'événements à compter. Un attribut DSL sera donc utilisé pour traduire la taille d'un lot dans une précondition.

Sa définition se fera au moyen de la phrase DSL suivante :

```
DEF ATTRIBUT nom-attribut;
     VALEUR POUR LA DYNAMIQUE nombre-éléments-du-lot EN
               REALISATION DE nom-point-de-synchronisation;
```

D : CONDITION

A chaque condition référencée dans une pré ou postcondition, doit être affectée une valeur probabilistique afin de pouvoir traduire ces conditions en DSL. Les conditions MIBL obéissent aux mêmes règles syntaxiques et sémantiques que les conditions DSL.

## 6.4. Exemple de traduction

Voyons à présent comment les tâches constitutives de l'activité décrite dans l'exemple de base peuvent être traduites en DSL, du moins en ce qui concerne leurs pré et postconditions.

Nous allons appliquer les principes développés ci-dessous à quelques tâches représentatives. La traduction complète de l'exemple tout entier est définie en annexe.

Remarque : le nom qui se trouve à coté de l'objet informationnel dans les pré et postconditions est le nom de l'objet DSL au moyen duquel on le traduit.

TACHE 5: vérification demande inscription complète

- NOM : vérification-complet

- PRECONDITION :

( FORMULAIRE demande-inscription : rempli ) m1

- POSTCONDITION :

OU

( FORMULAIRE demande-inscription : complet, m2
SI cond-complet )

( FORMULAIRE demande-inscription : à-compléter ) m3

- TRADUCTION EN DSL :

la génération du message DSL correspondant au formulaire demande-inscription dans l'état rempli (m1) a été effectuée lors de la traduction de la tâche dont il était le résultat.
De plus, au niveau de la précondition, on trouve uniquement un seul prédicat portant sur un objet informationnel : il n'est donc pas nécessaire de définir un point de synchronisation en DSL.
Le processus est directement déclenché par la génération du message (m1).

DEF MESSAGE m2 ;

DEF MESSAGE m3 ;

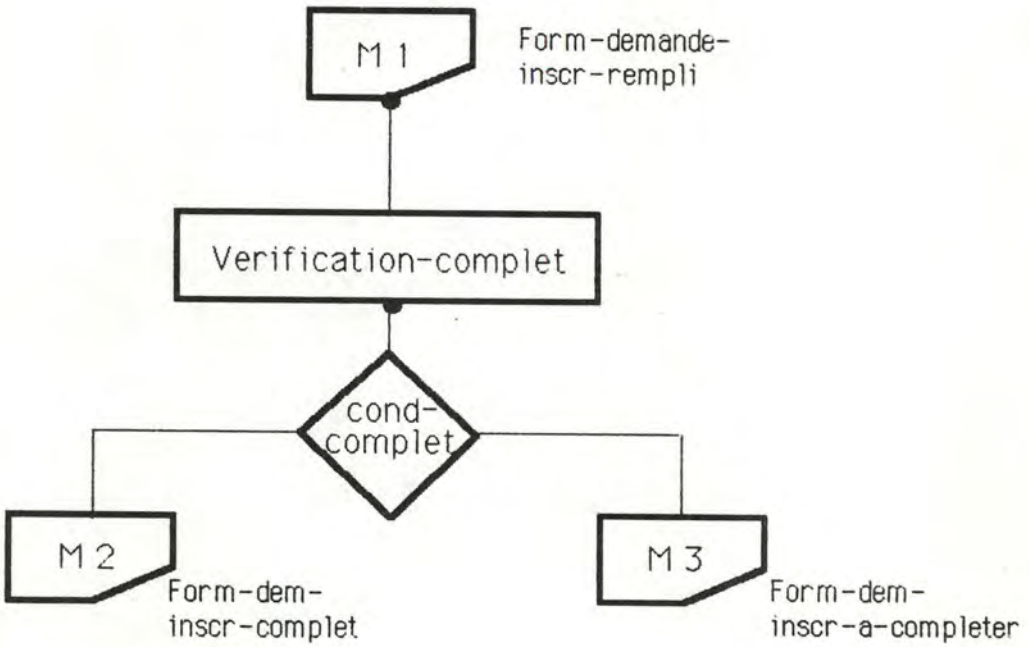
```
DEF PROCESSUS verification-complet ;
    EST DECLENCHE PAR GENERATION DE m1 ;
    GENERE m2 SI cond-complet ;
    GENERE m3 SI PAS cond-complet ;
```

La condition cond-complet est définie dans "l'annexe technique au MIB" sous la forme :

```
DEFINIR     CONDITION cond-complet;
            PROBABILITE VRAIE 0.90;
```

Nous avons choisi cette façon d'exprimer une condition, car elle est très proche de toute autre définition d'un objet du MIB.
En DSL, un objet "condition" se décrit exactement de la même manière, et aucun processus de traduction n'est nécessaire à ce niveau.

- SCHEMA DE LA DYNAMIQUE DSL :



TACHE 6: classement de demande d'inscription

- NOM : classer_demande_inscription

- PRECONDITION :

ET

( FICHIER inscription : créé,
SURVIENT 1 FOIS DE administration PENDANT rentrée,
CONTRIBUE PLUSIEURS FOIS,
RETENU PENDANT "60d" ) m4

( FORMULAIRE demande-inscription : complet ET correct,
PAR LOT DE 5 ) m5 m6

- POSTCONDITION :

( FICHIER inscription MAJ+ PAR
FORMULAIRE demande-inscription : classé-fich-inscr,
PAR LOT DE 5 ) m7

- TRADUCTION EN DSL :

les messages m5 et m6 sont internes à l'activité et sont les résultats d'une autre tâche. Ils ont été définis lors de la traduction de cette dernière.
Par contre, le message m4 est externe; il faut définir dès maintenant l'interface qui indique sa provenance (cette notion n'apparait pas dans le MIB). Ultérieurement, on traduira tous les calendriers du MIB en DSL, ce qui permettra de définir "rentrée".

DEF MESSAGE m4 ;
SURVIENT 1 FOIS DE administration PENDANT rentrée ;

DEF INTERFACE administration ;

On remarque que dans cette précondition, il y a deux types de synchronisation :
- un premier qui synchronise tous les formulaires de demande d'inscription complets et corrects
- un second qui compte cinq formulaires parmi ceux-ci.

En DSL, ces deux niveaux de synchronisation ne peuvent être exprimés dans un même objet POINT-DE-SYNCHRONISATION, alors qu'en MIB, on les retrouve dans la même précondition.

Il nous faut donc définir un point de synchro supplémentaire (s1) dans le but de réunir les formulaires dans l'état complet et dans l'état correct. La réalisation de ce point de synchronisation contribuera directement à la condition du suivant et sera entièrement transparente à l'utilisateur.

```
DEF POINT-DE-SYNCHRO s1 ;
    EST CONTRIBUE PAR GENERATION DE M5 ;
    EST CONTRIBUE PAR GENERATION DE m6 ;
    EST-REALISE-QUAND ;
          ET
              GEN m5
              GEN m6 ;
    EN REALISATION CONTRIBUE A s2 ;

DEF POINT-DE-SYNCHRO s2 ;
    EST CONTRIBUE PLUSIEURS FOIS PAR GENERATION DE m4 ;
    EST CONTRIBUE PAR REALISATION DE s1 ;
    EST REALISE QUAND ;
        ET
            GEN m4
            COMPTER  n-lot-1  REALISATION DE s1 ;
    MEMORISATION;
        MEMORISE GENERATION DE m4 PENDANT "60d";
    EN REALISATION DECLENCHE classer-demande-inscription ;



DEF MESSAGE m7 ;

DEF PROCESSUS classer-demande-inscription ;
    GENERE  5  m7 ;
```

En DSL, pour exprimer le fait que l'on désire considérer un certain nombre de formulaires avant l'exécution du traitement, on indique dans la condition de synchronisation qu'il faut en compter un certain nombre.

La syntaxe du langage DSL nous oblige à définir un attribut pour exprimer ce nombre, en choisissant sa "valeur pour la dynamique" de telle façon qu'elle égale la taille du lot désiré lors de la simulation ( dans notre exemple : 5 ).

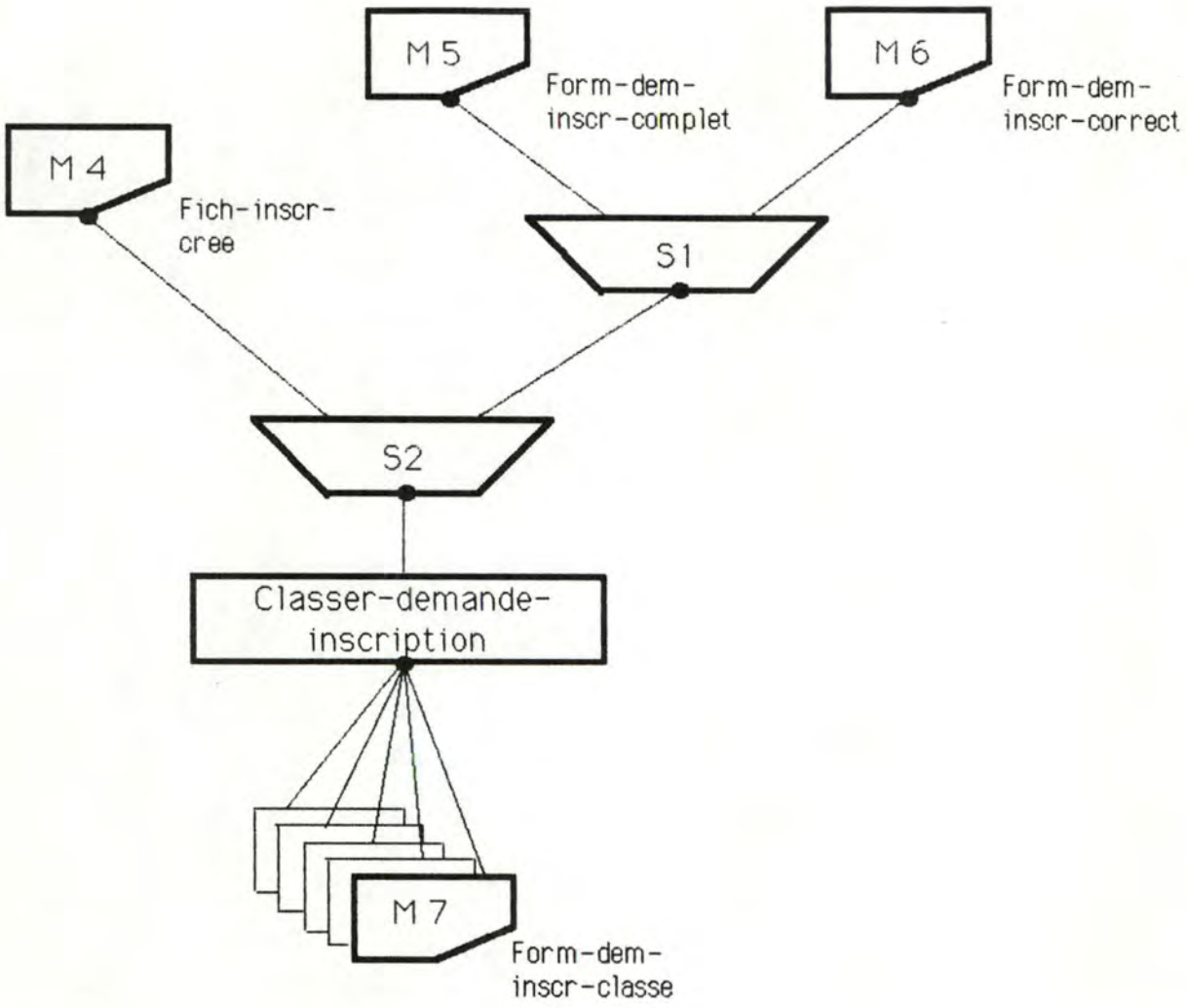
```
DEF ATTRIBUT n-lot-1 ;
    VALEUR POUR LA DYNAMIQUE 5
           EN REALISATION DE s2 ;
```

On exécutera le processus, autant de fois qu'il y a eu de réalisations du point de synchronisation s2, puisque nous

avons spécifié un "traitement par lot". Dans le cas d'un "traitement individuel", nous aurions traduit :

```
EN REALISATION DECLENCHE classer-demande-inscription
                POUR CHAQUE n-lot-i ;
```

- SCHEMA DE LA DYNAMIQUE DSL :



TACHE 7: ouverture du dossier

- NOM : ouverture_dossier

- PRECONDITION :

ET

( FORMULAIRE demande-inscription : complet ET correct )
m5 m6
( MESSAGE rentrée-académique : survenu, m8
SURVIENT UNE FOIS DE administration PENDANT rentrée,
RETENU PENDANT "60d",
CONTRIBUE PLUSIEURS FOIS )

- POSTCONDITION :

ET

( DOSSIER étudiant : ouvert ) m9

ET

(FORMULAIRE demande-inscription : classé-dossier-etd) m10
(DOCUMENT lettre-contact : classé-dossier-etd) m11

- TRADUCTION EN DSL :

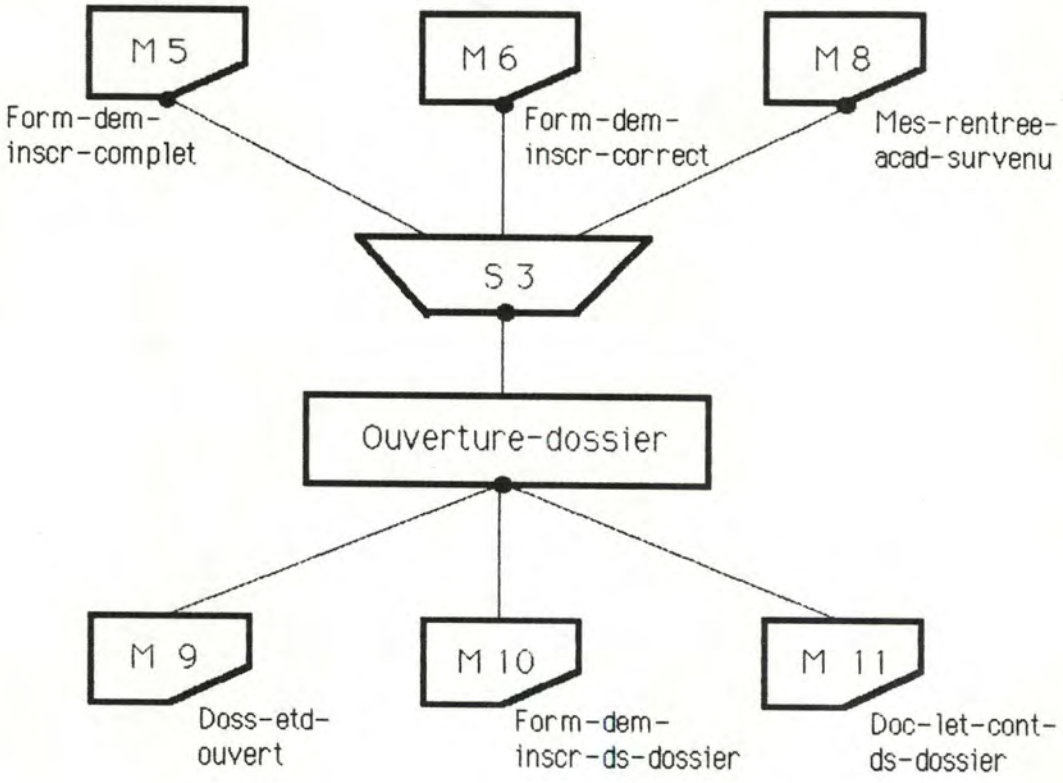
```
DEF MESSAGE m8 ;
    SURVIENT 1 FOIS DE administration PENDANT rentrée ;

DEF POINT-DE-SYNCHRO s3 ;
    EST CONTRIBUE PLUSIEURS FOIS PAR GENERATION DE m8 ;
    EST CONTRIBUE PAR GENERATION DE m5 ;
    EST CONTRIBUE PAR GENERATION DE m6 :
    EST-REALISE-QUAND ;
          ET
              GEN m8
              ET
                 GEN m5
                 GEN m6 ;
    MEMORISE GEN DE m8 PENDANT "60d" ;
    EN REALISATION DECLENCHE ouverture-dossier ;

DEF PROCESSUS ouverture-dossier ;
```

```
GENERE m9 ;
GENERE m10 ;
GENERE m11 ;
```

- SCHEMA DE LA DYNAMIQUE DSL :



TACHE 7bis: ouverture du dossier bis

Ajoutons cette tâche qui n'a peut-être pas de sens au niveau de l'activité décrite, mais qui illustre bien l'expression d'un condition portant sur un prédicat de précondition.

- DESCRIPTION :

  La tâche considérée a les mêmes spécifications que la tâche "ouverture_dossier" décrite ci-dessus, hormis le fait qu'elle ne doit plus s'exécuter pour tous les formulaires de demande d'inscription complets et corrects, mais seulement pour une portion de ceux-ci (70%).

- NOM : ouverture_dossier_bis

- PRECONDITION :

  ET

  ( FORMULAIRE demande-inscription : complet ET correct,
  SI cond-ouverture-dossier ) m5 m6

  ( MESSAGE rentrée-académique : survenu, m8
  SURVIENT UNE FOIS DE administration PENDANT rentrée,
  RETENU PENDANT "60d",
  CONTRIBUE PLUSIEURS FOIS )

- POSTCONDITION :

  ET

  ( DOSSIER étudiant : ouvert ) m9

  ET

  (FORMULAIRE demande-inscription : classé-dossier-etd) m10

  (DOCUMENT lettre-contact : classé-dossier-etd) m11

- TRADUCTION EN DSL :

  DEF MESSAGE m8 ;
  SURVIENT 1 FOIS DE administration PENDANT rentrée ;

  La condition cond-ouverture-dossier est définie dans une

"annexe technique au MIB" sous la forme :

```
DEFINIR     CONDITION cond-ouverture-dossier;
            PROBABILITE VRAIE 0.70;


DEF POINT-DE-SYNCHRO s3 ;
    EST CONTRIBUE PLUSIEURS FOIS PAR GENERATION DE m8 ;
    EST  CONTRIBUE  PAR GENERATION DE  m5
    EST  CONTRIBUE  PAR  GENERATION DE m6

    EST-REALISE-QUAND ;
          ET
              GEN m8
              ET
                 GEN m5 SI  cond-ouverture-dossier
                 GEN m6 SI  cond-ouverture-dossier ;
    MEMORISE GEN DE m8 PENDANT "60d" ;
    EN REALISATION DECLENCHE ouverture-dossier ;


DEF PROCESSUS ouverture-dossier-bis ;
    GENERE m9 ;
    GENERE m10 ;
    GENERE m11 ;
```

La tâche sera uniquement déclenchée lorsque les messages m4, m5 et m6 auront été générés
et que la condition "cond-ouverture-dossier" aura été vérifiée, c'est à dire dans 70% des cas.

## 6.5. Conclusion

Ce chapitre a permis de montrer non seulement que notre langage permettait d'exprimer les structures d'enchaînement élémentaires de DSL, mais encore qu'il était possible de concevoir un processus automatique de traduction de MIBL en DSL;

Nous allons maintenant présenter les fonctionnalités et l'architecture logique du programme automatisant le processus de traduction.

# CHAPITRE VII

# CONCEPTION DU PROGRAMME DE TRADUCTION

# ET

# ARCHITECTURE LOGIQUE

## 7.1. Introduction

Après avoir étudié les principes de traduction de MIBL en DSL, ce chapitre examine comment réaliser le logiciel d'interfaçage avec le logiciel de simulation. Comme nous devons nous intégrer dans un environnement existant, une première partie de ce chapitre présente cet environnement.
Dans un deuxième temps nous situons le programme de traduction ainsi que ses fonctionnalités.
Enfin, nous développerons l'architecture logique de ce programme (spécifications externes).

## 7.2. Environnement existant

L'environnement dans lequel vient s'insérer notre programme de traduction est composé du MIB, de IDA, et du simulateur DSL-SIM.

### 7.2.1. Le MIB

Comme nous l'avons déja vu, MIB est un modèle de spécification, mais également un environnement logiciel.

Il possède donc :

- une base de données des spécifications,
- un langage permettant de les exprimer, et
- des outils d'enregistrement, de validation, et d'extraction de spécifications.

### 7.2.2. IDA

Nous avons déja présenté IDA (cfr. chapitre VI) ainsi que son langage d'expression des spécifications, DSL.

IDA, commme MIB, est également un ensemble de modèles, de méthodes et d'outils. Il possède donc également une base de données, un langage, et des outils généraux d'enregistrement, d'extraction et de validation des spécifications.

De plus, IDA dipose de deux outils spécifiques au langage DSL :
DSL_PROTO, un outil de prototypage et
DSL_SIM, l'outil de simulation que nous allons utiliser.

Voyons à présent plus en détail, comment fonctionne ce simulateur.

## 7.2.3. DSL-SIM

Le simulateur DSL-SIM est composé de trois éléments :

### A : LA GENERATION

Le programme de génération permet de dériver un programme de simulation, à partir des spécifications dynamiques ( spécifications du fonctionnement) et de la spécification des ressources.

### B : L'EXECUTION DE LA SIMULATION

L'objectif de cette exécution est de simuler le comportement spécifié pendant une période donnée, et de collecter toute une série de mesures sur la modélisation du fonctionnement qui a été simulée.

La simulation fournit un résultat très intéressant pour notre étude : le fichier de "trace" qui reprend l'historique de tous les événements qui se sont produits (déclenchement et terminaison de processus, génération de messages). Ce type de résultat, une fois adapté aux objets MIB, permettra de vérifier si l'enchaînement des tâches tel qu'il s'est produit correspond à ce qui était attendu.

### C : L'EXPLOITATION DES RESULTATS DE LA SIMULATION

L'outil d'exploitation des résultats permet d'extraire les résultats statistiques fournis par la simulation. Ces résultats permettent de comparer les performances du fonctionnement établies par le simulateur avec celles initialement prévues par l'utilisateur.

## 7.2.4. Analyse et intérêt des résultats du simulateur

Examinons à présent quels sont exactement les résultats de DSL-SIM et l'intérêt qu'ils présentent dans le cadre de notre travail.

Ces résultats sont soit, "globaux", ce sont alors des valeurs agrégées (moyennes, extrêmes, ...), soit "chronologiques", ce sont alors des valeurs éclatées par tranches de temps de simulation.

Les résultats concernent les processus, les ressources, les points-de-synchronisation; et donc une fois traduits en MIBL, ils concerneront, les tâches, les ressources et les préconditions.

### A : TACHE

Les résultats attendus pour une tâche sont :

1) le nombre de déclenchements, de (re)démarrages, d'interruptions ou de terminaisons

2) les durées telles que durée d'exécution, d'interruption, et temps mort,

3) la répartition des tâches suivant leur état.

Ces résultats sont importants dans le cadre de notre travail afin d'examiner si le fonctionnement modélisé est réalisable et performant en fonction des ressources.

### B : RESSOURCES

Les résultats concernent :

1) le nombre d'allocations et de désallocations,

2) la quantité utilisée,

3) le nombre de tâches en attente.

Ces informations permettent de déterminer si l'agencement des tâches est "intéressant". Dans le cas contraire, si la modélisation doit être modifiée, les premiers changements à envisager ( avant de modifier le fonctionnement ) sont l'ajustement de certaines ressources.

C : PRECONDITION

Les résultats interprétés en terme d'objets du MIB concernent :

1) le nombre de vérifications des prédicats constituant la précondition,

2) le temps de participation ( différence entre la vérification d'un prédicat et la vérification de la précondition qui le contient),

3) le temps de réalisation ( différence entre la vérification du premier prédicat de la précondition et la vérification de la précondition toute entière ),

4) le caractère plus ou moins contraignant des prédicats participants à la précondition. Un prédicat est considéré comme contraignant si son temps de participation est court.

L'analyse de ces résultats permet de déterminer les contraintes inhérentes au déclenchement d'une tâche.

D : EXTENSIONS INTERESSANTES

Ces résultats sont ceux de DSL-SIM traduits en terme de nos concepts. Plusieurs extensions sont donc intéressantes bien que non réalisables étant donné que nous ne pouvons modifier le simulateur.

Ces extensions devraient nous permettre d'analyser le cycle de vie des objets informationnels à travers l'activité simulée.

Nous avons situé l'environnement de notre travail, voyons maintenant comment va s'y intégrer notre logiciel de traduction et présentons ses fonctionnalités.

## 7.3. Programme de traduction

Le programme que nous concevons et développons doit permettre à l'utilisateur du MIB de simuler la modélisation du fonctionnement de son SIB, qu'il a effectuée en MIBL; ceci afin de l'évaluer et éventuellement de la corriger.

Le fonctionnement de notre programme reprend les étapes suivantes :

- Extraire de la base de données MIB les spécifications relatives au fonctionnement et aux ressources.

- Vérifier si la modélisation est syntaxiquement et sémantiquement valide.

- Générer sur base des principes de traduction présentés dans le chapitre VI, une spécification DSL de la dynamique et des ressources, ensuite, introduire celle-ci dans la base de données DSL.

- générer une table de conversion des objets MIB (rencontrés dans la spécification de l'utilisateur) en objets DSL.
En effet, les concepts et donc les objets utilisés sont différents en MIBL et en DSL. Cette table sera utile pour effectuer la conversion des résultats dans le langage de l'utilisateur; c-à-d, permettre à celui-ci d'exploiter les résultats portant sur les objets qu'il a spécifiés.

  Exemple : si l'utilisateur désire des renseignements sur la précondition de la tâche "traiter-contacts" il faut pouvoir traduire cela pour le simulateur en "point-de synchronisation x". De la même manière les résultats du simulateur sont exprimés en terme de "message xyz" et non en terme de DOSSIER "étudiant dans l'état ouvert".

- permettre à l'utilisateur d'exploiter ses résultats; notamment grâce à la table de conversion, qui permettra de rendre complètement invisible par exemple, le message dsl "xyz" en le traduisant en "dossier étudiant dans l'état ouvert".

Le fonctionnement de notre programme dans son environnement peut être exprimé au moyen de la figure 7.1.

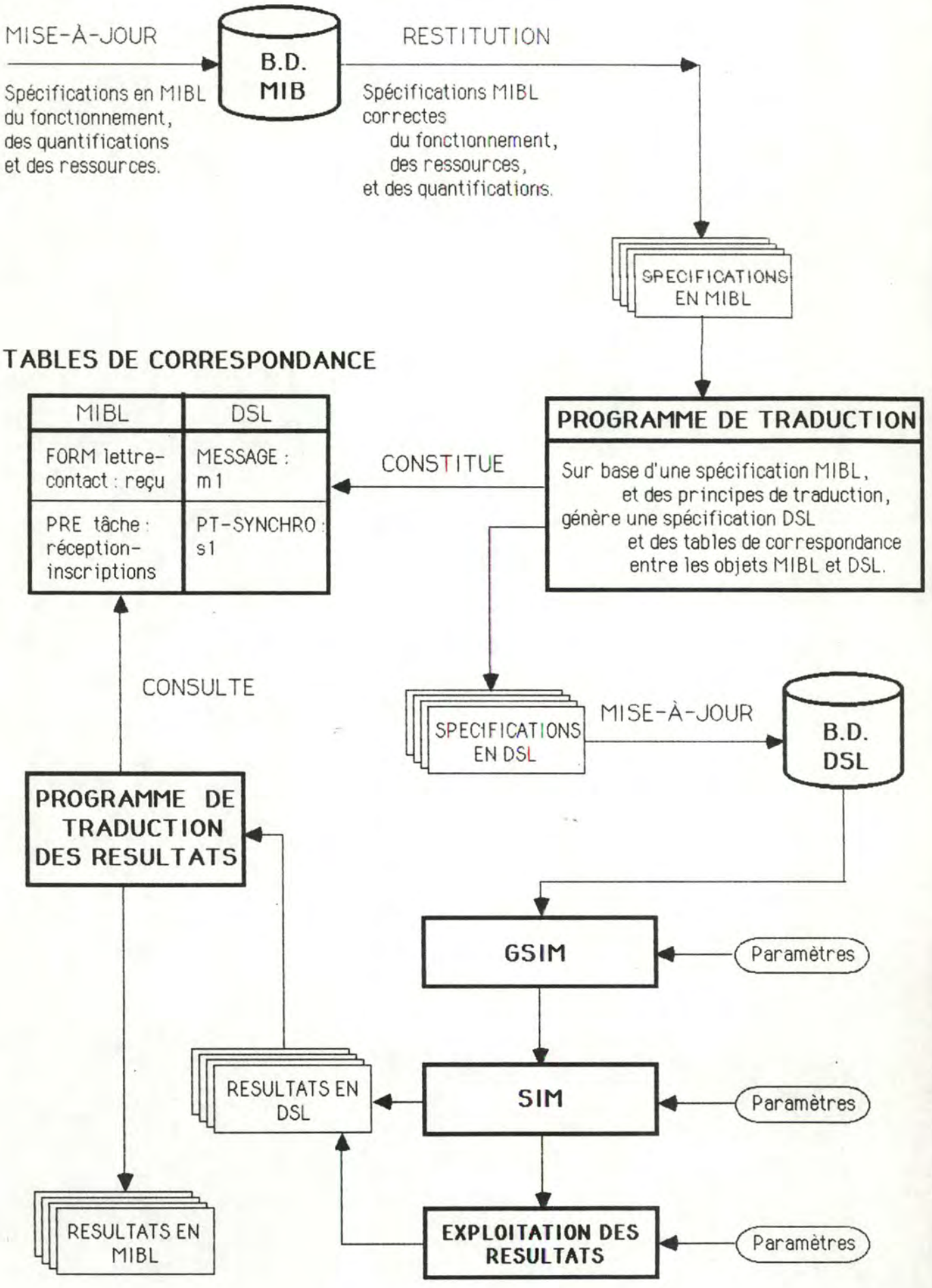



Fig. 7.1. Fonctionnement du système

## 7.4. Architecture logique du programme de traduction

Nous présentons à présent l'architecture logique du logiciel implémentant les fonctionnalités décrites au point 7.3..
La conception de cette architecture repose sur les principes exposés dans [VANLAM1, 86].

### 7.4.1. La démarche de conception

La première démarche de conception d'une architecture logique est de hiérarchiser le système en différents niveaux, il faut ensuite structurer chacun des niveaux en modules. Enfin, pour chacun des modules, nous spécifierons les interfaces aux moyens desquelles on peut y accéder.

#### La hiérarchisation

La hiérarchie choisie est la hiérarchie "utilise". Celle-ci se définit comme suit :

Un système hiérarchisé de type "utilise" est un système pour lequel :
"Si A et B sont deux composants du système, on dira que A utilise B si, et seulement si, le fonctionnement correct de A dépend de la disponibilité d'une version correcte de B."

#### La structuration modulaire

Après avoir hiérarchisé le sytème en niveaux, il faut ensuite identifier et spécifier de manière précise les composants ou modules de chaque niveaux.

La découpe en modules se fera selon les critères suivants :

- la spécification de chaque module pourra se faire d'une manière simple et précise

- chaque module offrira :
  - une forte capacité de cacher de l'information,
  - une forte cohésion interne,
  - un faible degré de couplage.

Spécification des modules

Un module n'est visible de l'extérieur que par ses interfaces. Nous allons donc, pour chacun des modules, spécifier ses interfaces, par assertions.

## 7.4.2. La hiérarchisation

Nous allons suivre la démarche présentée dans [VANLAM1,86] en l'adaptant à notre système afin de le hiérarchiser en plusieurs niveaux.

Le niveau six :

il est dérivé des fonctions identifiées dans l'analyse fonctionnelle. Dans le cadre de notre système de traduction, ce niveau traitera les trois aspects nécessaires à la simulation : fonctionnement (la tâche), ressources et quantifications.

Le niveau cinq :

il permet d'isoler un noyau fonctionnel de base utilisé par le niveau six. Dans notre système, ce niveau permettra la décomposition du traitement des différentes tâches suivant les informations contenues dans les attributs de celles-ci (précondition, postcondition, durée, priorité, réquisition de ressources).

Le niveau quatre :

ce niveau, utilisé par les niveaux supérieurs, permettra l'analyse et la validation syntaxique des pré et postconditions, ainsi que leur analyse sémantique.

Le niveau trois :

il est utilisé par les niveaux supérieurs et va nous permettre l'accès aux spécifications MIB, la génération de spécifications DSL, et la gestion des correspondances entre les objets des deux langages. En quelque sorte, ce niveau assure les entrées sorties de notre système.

Le niveau deux :

les composants de ce niveau assurent le comportement dynamique du système. Son but est d'agencer correctement les composants des niveaux supérieurs. Le concept de scénario d'exécution du système n'a aucun sens en dehors de ce niveau.

## 7.4.3. Découpe des niveaux en modules

Examinons à présent pour chacun des niveaux et en fonction de la spécialité qui leur a été attribuée en 7.4.2., comment chacun d'entre eux va être structuré en modules.

NIVEAU 6

### Module TACHE

Le module TACHE doit effectuer le traitement complet d'une tâche MIB ( si celle-ci existe ) ainsi que de toutes ses relations. Ce module va s'occuper de la traduction d'une tâche MIB suivant les principes définis au chapitre **VI.**

Afin de fonctionner correctement, TACHE utilise les modules PRECONDITION, POSTCONDITION, NOM_TACHE, REQRESS_DUREE_PRIORITE, ACCES_MIB.

### Module RESSOURCES ET QUANTIFICATIONS

Ce module assure le traitement des ressources et des quantifications. Nous avons regroupé ces deux traitements qui fonctionnellement sont dissemblables, car le principe de traduction des ressources diffère très peu de celui des quantifications. De plus, la définition de ces objets en MIBL est très proche de celle de DSL.

Ce module, afin de fonctionner correctement, utilise le module : GENERE_DSL.

NIVEAU 5

### Module PRECONDITION

Le module précondition assure la validation syntaxique et sémantique, ainsi que la traduction de la précondition d'une tâche donnée.

Ce module afin de fonctionner correctement, utilise les modules : VALIDE_SYNTAXE, VALIDE_SEMANTIQUE, CORRESPONDANCE, GENERE_DSL, ACCES_MIB.

### Module POSTCONDITION

Le module postcondition assure la validation syntaxique et sémantique, ainsi que la traduction de la postcondition d'une

tâche donnée.

Ce module afin de fonctionner correctement, utilise les modules : VALIDE_SYNTAXE, VALIDE_SEMANTIQUE, CORRESPONDANCE, GENERE_DSL, ACCES_MIB.

Module NOM TACHE

Ce module assure le traitement et la traduction du nom de la tâche.

Ce module afin de fonctionner correctement, utilise le module : GENERE_DSL.

Module REQRESS DUREE PRIORITE

Ce module assure le traitement la validation syntaxique et sémantique, ainsi que la traduction des réquisitions de ressources, de la durée, et de la priorité d'une tâche donnée. Nous avons regroupé ces relations car leur traitement est fort semblable.

Ce module afin de fonctionner correctement, utilise les modules : VALIDE_SEMANTIQUE, GENERE_DSL, ACCES_MIB.

NIVEAU 4

Module VALIDE SYNTAXE

Le module valide_syntaxe assure l'analyse et la validation syntaxique des pré et postconditions d'une tâche.

Module VALIDE SEMANTIQUE

Ce module assure la validation sémantique des informations nécessaires à la simulation. Ce module est chargé de la détection d'incohérences au niveau de la description du fonctionnement ainsi que de la complétude des spécifications.

Ce module afin de fonctionner correctement, utilise les modules : CORRESPONDANCE, ACCES_MIB.

NIVEAU 3

Module ACCES MIB

Le module acces_mib permet l'accès aux spécifications MIB.

Il offre aux modules supérieurs des primitives leur permettant d'accéder aux objets et relations utiles pour la traduction.

La structure de données dans laquelle est enregistrée la base de données MIB est "cachée" dans ce module et est donc transparente pour tous les modules utilisant accès_mib.

Module CORRESPONDANCE

Ce module assure la gestion des correspondances entre les objets MIBL et les objets DSL.
Il permet aux modules supérieurs d'enregistrer et de consulter facilement les correspondances sans connaître la structure de données dans laquelle elles sont enregistrées. Cette structure est en fait transparente à tout module utilisant le module correspondance.

Il permet également un contrôle de faisabilité au niveau des spécifications.
En effet, ce module permet de détecter le cas où un prédicat de précondition porte sur un objet informationnel dans un certain état, mais aucun prédicat de postcondition ne stipule que cet objet va se trouver, à un moment donné, dans cet état. On attend donc un objet informationnel qui n'atteindra jamais un état dans le contexte de l'activité décrite. Dans le cadre de l'évaluation du fonctionnement de cette activité, la tâche qui requiert cet objet ne pourra jamais s'exécuter.

Module GENERE DSL

Le module génère_dsl assure la traduction en DSL. Il offre, en fait, une série de primitives permettant de générer la définition (en tout ou en partie) d'objets ou de relations DSL sans en connaître la syntaxe. Cette syntaxe est cachée dans ce module et est donc transparente pour tous les modules supérieurs.

NIVEAU 2

Module COORDINATEUR

Ce module assure le comportement dynamique du système. Il n'utilise et n'est utilisé par aucun module; mais à l'implémentation et à l'exécution du système, il réglera le bon agencement des modules supérieurs par des relations d'appels. Le concept de scénario d'exécution du système n'a aucun sens en dehors de ce module.

La figure 7.2. représente l'architecture logique de notre système.

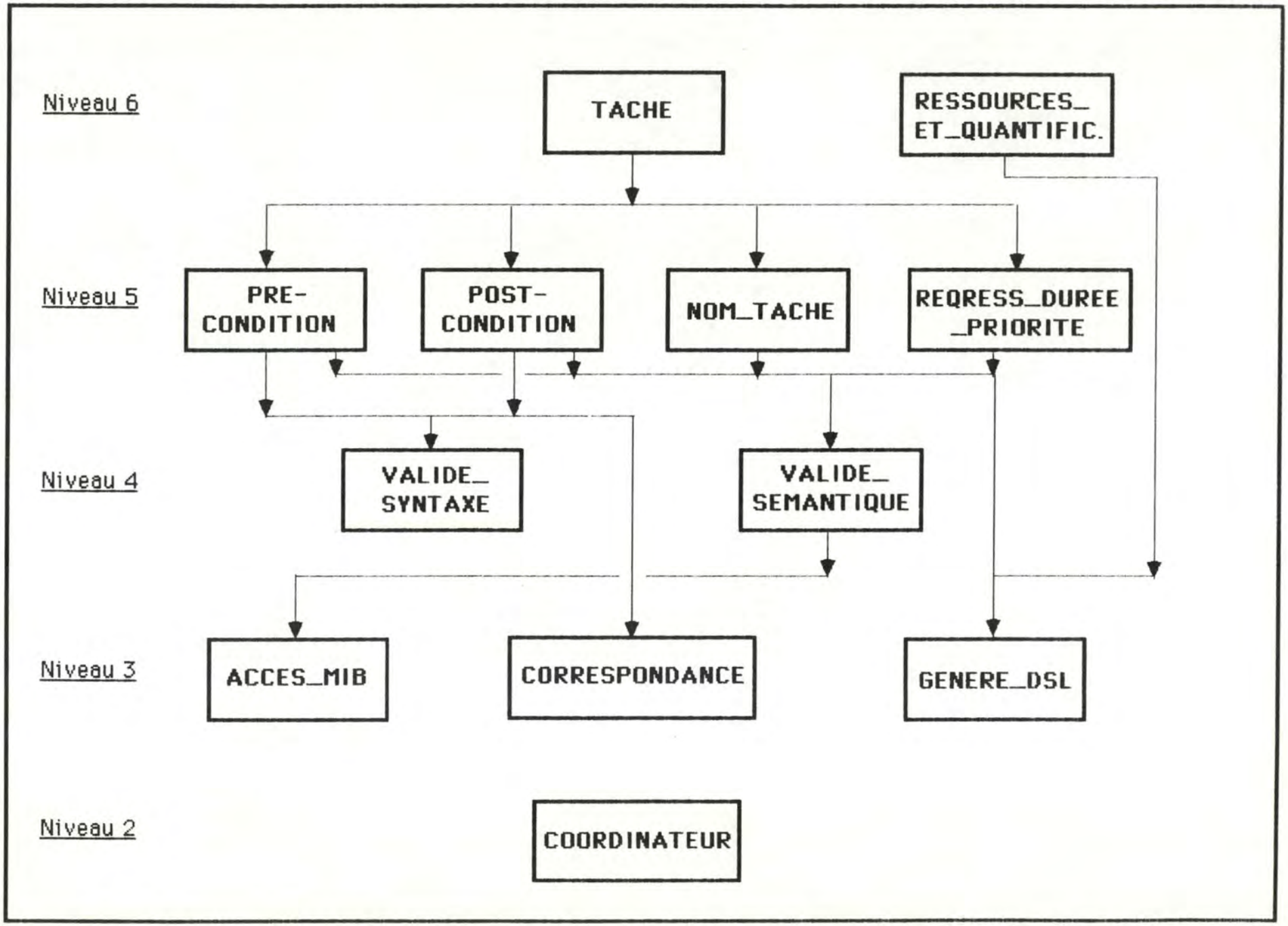



Fig. 7.2. Architecture logique

## 7.4.4. Spécification des modules

Nous allons reprendre chaque module défini en 7.4.3. et en spécifier les interfaces. Pour chacune d'entre elles nous citerons : ses arguments, sa précondition, ses résultats et sa postcondition.

NIVEAU 6

### Module TACHE

Ce module est visible par une interface : trt tâche

Trt tâche

Arguments : -

Précondition : -

Résultats :
- existe (booléen)
- erreur (booléen)
- nom_tâche (chaîne de caractères)

Postcondition :

erreur est VRAI et nom_tâche est le nom d'une tâche qui n'a pas pu être traduite en DSL.

ou

erreur est FAUX et existe est VRAI et nom_tâche est le nom d'une tâche qui a pu être traduite en DSL avec toutes ses relations (durée, priorité, ressources, précondition et postcondition)

ou

erreur est faux et existe est faux et il n'y a plus de tâches à traduire en DSL.

### Module RESSOURCES_ET_QUANTIFICATIONS

Ce module ne possède qu'une interface : trt ress quant

Trt ress quant

Arguments : -

Précondition : -

Résultats : - erreur (booléen)

Postcondition :

erreur est VRAI et les ressources et quantifications n'ont pas pu être traduites en DSL car elles sont erronées ou incomplètes.

ou

erreur est FAUX et les ressources et quantifications ont pu être traduites en DSL.

NIVEAU 5

Module PRECONDITION

Ce module est visible par l'interface : trt precond

Trt precond

Arguments : nom_tâche (chaîne de caractères)

Précondition : nom_tâche est le nom d'une tâche existante

Résultats : - one_pred (booléen)
- erreur ( " )

Postcondition :

erreur est VRAI et il y a des erreurs syntaxiques ou sémantiques dans la précondition de la tâche nom_tâche

ou

erreur est FAUX et la précondition de la tâche nom_tâche a pu être traduite en DSL suivant les principes examinés au chapitre VI, et one_pred est VRAI si la précondition contient un seul prédicat, FAUX sinon.

Module POSTCONDITION

Ce module est visible par son interface : trt post

Trt post

Arguments : -

Précondition : -

Résultats : - erreur (booléen)

Postcondition :

erreur est VRAI et il y a des erreurs syntaxiques ou sémantiques dans la postcondition de la tâche dont la traduction est en cours

ou

erreur est FAUX et la postcondition de la tâche dont le traitement est en cours a pu être traduite en DSL suivant les principes examinés au chapitre VI.

Module NOM_TACHE

Ce module est visible par son interface : trt n tâche

Trt n tâche

Arguments : - n_tâche (chaîne de caractères)
- one_pred (booléen)

Précondition : n_tâche est le nom d'une tâche existante
et
one_pred est VRAI si la précondition de cette tâche contient 1 prédicat, FAUX sinon

Résultats : -

Postcondition :
La traduction de la tâche dont le nom est nom_tâche, en un processus est effectuée, ainsi que la traduction du déclenchement de ce processus par un message si one_pred est VRAI. Si one_pred est faux aucune relation de déclenchement n'est utilisée dans la section processus.

Module REQRESS_DUREE_PRIORITE

Ce module est visible par ses interfaces : trt durée
trt req ress
trt priorité

Trt durée

Arguments : -

Précondition : -

Résultats : -

Postcondition :
La durée de la tâche dont le traitement est en cours est traduite en DSL.

Trt req ress

Arguments : -

Précondition : -

Résultats : -

Postcondition :
Les réquisitions de ressources effectuées par la tâche dont le traitement est en cours sont traduites en DSL.

Trt priorité

Arguments : -

Précondition : -

Résultats : -

Postcondition :
La priorité de la tâche dont le traitement est en cours est traduite en DSL.

NIVEAU 4

Module VALIDE_SYNTAXE

Nous nous limitons uniquement à la spécification et à l'implémentation des interfaces permettant l'analyse du contenu des pré et postcondition.

Nous supposons, dans le cadre de cette analyse, que l'utilisateur a défini des pré et postconditions répondant à la syntaxe présentée dans le chapitre IV.

Ce module est visible par ses interfaces : cherche
s b s
take str

Cherche

Arguments : -condition (pré ou postcondition)
-mot (chaîne de caractères)
-ind (entier)

Précondition :
condition est une chaîne de caractères

contenant le texte d'une pré ou postcondition
et
mot est une chaîne de caractères de taille plus petite que condition
et
ind est un indice dont la valeur est plus grande ou égale à zéro, et inférieure ou égale au nombre de caractères dans condition

Résultats : - not_exist (booléen)
- ind (entier)

Postcondition :
not_exist est VRAI et mot n'est pas inclus dans condition à partir du ind'ème caractère jusqu'à la fin
ou
not_exist est FAUX, et mot est inclus dans condition, et ind est l'indice du caractère suivant la chaîne de caractères mot.

S b s

Arguments : - condition (pré ou postcondition)
- ind (entier)

Précondition :
condition est une chaîne de caractères contenant le texte d'une pré ou postcondition
et
ind est un indice dont la valeur est plus grande ou égale à zéro, et inférieure ou égale au nombre de caractères dans condition

Résultats : - fin (booléen)
- ind (entier)

Postcondition :
fin est VRAI et on est à la fin de condition
ou
fin est FAUX, et on a passé tous les caractères blancs (fin de ligne, tabulateur, blanc) à partir du ind'ème caractère de condition et ind est maintenant l'indice du caractère suivant la suite de blancs.

Take str

Arguments : - condition (pré ou postcondition)
- place (entier)
- stop (chaine de caractères)

Précondition :
condition est une chaîne de caractère

contenant le texte d'une pré ou postcondition
et
place est un indice dont la valeur est plus grande ou égale à zéro, et inférieure ou égale au nombre de caractères dans condition
et
stop est une chaîne de caractères dont la longueur est inférieure ou égale à condition

Résultats : - place (entier)
- str (chaîne de caractères)
- début (entier)
- fin (entier)

Postcondition :
à partir de place (passé en argument) dans condition, str contient la chaîne de caractères constituée jusqu'aux caractères d'arrêt contenus dans stop; place est l'indice dans condition du premier caractère non-blanc suivant str.

Module VALIDE_SEMANTIQUE
---

Ce module assure la validation sémantique des informations nécessaires à la simulation. Nous ne le spécifions pas davantage. Nous supposons donc que les objets manipulés par l'utilisateur sont complètement et correctement définis dans le MIB.

S'il était spécifié, VALIDE_SEMANTIQUE devraient contenir des interfaces permettant aux modules supérieurs de vérifier la sémantique des informations manipulées dans les spécifications MIB. Ces interfaces auraient pour fonction de vérifier si tel ou tel objet existe bien dans la modélisation de l'utilisateur, ou si telle propriété a été définie, etc... .

NIVEAU 4

Module ACCES-MIB
---

Ce module est visible par ses interfaces : g tâche
g t durée
g t pré
g t post
g t ressource
g t priorité
init mib
close mib

G tâche

Arguments :

Précondition :

Résultats : - not_exist (booléen)
- nom_tâche (chaîne de caractères)

Postcondition :
not_exist est VRAI et il n'y a plus de tâches dans la BD de l'utilisateur
ou
not_exist est FAUX et nom_tâche est le nom d'une tâche

G t durée

Arguments :

Précondition :

Résultats : -not_exist (booléen)
-durée_tâche (chaîne de caractères)

Postcondition :
not_exist est VRAI et la tâche courante n'a pas de durée spécifiée
ou
not_exist est FAUX et durée_tâche est la durée de la tâche courante

G t pré

Arguments :

Précondition :

Résultats : - not_exist (booléen)
- précond (chaîne de caractères)

Postcondition :
not_exist est VRAI et la tâche courante n'a pas de précondition
ou
not_exist est FAUX et précond est la précondition de la tâche courante.

G t post

Arguments :

Précondition :

Résultats : - not_exist (booléen)
- postcond (chaîne de caractères)

Postcondition :
not_exist est VRAI et la tâche courante n'a pas de postcondition
ou
not_exist est FAUX et postcond est la postcondition de la tâche courante.

G t ressource

Arguments :

Précondition :

Résultats : - not_exist (booléen)
- nom_ressource (chaîne de caractères)

Postcondition :
not_exist est VRAI et la tâche courante n'a plus de ressources spécifiées
ou
not_exist est FAUX et nom_ressource est le nom d'une ressource requise par la tâche courante.

G t priorité

Arguments :

Précondition :

Résultats : - not_exist (booléen)
- priorité (chaîne de caractères)

Postcondition :
not_exist est VRAI et la tâche courante n'a pas de priorité spécifiée
ou
not_exist est FAUX et priorité est la priorité de la tâche courante.

Init mib

Arguments :

Précondition :

Résultats :

Postcondition :
Les informations nécessaires à l'exploitation des spécifications MIB, sont initialisées.

Close mib

Arguments :

Précondition :

Résultats :

Postcondition :
Les spécifications MIB sont cloturées.

Module CORRESPONDANCE
---

Ce module est visible par ses interfaces : init corresp
g pre
g synchro
g message
g obj état
p pre
p objet état
show tables
faisabilité
save tables
get tables

Init corresp

Arguments :

Précondition :

Résultats :

Postcondition :
Initialise les informations nécessaires à la gestion des correspondances entre DSL et MIBL.

G pré

Arguments : - nom_point_synchro (chaîne de caractères)

Précondition :

Résultats : - existe (booléen)
- nom_tâche (chaîne de caractères)

Postcondition :
existe est faux et le point-de-synchronisation dont on donne le nom en argument ne correspond à la précondition d'aucune tâche

ou

existe est vrai et nom_tâche est le nom de la tâche dont la précondition a été traduite en un point-de-synchronisation dont le nom a été communiqué en argument.

G synchro

Arguments : - nom_tâche (chaîne de caractères)

Précondition :

Résultats : - existe (booléen)
- nom_point_synchro (chaîne de caractères)

Postcondition :

existe est faux et la précondition de la tâche dont on donne le nom ne correspond à aucun point-de-synchronisation existant

ou

existe est vrai et nom_point_synchro est le nom du point-de-synchronisation qui est la traduction de la précondition de la tâche nom_tâche.

G message

Arguments : - nom_type_obj (chaines de caractères)
- nom_obj ( " )
- etat_obj ( " )

Précondition :

Résultats : - existe (booléen)
- nom_message (chaine de caractères)

Postcondition :

existe est faux et il n'y a aucun message qui soit la traduction de l'objet dont le type est nom_type_obj, le nom nom_obj et l'état état_obj

ou

existe est vrai et nom_message est le nom du message DSL utilisé pour traduire l'objet de type nom_type_obj, de nom nom_obj et d'état état_obj.

G obj état

Arguments : - nom_message

Précondition :

Résultats : - existe (booléen)
- nom_type_obj (chaines de caractères)
- nom_obj ( " )
- etat_obj ( " )

Postcondition :
existe est faux et il n'y a aucun objet dont le type est nom_type_obj, le nom nom_obj et l'état état_obj qui soit la traduction du message DSL nom_message
ou
existe est vrai et l'objet de type nom_type_obj, de nom nom_obj et d'état état_obj est la traduction du message DSL de nom nom_message

P_pre

Arguments : - n_tâche (chaîne de caractères)

Précondition :

Résultats : - erreur (booléen)

Postcondition :
erreur est vrai et il n'y a pas eu moyen d'enregistrer la précondition de la tâche n_tâche dans la table des correspondances
ou
erreur est faux et la précondition de la tâche nom_tâche est traduite en un point_de_synchronisation dans la table des correspondances.

P_objet_état

Arguments : - pré (booléen)
- nom_type_obj (chaines de caractères)
- nom_obj ( " )
- etat_obj ( " )

Précondition :

Résultats : - erreur (booléen)

Postcondition :
erreur est vrai, et, l'objet dont le type est nom_type_obj, le nom nom_obj et l'état état_obj n'a pas pu être enregistré dans la table des correspondances
ou
erreur est faux, et, l'objet dont le type est nom_type_obj, le nom nom_obj et l'état état_obj est enregistré dans la table des correspondances

Show tables

Arguments :

Précondition :

Résultats :

Postcondition :
Les tables de correspondances ont été affichées à l'écran.

Faisabilité

Arguments :

Précondition :

Résultats : - faisable (booléen)

Postcondition :
faisable est vrai et il n'y a aucun objet qui soit dans un état qu'il ne puisse atteindre
ou
faisable est faux et les objets affichés ne peuvent atteindre l'état qui est affiché à leur droite.

Save tables

Arguments :

Précondition :

Résultats :

Postcondition :
Les tables de correspondances ont été sauvées dans un fichier, pour permettre au programme d'analyse des résultats de la simulation de les consulter.

Get tables

Arguments :

Précondition :

Résultats :

Postcondition :
Les tables de correspondances sont restituées.

## Module GENERE_DSL

Toutes les interfaces ci-dessous ne nécessitent aucune spécification détaillée supplémentaire étant donné la simplicité du traitement qu'elles effectuent. En effet, nous avons subdivisé la génération du langage en un ensemble de "petites boîtes" dont chacune connaît la syntaxe des concepts DSL qu'elle doit manipuler.
Nous nous contentons donc pour la plupart d'en donner une brève description.

Ce module est visible par les interfaces suivantes :

init dsl
close dsl
w conditions
w processus
w priorite
w req ress
w gen mess
w duree proc
w est declenche
w message
w survenance
w interface
w attribut
w est contribue
w est realise
w et
w ou
w gen
w compter
w memorisation
w memorise
w decl nombre
w decl for
w pt synchro
w pt virgule
w ligne blanche
w ress quant

### Init dsl

Arguments :

Précondition :

Résultats :

Postcondition :
Les informations nécessaires à la traduction DSL sont initialisées.

Close dsl

Arguments :

Précondition :

Résultats :

Postcondition :
Les spécifications DSL générées sont cloturées.

W conditions

Cette interface génère les objets conditions DSL à partir des conditions MIB.

W processus

Cette interface génère sur base du nom de processus l'entête de section du processus.

W priorite

Cette interface sur base d'une priorité génère une relation de priorité.

W req ress

Cette interface sur base d'un taux de réquisition et d'un nom de ressource, génère une relation de réquisition par un processus.

W gen mess

Cette interface sur base d'un nombre de messages, d'un nom de condition et d'un nom de message, génère une relation de génération de messages par un processus.

W duree proc

Cette interface sur base de la durée d'un processus, génère une relation de durée.

W est declenche

Cette interface sur base d'un nom de message, génère une relation de déclenchement dans sa forme passive, dans une section processus.

W message

Cette interface sur base d'un nom de message génère l'entête de section d'un message.

W survenance

Cette interface sur base d'une relation de survenance, génère celle-ci en DSL.

W interface

Cette interface sur base d'un nom d'interface génère l'entête de section d'une interface.

W attribut

Cette interface sur base d'un nom d'attribut, de savaleur et d'un nom de point-de-synchronisation, génère la définition d'un attribut en DSL.

W est contribue

Cette interface sur base d'un nom de message ou de point de synchronisation ainsi que du nombre de contributions, génère une relation de contribution à la forme passive dans une section point-de-synchronisation.

W est realise

Cette interface génère l'entête de la condition de synchronisation, dans une section point-de-synchronisation.

W et

Cette interface génère un ET.

W ou

Cette interface génère un OU.

W gen

Cette interface sur base d'un nom de message génère une opérande de type génération de message dans une condition de synchronisation.

W compter

Cette interface sur base d'un nom de message ou de point_de synchronisation, et d'un nom d'attribut, génère l'opérateur compter ainsi que son opérande, dans une condition de synchronisation.

W memorisation

Cette interface génère l'entête des relations de mémorisation.

W memorise

Cette interface sur base d'un nom de message ou de point-de-synchronisation, et d'une durée, génère une relation de mémorisation.

W decl nombre

Cette interface sur base d'une valeur et d'un nom de processus génère un relation de déclenchement dans les sections point-de-synchronisation

W decl for

Cette interface sur base d'un nom d'attribut et d'un nom de processus génère un relation de déclenchement multiple dans les sections point-de-synchronisation

W pt synchro

Cette interface sur base d'un nom de point-de-synchronisation, génère l'entête d'une section point-de-synchronisation.

W pt virgule

Cette interface génère un point-virgule(;).

W ligne blanche

Cette interface génère une ligne vide.

W ress quant

Cette interface génère en DSL, après les avoir traduites si nécessaire, les ressources et quantifications du MIB

NIVEAU 2

Module COORDINATEUR

Le module coordinateur n'est pas à proprement parler d'interface, car il n'est visible pour aucun module, il s'agit en fait d'un "programme principal" qui à l'implémentation se chargera d'appeler les procédures qui seront l'implémentation des interfaces décrites lors de la spécification. l'agencement des appels assurés par ce module aura pour but de réaliser la traduction d'une base de données MIB en une spécification DSL.

## 7.5. Conclusion

Nous avons présenté dans ce chapitre l'environnement logiciel dans lequel s'inscrit notre programme de traduction. Ensuite, après avoir examiné les fonctionnalités de ce programme, nous en avons conçu l'architecture logique.

Cette architecture a servi de base à l'implémentation que le lecteur peut retrouver en annexe. Cette implémentation nous permet maintenant sur base d'une spécification du fonctionnement et des ressources, exprimée en MIBL, de traduire cette spécification en DSL.

Le résultat de notre traduction peut servir d'entrée à DSL-SIM, qui simulera la spécification DSL et produira des résultats statistiques.

Cependant, ces résultats sont exprimés en terme d'objets DSL et doivent donc encore être "retraduits" en MIBL afin qu'ils puissent être interprétés par l'utilisateur de notre système.

# CHAPITRE VIII

# CONCLUSION

## 8.1. Synthèse

Ce travail avait pour but de permettre à un concepteur d'applications bureautiques de modéliser le fonctionnement d'un bureau au moyen d'un langage (MIBL) dont la syntaxe serait simple et la sémantique aussi proche que possible des concepts bureautiques.

Etant donné l'approche centrée sur les objets que nous avons choisie pour décrire le fonctionnement ( cfr. chapitre II ), une modélisation en MIBL ne permet pas de connaître le scénario d'exécution de l'activité globale. Il était donc nécessaire de mettre à la disposition de l'utilisateur un outil lui permettant d'évaluer une modélisation effectuée au moyen de notre langage.

Afin de pouvoir évaluer les spécifications exprimées en MIBL, nous avons choisi d'utiliser le simulateur DSL-SIM existant dans l'environnement logiciel IDA. Ce programme de simulation travaille sur des spécifications dynamiques (basées sur les notions d'événement et de processus) rédigées en DSL.

Puisque les principes fondamentaux de ce langage d'expression du fonctionnement sont différents de ceux adoptés en MIBL, il s'est avéré nécessaire de définir et d'implémenter un processus automatique de traduction de MIBL, centré objets, en DSL, centré traitements (cfr. chapitre VI, VII, et Annexes).

Cependant, pour utiliser le simulateur DSL-SIM, nous avons dû compléter notre langage afin d'exprimer certaines propriétés "dynamiques", ainsi que certaines quantifications nécessaires à l'exécution de la simulation. Cette nécessité d'effectuer une traduction nous a également obligé à "handicaper" MIBL, en restreignant notamment la puissance d'expression des postconditions, car nous n'avons pas trouvé de sémantique en DSL permettant d'exprimer tout ce que nous désirions.

Notre langage de spécification du fonctionnement aurait peut-être été plus approprié et plus simple si, en le définissant, nous ne nous étions pas tant souciés de sa traduction en DSL.
Cependant, nous estimons qu'il peut s'adapter à la description de nombreux cas de SIB, comme en témoigne l'exemple qui se trouve en annexe; de plus, cet exemple simple n'utilise qu'une partie des constructions syntaxiques offertes par MIBL.

## 8.2. Perspectives

Notre travail voulait avant tout poser les bases d'un langage d'expression du fonctionnement des SIB. Ce n'est pas un langage opérationnel, et son pouvoir d'expression a été limité par l'utilisation du simulateur.

En guise de perspectives, nous préconisons dans un premier temps d'implémenter les modules d'analyse syntaxique et sémantique qui permettront de vérifier si le modèle des données, le modèle des ressources et les quantifications sont complets et cohérents dans le cadre de la simulation.
Ensuite, il serait intéressant de raffiner le langage en le testant auprès d'analystes spécialisés dans ce genre de problème.

Cette étape permettra de compléter sa syntaxe et sa sémantique, de vérifier la pertinence des résultats fournis par le simulateur, et de déterminer avec précision les résultats attendus d'une simulation.

Une fois cette analyse effectuée, viendront tout d'abord, la conception et l'implémentation d'un logiciel opérationnel de validation des informations nécesaires à la simulation, et ensuite d'un simulateur dédicacé au langage MIBL.

# BIBLIOGRAPHIE


[BOCHMAN, 83] G. von BOCHMANN
Concepts for distributed systems design 1983

[BOD_PIG, 83] F. BODART, Y. PIGNEUR.
Conception appliquée des applications informatiques:
1. Etude d'opportunité et analyse conceptuelle.
MASSON Presses Universitaires de NAMUR 1983

[BRO_SIL, 82] M.L. BRODIE, E. SILVA
Active and passive components modelling : ACM/PCM
IFIP 1982

[DACHOUF, 86] N. DACHOUFFE
MIB : un modèle d'information de bureau.
FNDP NAMUR 1986

[DIJKSTR, 76] E.W. DIJKSTRA
A discipline of programming.
Prentice-Hall, series in automatic computation 1976

[DSL_SPE, 83] Manuel de référence DSL SPEC
FNDP NAMUR 1983

[DSL_SIM, 83] Manuel de référence DSL SIM
FNDP NAMUR 1983

[ELLIS, 80] C.L. ELLIS
Information Control Nets : a mathematical model
of office information system flow.
Conference on simulation, measurement and modelling
of computer systems 1980

[LEAVITT, 65] H.J. LEAVITT
Applying organizational change in industry :
structural, technological, and humanistic
approaches. A handbook of organizations.
RAND MC NALLY, Chicago 1965

[GOL-ROB, 83] A. GOLDBERG, D.ROBSON
SMALLTALK 80 : the language and his implementation
A d d i s o n - W e s l e y 1 9 8 3

[PIGNEUR, 84] Y. PIGNEUR
Spécification et évaluation du comportement des SI.
Thèse de doctorat FNDP NAMUR 1984

[ROLLAND, 82] C. ROLLAND, C.RICHARD
The REMORA methodology for IS design and management.
IS design methodologies, a comparative review
I F I P 1 9 8 2

[TSICHRI, ??] O. NIERSTRASZ, D. TSICHRITZIS
Office Object Flow.
Technical report CSRG 150
Edité par D. TSICHRITZIS
Dpt of Computer science, University of Toronto

[VANLAM1, 86] A. van LAMBSWEERDE
Notes du cours de méthodologie de développement.
FNDP NAMUR 1986

[VANPEVE, 87] M. SIMOENS, O. VAN PEVENAEYGE
Animation graphique des tâches de bureau.
FNDP NAMUR 1987