THESE

présentée par

Luis Otavio Campos ALVARES

pour obtenir le titre de **DOCTEUR**

de l'**UNIVERSITE JOSEPH FOURIER - GRENOBLE I**

(arrêté ministériel du 5 juillet 1984)

Spécialité: INFORMATIQUE

CONTRIBUTION A L'ETUDE DU PILOTAGE DE LA MODELISATION DES SYSTEMES D'INFORMATION

Date de soutenance: 17 octobre 1988

Composition du jury:

Président	Y. Chiaramella
Rapporteurs	F. Bodart
	J. Kouloumdjian
Examinateurs	M. Adiba
	J-P. Giraudin
	M. Léonard
	F. Peccoud

Thèse préparée au sein du Laboratoire de Génie Informatique - IMAG
Université Joseph Fourier - Grenoble I

À Mariza, ao Lucas e à Ticiana

Je tiens à remercier

Monsieur Yves Chiaramella, Professeur à l'Université Joseph Fourier de Grenoble, de me faire l'honneur de présider le jury de cette thèse.

Monsieur Michel Adiba, Professeur à l'Université Joseph Fourier de Grenoble et Directeur de cette thèse, pour m'avoir acueilli au sein de son équipe de recherche et en particulier pour m'avoir donné les conditions matérielles pour la bonne réalisation de cette thèse.

Monsieur Jacques Kouloumdjian, Professeur à l'Institut National des Sciences Appliquées de Lyon, d'avoir bien voulu juger mon travail et participer à ce jury.

Monsieur François Bodart, Professeur aux Facultés Universitaires Notre Dame de la Paix, Namur, Belgique, d'avoir bien voulu apporter son jugement sur ce travail et participer à ce jury. L'intérêt qu'il a porté à cette recherche, ses suggestions permanentes et ses critiques, m'ont été d'une grande aide.

Monsieur Michel Léonard, Professeur à l'Université de Genève et Monsieur François Peccoud, Professeur à l'Université des Sciences Sociales de Grenoble, pour m'avoir fait l'honneur et le plaisir de participer au jury de cette thèse.

Monsieur Jean-Pierre Giraudin, Maître de Conférences à l'Université des Sciences Sociales de Grenoble, qui est à l'origine de cette thèse, pour m'avoir conduit tout au long de ce travail. Sa totale disponibilité, son expérience, ses nombreux conseils, son amitié et surtout sa grande patience pour corriger le manuscrit de cette thèse, représentent sans nul doute, un facteur décisif dans l'aboutissement de ce travail. Je lui en suis tout particulièrement reconnaissant.

Monsieur Claude Del Vigna, Ingénieur du Centre National de la Recherche Scientifique, pour l'encadrement efficace d'un projet d'étudiant de réalisation d'un module expérimental.

Madame Monique Chabre-Peccoud, Maître de Conférences à l'Université Joseph Fourier de Grenoble, pour avoir accepté d'assurer la relecture de cette thèse.

L'Universidade Federal do Rio Grande do Sul, Brésil, et la CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pour leur soutien financier.

Le Service de Reprographie de l'IMAG, qui a assuré le tirage de ce document.

Je tiens tout particulièrement à remercier Mariza, ma femme, pour avoir accepté de rester ces années loin de notre famille et de ses amis, pour son soutien moral, sa patience, sa compréhension et son dévouement.

J'embrasse Lucas et Ticiana dont le sourire et la fraîcheur nous illuminent.

Que tous les autres membres de ma famille soient remerciés pour leur encouragement et leur soutien.

Enfin, je ne saurais oublier tous mes amis qui ont écouté mes complaintes, mes joies, mes peurs depuis le début de cette galère: Claudio, Ligia, Guilherme, Valdeni, Vera, Thadeu, Benoît, Norberto, Fabio, Norma,...

RESUME

Ce travail se situe dans le cadre des outils de modélisation des systèmes d'information (SI) en informatique de gestion. La complexité croissante des logiciels d'application oblige en effet à créer de nouveaux outils de développement.

La réalisation d'outils de pilotage de la modélisation est rendue difficile à cause des définitions imprécises des méthodes de modélisation utilisées. Notre étude consiste à proposer une représentation formelle complète de ces méthodes dans le but de constituer une base de connaissances d'un système expert pour diriger des modélisations.

Une architecture fonctionnelle d'un tel système de pilotage est proposée et expérimentée. Cette architecture intègre un sous-système de configuration pour permettre la définition précise d'une méthode personnalisée. La description d'une méthode par des regroupements d'expressions formelles utilisées pour vérifier la conformité des spécifications vis à vis de cette méthode assure la cohérence entre la définition et la vérification.

Mots clés: informatique de gestion, système d'information, pilotage de modélisation, conception assistée, règles de cohérence et de complétude, base de connaissances.

TABLE DES MATIERES

ANNEXE B - LE PROTOTYPE DEVELOPPE

LISTE DES ILLUSTRATIONS

PREAMBULE

La forte évolution technologique des ordinateurs (augmentation des capacités de stockage et des performances, etc...), complétée par une diminution des coûts, conduit à mettre en œuvre sur ces machines des systèmes de plus en plus complexes.

Cette complexité associée aux caractéristiques du processus de modélisation conduit aux constats suivants:

• La conception des systèmes est plus un art qu'une science. Imagination, objectivité, expérience et bon jugement sont les qualités essentielles d'un concepteur [WIL65].

• Le résultat de la modélisation continue à reposer essentiellement sur le savoir-faire de l'analyste-concepteur, sur sa maîtrise du modèle et sur son expérience de la conception [PRO86].

• La qualité de la modélisation obtenue est très dépendante de l'expérience et de la perspicacité de l'analyste-concepteur. Ce processus est caractérisé par une certaine indétermination dans la façon de choisir les structures [BOU86a].

Ce type de constatation montre bien que le processus de modélisation des systèmes d'information (SI) est un domaine peu connu, peu structuré et peu formalisé, donc difficile à traiter. Il faut encore beaucoup d'efforts de recherche pour arriver à bien le comprendre et de ce fait pouvoir mieux contrôler cette activité de modélisation.

Aujourd'hui plusieurs méthodes de modélisation des SI sont proposées. Mais, qu'est-ce qu'une méthode de modélisation?

• Définition de "méthode" dans le Robert édition 1987:

"...

- ensemble de démarches raisonnées, suivies, pour parvenir à un but.

- ensemble des règles, des principes normatifs sur lesquels reposent l'enseignement, la pratique d'un art.

..."

• Définition de "méthode" dans le petit Larousse édition 1986:

"...

- Manière de dire, de faire, d'enseigner une chose, suivant certains principes et avec un certain ordre.

- Démarche ordonnée, raisonnée; technique employée pour obtenir un résultat.

..."

Ces définitions sont tout à fait satisfaisantes. Cependant nous pouvons les compléter en considérant plusieurs points de vue, propres au domaine de la modélisation des SI:

• un **point de vue technique**: Une méthode est une sorte d'"amalgame" entre modèles, langages, démarches et outils.

• un **point de vue évolutif**: Une méthode suppose une activité essentiellement dynamique qui doit s'adapter en permanence à l'environnement.

• un **point de vue humain**: Dans le cadre d'un travail d'équipe, une méthode doit permettre une certaine "personnalisation" de l'activité de chacun par rapport à la méthode globalement définie pour toute l'équipe.

Par rapport au premier point de vue de nombreuses questions se posent.

Qu'est-ce qu'une "bonne" méthode? Est-ce que c'est décrire de "bons" modèles avec un "bon" langage et selon une "bonne" démarche? Qu'est-ce qu'un "bon" modèle, un "bon" langage, une "bonne" démarche? Ces questions restent assez ouvertes et n'ont que des réponses partielles.

En particulier, les nouvaux outils de modélisation sont centrés sur deux préoccupations principales: d'une part, **amélioration de la convivialité** à l'aide d'interfaces graphiques et d'interfaces en langue naturelle et d'autre part, **augmentation de la cohérence** par une meilleure formalisation.

Nos propositions se situent selon ce deuxième axe. D'une manière générale nous pouvons dire que notre objectif est de passer d'une notion de cohérence **"immergée"** dans les outils logiciels actuels à une **spécification** explicite et rigoureuse de cette cohérence. Nous étudions aussi la notion de complétude d'une modélisation.

Ainsi, cette thèse s'inscrit dans le cadre d'une recherche sur le processus de modélisation des SI et aborde plus spécifiquement l'étude du pilotage d'une modélisation.

Chaque chapitre proposé traite d'un aspect particulier de ce domaine.

Le **chapitre I** concerne la notion de modélisation des SI. Initialement sont présentées des définitions de SI et quelques méthodes de modélisation, représentatives du domaine, sont analysées succinctement selon les quatre composantes: modèles, langages, démarche et outils. La deuxième partie de ce chapitre introduit l'approche système expert dans le cadre de la modélisation des SI, en considérant les avantages escomptés et en analysant les connaissances à prendre en compte.

Une modélisation rigoureuse de ces connaissances est proposée dans le **chapitre II** qui définit un formalisme précis de représentation des différentes composantes d'une méthode de modélisation des SI. Ce chapitre II est structuré en quatre sections: nous présentons d'abord une caractérisation informelle d'une méthode de modélisation, nous introduisons ensuite le formalisme retenu, puis nous proposons une technique de définition formelle d'une méthode et enfin nous définissons les notions de cohérence et de complétude d'une spécification.

La définition formelle d'une méthode est basée sur une représentation des langages de spécification de SI et sur des règles de vérification et de validation.

Le **chapitre III** est centré sur une étude systématique de ces règles de contrôle de spécifications. En particulier, nous proposons une typologie de ces règles basée sur les énoncés. Pour chaque type de règle un exemple élémentaire est proposé et un exemple complet d'un ensemble de règles pour une méthode particulière est décrit. Nous terminons par une analyse du

problème de l'inter-dépendance entre de telles règles.

Après une définition formelle d'une méthode de modélisation et l'étude des règles associées, nous pouvons considérer le pilotage proprement dit.

Dans le **chapitre IV** nous proposons d'abord une architecture fonctionnelle pour le pilotage de la modélisation des SI. Cette architecture, basée sur la représentation des méthodes retenue au chapitre II intègre un sous-système de configuration pour permettre la définition d'une méthode spécifique en fonction de l'application à modéliser et du groupe d'analystes-concepteurs concerné. Ensuite, nous présentons une solution d'implémentation dans le cadre d'un environnement Prolog.

Un bilan de ce travail de recherche en termes d'évaluation critique et de perpectives est développé au **chapitre V**.

Une première annexe concerne une description de la méthode IDA choisie pour illustrer nos propositions.

Une deuxième annexe présente certaines parties significatives du prototype développé.

Ce travail a été fortement influencé par nos 10 années d'expérience d'analyste-concepteur et de chef de projets au Centre de Calcul de l'Université Fédérale du Rio Grande do Sul (Brésil) ce qui peut paraître paradoxal par rapport au contenu de cette thèse. En effet, cette expérience pratique nous a conduits vers cette nouvelle voie d'étude préparant une nouvelle génération d'outils logiciels. Notre recherche n'est pas purement académique mais a la préoccupation fondamentale de proposer un instrument de génie logiciel qui puisse être réellement employé par les informaticiens dans un futur assez proche.

CHAPITRE I

MODELISATION DES SYSTEMES D'INFORMATION

MODELISATION DES SYSTEMES D'INFORMATION

I.1 Introduction

Ce premier chapitre a pour but, d'une part, de bien cerner la notion de modélisation des systèmes d'information et, d'autre part, d'introduire l'approche système expert pour aider l'analyste-concepteur à mieux réaliser son activité.

La première partie concerne des définitions sur les SI, la caractérisation des étapes du cycle de vie et les composantes des méthodes de modélisation pour fixer le vocabulaire utilisé. Quelques méthodes représentatives du domaine sont présentées succinctement.

La deuxième partie inclut une brève caractérisation des systèmes experts, les avantages escomptés par l'utilisation de cette approche dans un outil de pilotage de la modélisation de SI et une classification des connaissances mises en œuvre pour accomplir cette modélisation.

I.2 Les SI et leur modélisation

L'information est considérée aujourd'hui comme une des ressources très importante d'une organisation et son poids relatif a tendance à augmenter. Les informations mises à la disposition de chacun dans le cadre de ses activités sont souvent indispensables.

Une représentation schématique des organisations est montrée dans la figure I.1, empruntée à Le Moigne [LEM77], qui présente l'organisation composée par trois systèmes: système opérant, système d'information et système de décision. Le système d'information est couplé au système opérant et au système de décision.

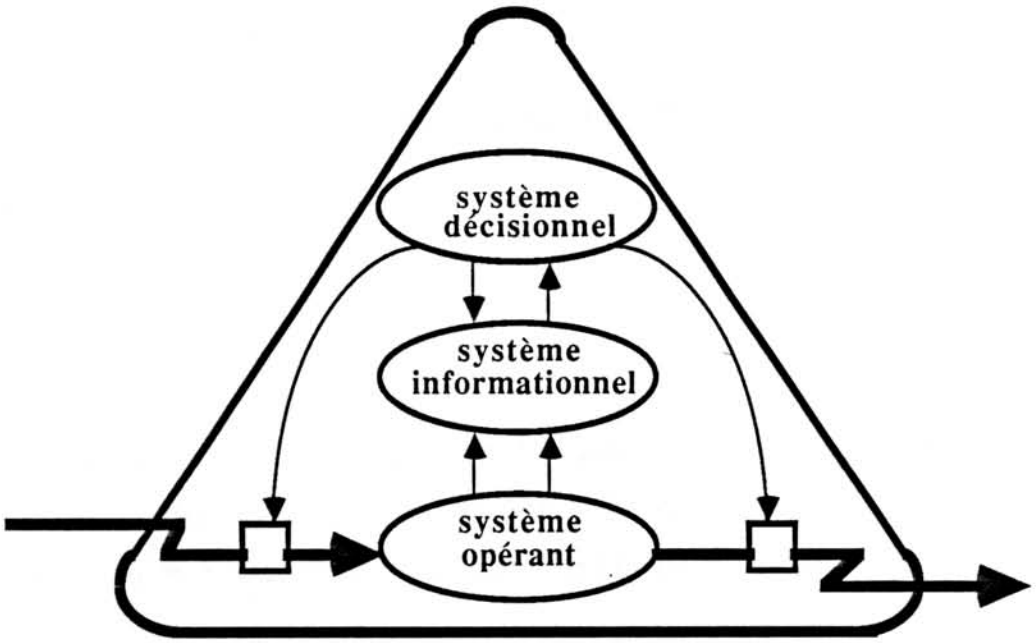


Figure I.1: Représentation schématique des organisations

Il n'y a pas de concept définitif et simple de la notion de système d'information d'une organisation. J. Gigch et L. Pipino [GIG86], par exemple, caractérisent un système d'information comme étant

> "a collection of MULTILEVEL and RECURSIVELY related subsystems where at least one PERSON of a certain PSYCHOLOGICAL TYPE within some ORGANIZATIONAL CONTEXT faces a problem of a given class for witch EVIDENCE, RATIONALITY and LOGIC are needed to arrive at a solution (that is, to select some course of action) and that the EVIDENCE is made available through some MODE OF PRESENTATION".

De cette définition nous mettons en évidence le fait de considérer un système d'information comme une collection de sous-systèmes de plusieurs niveaux.

Dans une notion moins théorique, Jacques Melèse [MEL79] propose un approche qui considère l'organisation comme un système socio-technique

complexe et examine sa capacité à se comporter comme un système informationnel adapté à l'organisation interne et aux relations organisation-environnement. Le concept de système d'information désigne alors

> "l'ensemble interactif de toutes les situations informationnelles, autrement dit, le jeu complexe de tous les échanges d'information signifiante".

Le groupe GALACSI [GAL84] définit le système d'information d'une organisation sociale comme étant

> "l'ensemble des moyens, humains et matériels, et des méthodes se rapportant au traitement des différentes formes d'information rencontrées dans les organisations".

Une définition plus détaillée est donnée par C. Rolland [ROL88] en considérant un système d'information d'une organisation comme un ensemble formé:

> "- de collections de données, représentations partielles, en partie arbitraires mais nécessairement opératoires, d'aspects pertinents de la réalité de l'organisation sur lesquels on souhaite être renseigné. Ces collections inter-reliées, aussi cohérentes que possible, sont mémorisées et communiquées dans le lieu, le moment et la présentation appropriés aux acteurs qui en ont l'usage,
> - de collections de règles qui fixent le fonctionnement informationnel. Ces règles traduisent ou sont calquées sur le fonctionnement organisationnel. Partie intégrante du SI, ces règles doivent être connues des acteurs qui utilisent le SI. Elles leur sont nécessaires pour l'interprétation et la manipulation des collections de données,
> - d'un ensemble de procédés pour l'acquisition, la mémorisation, la transformation, la recherche, la communication et la restitution des renseignements,

- d'un ensemble de ressources humaines et de moyens techniques intégrés dans un système, coopérant et contribuant à son fonctionnement et à la poursuite des objectifs qui lui sont assignés."

Le système d'information retenu dans cette étude est le système d'information automatisé qui est la partie du système d'information de l'organisation correspondant au sous-système d'information fonctionnant à l'aide d'un système informatique. A partir de ce point, nous utilisons la dénomination système d'information (SI) pour signifier le système d'information automatisé.

I.2.1 Cycle de vie des systèmes d'information

Sans entrer dans des querelles d'écoles concernant ce qu'il est convenu d'appeler le cycle de vie d'un système d'information, on peut schématiser l'évolution d'un SI selon les étapes traditionnelles suivantes:

- l'**étude d'opportunité**: préparation d'un avant-projet de solution à partir des besoins exprimés par l'organisation;
- l'**analyse conceptuelle**: spécification d'une solution détaillée indépendante de tout moyen de réalisation;
- la **conception technique**: description précise d'une solution qui prend en compte les caractéristiques logiques des moyens de réalisation;
- la **réalisation**: production d'une solution exécutable en fonction des caractéristiques réelles des matériels, des logiciels et de l'organisation;
- l'**utilisation et la maintenance**: utilisation avec correction éventuelle et évolution du système opérationnel.

La figure II.2 présente une vision linéaire du cycle de vie des SI. Cette vision linéaire idéale n'est qu'un cadre de référence, car dans la réalité, de nombreux retours en arrière sont nécessaires et le déroulement est fréquemment itératif.

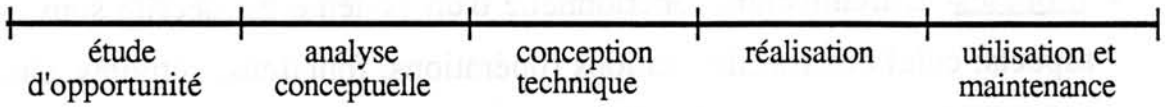


Figure I.2: Vision linéaire du cycle de vie des SI

I.2.2 Méthodes de modélisation des systèmes d'information

La complexité croissante des SI automatisés a pour conséquence un transfert d'efforts de la réalisation vers les étapes initiales d'étude d'opportunité et d'analyse conceptuelle et nécessite une méthode de travail. Ce sont ces premières étapes que nous voulons traiter; les dernières étapes sont plus particulièrement abordées par les techniques classiques de Génie Logiciel.

On peut distinguer quatre composantes essentielles des méthodes de modélisation des SI [ROL86]:

- les **modèles**: ensemble de concepts et de règles relatives à leur utilisation pour fixer le vocabulaire et le type d'abstraction;
- les **langages**: descriptions formelles des images du système d'information selon les modèles retenus;
- les **démarches**: processus opératoires par lesquels s'organise et s'effectue le travail d'analyse, de description et de spécification du SI;
- les **outils:** logiciels de documentation, d'aide à la spécification, d'évaluation, de traduction et de simulation.

Chaque méthode intègre, à des degrés divers, ces quatre dimensions. Nous présentons ci-dessous quelques méthodes, représentatives du domaine de conception de SI, analysées succinctement selon ces quatre composantes. Nous ne considérons que les étapes initiales d'étude d'opportunité et d'analyse conceptuelle définies ci-dessus.

SADT [ROS77, LIS86, MAR87]

- modèles- L'architecture fonctionnelle d'un système est décrite sous deux aspects: celui des transformations (opérations, fonctions, activités, tâches, traitements, ...) et celui des éléments (objets, données, informations, ...). Un modèle SADT est une suite cohérente et hiérarchisée de diagrammes. Le diagramme de plus haut niveau représente l'ensemble du problème. Chaque diagramme de niveau inférieur ne révèle qu'une quantité limitée de détails sur un sujet parfaitement délimité. Deux modèles sont proposés, l'un mettant l'accent sur les transformations et l'autre mettant l'accent sur les objets, appelés respectivement **actigramme** et **datagramme**. Les actigrammes portent notamment sur la décomposition d'une activité en activités de plus en plus élémentaires et sur la définition des entrées, des sorties, des contrôles et des mécanismes utilisés par chaque activité. Les datagrammes décrivent la décomposition des données en indiquant les activités utilisatrices, les activités génératrices, les activités de contrôle et la mémorisation.
- langages- La méthode SADT utilise un langage graphique pour construire les actigrammes et les datagrammes formé par seulement deux éléments: des rectangles et des flèches. La notation graphique met en évidence les interfaces des sous-systèmes et concrétise les relations entre tous les composants du système (cf. figure I.3). Il existe aussi un langage pour représenter les conditions d'activation des traitements. Des textes en langage naturel sont utilisés tout au long du travail pour compléter les diagrammes.
- démarche- La démarche SADT préconise une modélisation descendante du système basée sur un travail d'équipe discipliné par le cycle auteur-lecteur: chaque diagramme écrit est communiqué à plusieurs autres membres de l'équipe afin d'être revu et commenté. Ces commentaires sont soumis par écrit à l'auteur qui, à son tour, porte ses réactions aux remarques et aux suggestions faites par les lecteurs. Ce cycle de critique et d'approbation se déroule jusqu'à ce que la totalité du modèle soit finalement approuvée. Cette

organisation du travail bien que classique est complétement formalisée dans la méthode SADT.

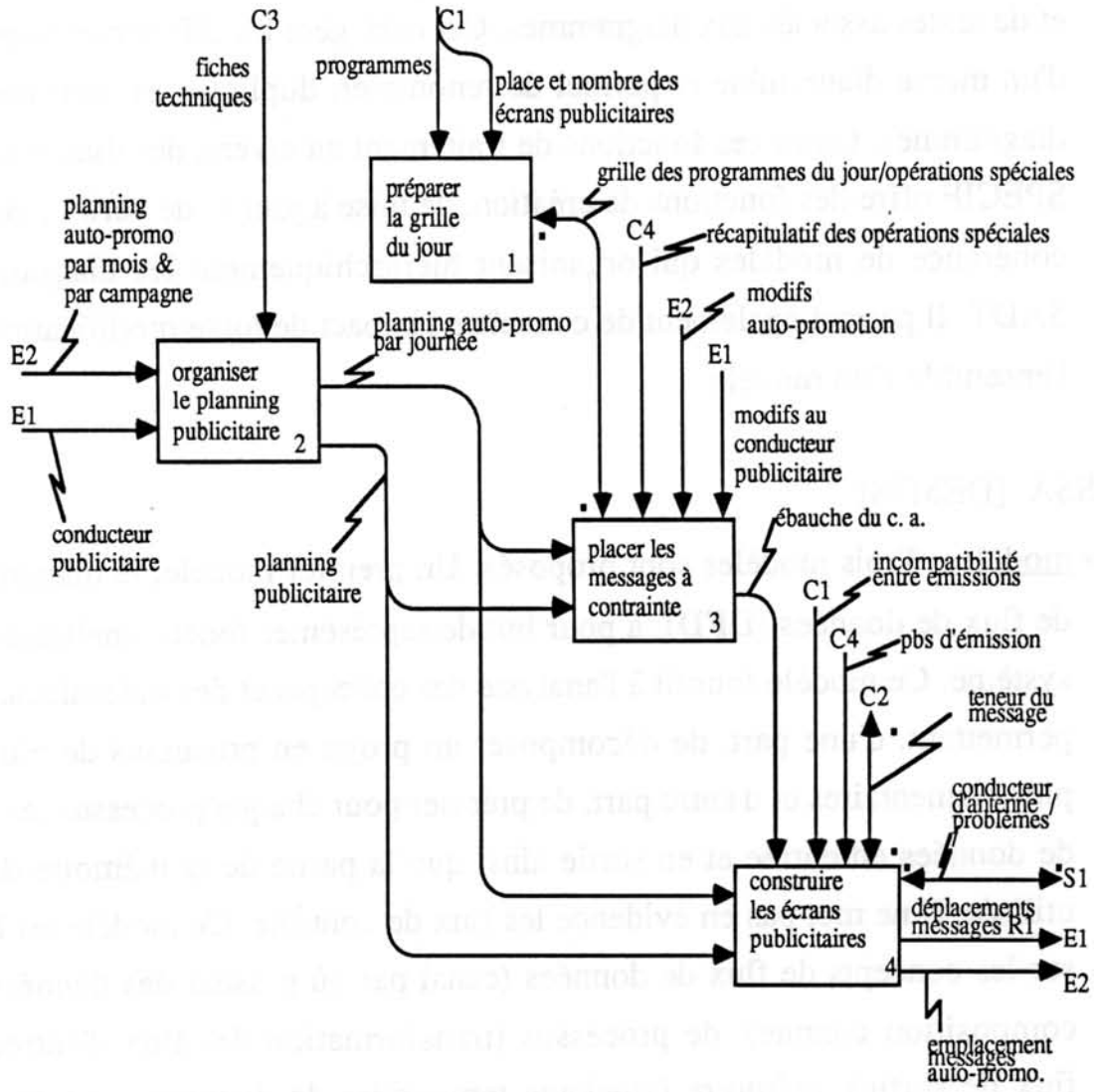


Figure I.3: Exemple d'actigramme sur une gestion publicitaire [LIS86]

- outils- Historiquement la méthode SADT a été développée indépendamment d'outils logigiels. Actuellement quelques outils sont disponibles, comme par exemple SPECIF [RIG86]. On peut décrire l'utilisation de SPECIF sur trois niveaux. Le niveau "production" est essentiellement chargé de saisir, de modifier et d'imprimer les diagrammes; le niveau "organisation" gère les modèles et les projets; le niveau "exploitation" fournit des vérifications et

des documents issus de ces modèles. S'appuyant sur cette organisation, SPECIF offre des fonctions de saisie, de modification et de vérification syntaxique de diagrammes. Il permet également la constitution de glossaires et de textes associés aux diagrammes. Cet outil gère les différentes versions d'un même diagramme et permet de renommer, dupliquer et réutiliser les diagrammes. Outre ces fonctions de traitement au niveau des diagrammes, SPECIF offre des fonctions de création, de mise à jour et de vérification de cohérence de modèles qui organisent hiérarchiquement les diagrammes SADT. Il permet également de connaître l'impact de toute modification sur l'ensemble d'un modèle.

SSA [DEM78]

- modèles: Trois modèles sont proposés. Un premier modèle, le diagramme de flux de données (**DFD**), a pour but de représenter fonctionnellement le système. Ce modèle fournit à l'analyste des concepts et des mécanismes lui permettant, d'une part, de décomposer un projet en processus de plus en plus élémentaires et d'autre part, de préciser pour chaque processus les flux de données en entrée et en sortie ainsi que la partie de la mémoire du SI utilisée. Il ne met pas en évidence les flux de contrôle. Ce modèle est basé sur les concepts de flux de données (canal par où passent des données de composition connue), de processus (transformation des flux d'entrée en flux de sortie), mémoire (stockage temporaire de données) et interface externe (système, personne ou organisation en dehors du système étudié mais qui reçoit ou qui fournit des données au système). Un deuxième modèle, le dictionnaire de données (**DD**) sert à définir en termes de composition de données les flux de données et les entités de la mémoire du SI. Le troisième modèle, le diagramme de structure de données (**DSD**) sert à caractériser la structure de la mémoire du SI en termes d'entités et de liaisons entre ces entités.
- langages: Plusieurs langages sont utilisés. Pour le premier modèle, le DFD, un langage graphique simple est utilisé, avec seulement quatre symboles: le rond pour représenter les processus; la ligne droite pour la mémoire; la

flèche pour les flux de données; le rectangle pour les interfaces externes (cf. figure I.4). Le langage pour le DD utilise les concepts de concaténation, sélection et répétition pour définir rigoureusement chaque composant spécifié dans le DFD. Dans le cadre du DSD est utilisé un langage graphique formé par seulement deux symboles: le rectangle représentant une entité et la flèche reliant deux rectangles signifiant une liaison entre ces entités.

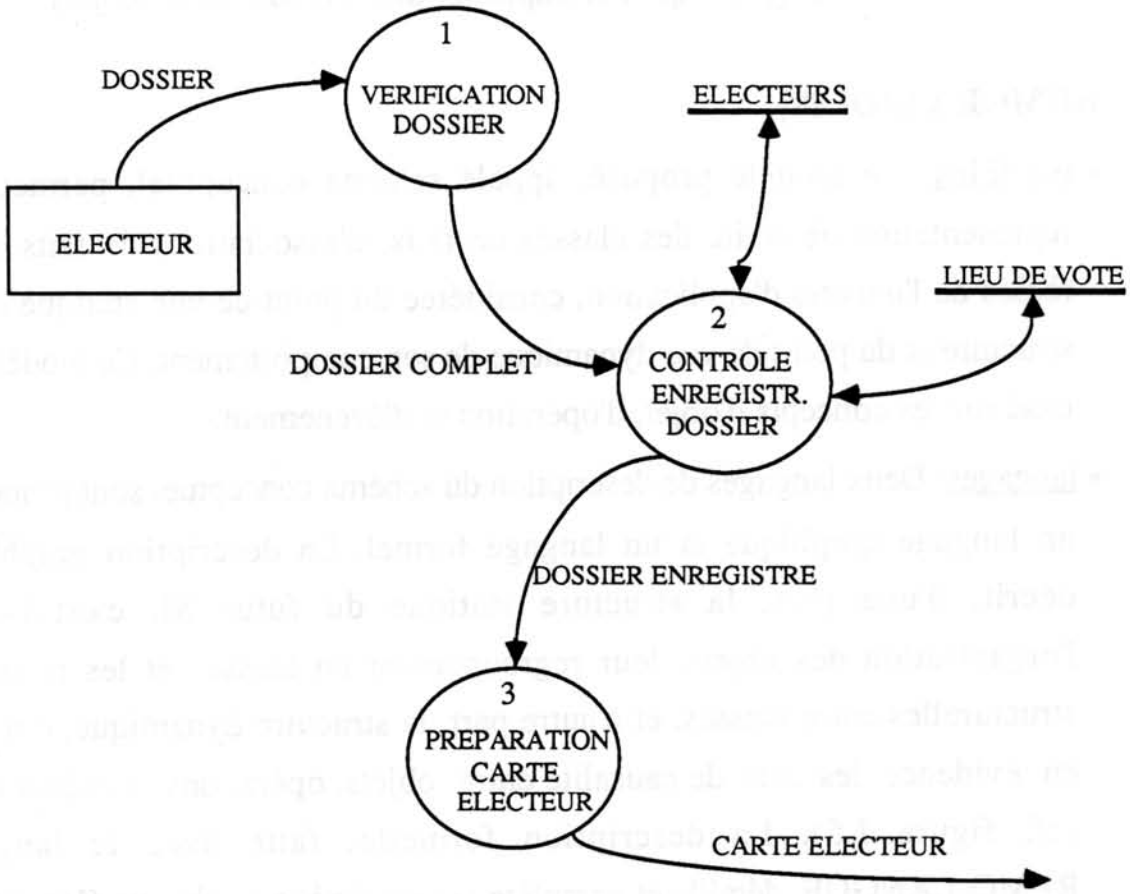


Figure I.4: Exemple de DFD pour un système de gestion d'électeurs.

- démarche: Tom de Marco préconise une démarche en quatre étapes. D'abord, une étude du système existant est faite pour obtenir un modèle physique actuel. Ensuite son équivalent logique est dérivé. A partir de ce modèle, le modèle logique du nouveau système est défini. Puis en adaptant le modèle logique aux caractéristiques de l'organisation on obtient le modèle physique du nouveau système.

- outils: Plusieurs outils supportent cette méthode, intégralement ou en partie, comme par exemple PACBASE [CGI87], Excelerator [IND85], IEW [MAR86] et STP [GAL87]. Excelerator, un des plus diffusés parmi ces outils, comporte les principales fonctions suivantes: graphiques- pour l'élaboration du DFD et du DSD; dictionnaire- pour créer ou modifier les entrées dans le dictionnaire; écrans et rapports- pour concevoir la description des écrans et des rapports; analyse- pour faire certaines validations sur les graphiques et imprimer des relations entre objets.

REMORA [ROL88]

- modèles: Le modèle proposé, appelé schéma conceptuel, permet une représentation abstraite des classes de faits, d'associations de faits et de règles de l'univers d'application, considérée du point de vue statique de sa structure et du point de vue dynamique de son comportement. Ce modèle est basé sur les concepts d'objet, d'opération et d'événement.
- langages: Deux langages de description du schéma conceptuel sont proposés: un langage graphique et un langage formel. La description graphique décrit, d'une part, la structure statique du futur SI, c'est-à-dire, l'organisation des objets, leur regroupement en classes et les relations structurelles entre classes, et d'autre part, la structure dynamique, qui met en évidence les liens de causalité entre objets, opérations et événements (cf. figure I.5). La description formelle, faite avec le langage REMO-LANGUE, détaille et complète la description de chaque élément du schéma, de chaque lien et des contraintes d'intégrité. Ce langage inclut à la fois des constructions relationelles (de type SQL) et des constructions plus algorithmiques (de type PASCAL).
- démarche: La démarche préconisée est basée sur quatre étapes: (i) l'analyse et la description du réel est l'étape de perception, d'identification et d'énumération des classes de phénomènes à prendre en compte; (ii) l'étape de conceptualisation est l'étape de recherche d'une représentation normative ayant des propriétés de cohérence, de complétude et de

non-redondance qui aboutit par la réalisation d'un schéma conceptuel; (iii) l'étape de validation a pour but de valider le schéma conceptuel; (iv) l'étape de spécification a pour but de décrire le schéma dans tous ses détails.

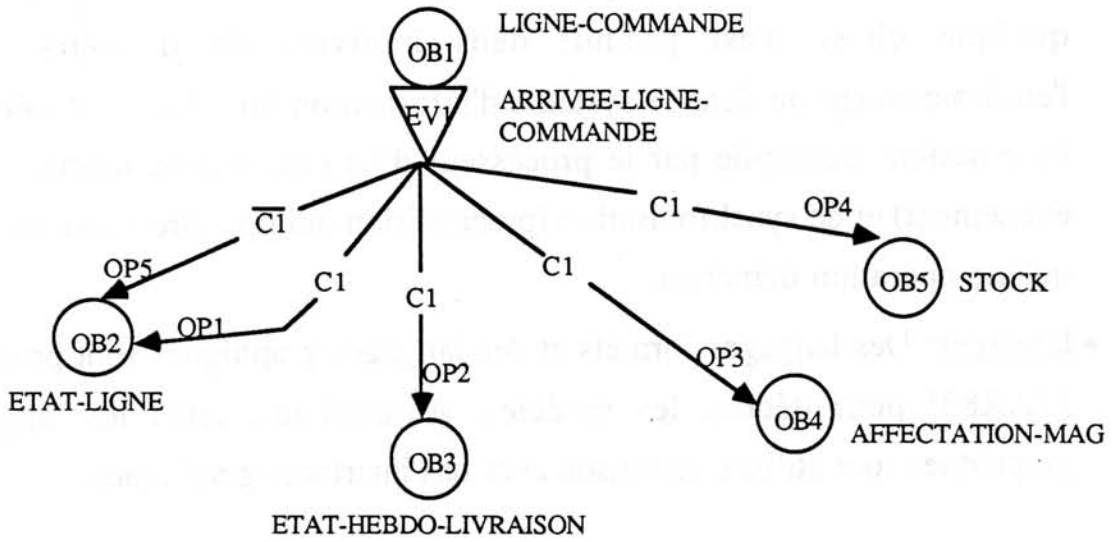


Figure I.5: Exemple de description graphique d'un sous-schéma conceptuel dynamique d'une application de commercialisation [ROL88]

- outils: Il n'y a pas, aujourd'hui, d'outil commercialisé pour supporter cette méthode mais prochainement des outils seront disponibles [LIN88]. Le système expert OICSI en développement à l'Université de Paris [PRO86, CAU88] génère le schéma conceptuel d'une application à partir d'une description faite avec un sous-ensemble du français.

MERISE [TAR83, TAR85, TAB86]

- modèles: La méthode MERISE préconise la modélisation des SI en distinguant ses aspects statiques (données) de ses aspects dynamiques (traitements). Pour les aspects statiques, le modèle conceptuel de données (MCD) a pour but la description des informations mémorisées dans l'entreprise permettant de représenter la sémantique des données en faisant abstraction de toute décision d'automatisation et de répartition. Ce modèle de type entité-association s'appuie sur les concepts d'individu (ou entité) qui

a des propriétés et sur le concept de relation (ou association) qui a aussi des propriétés. Pour caractériser le comportement du système (l'aspect dynamique), le modèle conceptuel des traitements (MCT) de type réseau de Petri est basé sur les concepts d'événement (défini comme le fait que quelque chose s'est produit dans l'univers du discours, dans l'environnement ou dans le système d'information lui-même), d'opération (une action accomplie par le processeur d'information en réaction à un événement) et de synchronisation (précondition qui doit être satisfaite pour qu'une opération démarre).

- langages: Des langages formels et des langages graphiques sont proposés [TAR83] pour décrire les modèles. Aujourd'hui, seuls les langages graphiques sont utilisés, en liaison avec des interfaces graphiques.

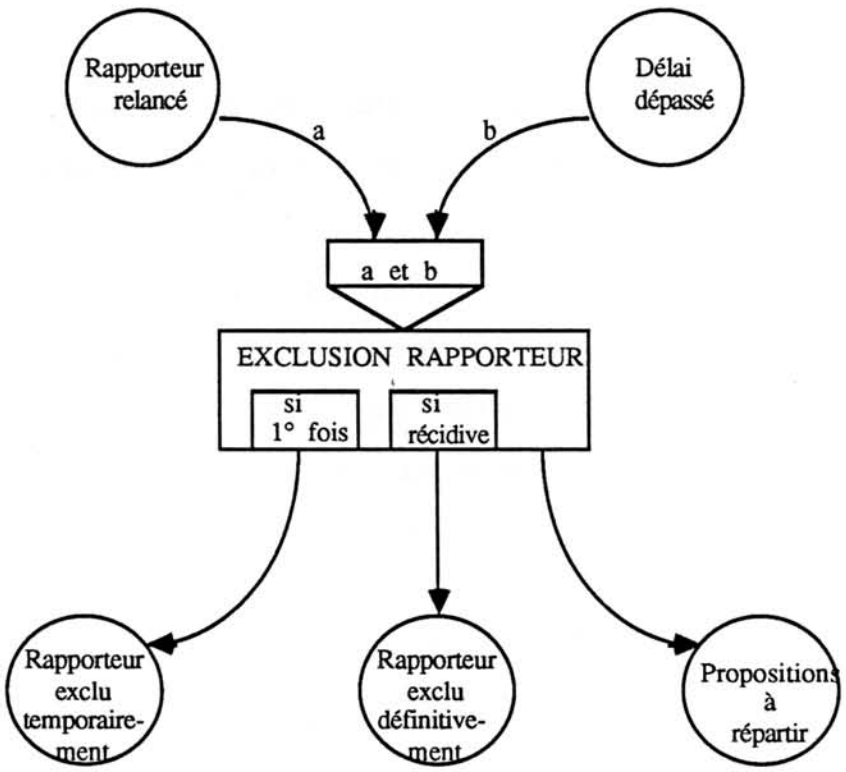


Figure I.6: Exemple de représentation graphique d'un MCT [TAR86]

- démarche: La démarche de la méthode MERISE prévoit une étude préalable réalisée en trois phases: une phase de recueil, une phase de conception et une

phase d'appréciation. A la fin de cette étape on a le modèle conceptuel de données complet et le modèle conceptuel de traitements pour un sous-ensemble représentatif du SI. Dans la deuxième étape, l'étude détaillée, le MCT est complété pour tout le SI et le MCD est validé par la réalisation de modèles conceptuels de données pour chaque traitement du SI.

- outils: Il y a plusieurs outils qui peuvent être utilisés dans le cadre de la méthode Merise comme par exemple Message Conception [CEC87], PACBASE [CGI87], MEGA [ROU87a] et Conceptor [VES87]. Message Conception est formé par trois modules: spécification, maquette et projet. Le module "spécification", organisé à partir d'un dictionnaire de spécifications, permet de décrire et de documenter la plupart des raisonnements mis en œuvre dans la méthode Merise. Ce module assure la vérification (limitée et rigide) des modèles selon les règles propres à Merise. Il contient un guide de la démarche qui rappelle au concepteur les différentes tâches à réaliser à chaque étape de la démarche et propose d'utiliser les fonctions les mieux adaptées à chaque tâche. Le module "maquettage" permet de décrire les futurs traitements informatiques associés aux tâches automatisées dans leur forme de présentation et leur mode de fonctionnement. Le module "projet" permet une estimation des charges d'un projet aux différents stades d'avancement et propose une maîtrise et suivi de projet comportant quatre fonctions: organisation et environnement du projet, répartition dans le temps et affectation, gestion documentaire et événements majeurs du projet.

IDA [BOD88]

- modèles: Cinq modèles sont proposés. Un **modèle de structuration des informations** de type entité-association pour définir la sémantique des données. Un **modèle de structuration des traitements** pour décomposer un projet en traitements de plus en plus élémentaires. Un **modèle de la dynamique des traitements** pour compléter le modèle de structuration des traitements par des conditions de déclenchement, d'exécution et d'enchaînement des traitements en vue de caractériser le

comportement du SI. Un **modèle de la statique des traitements** pour préciser pour chaque traitement les messages-données, les messages-résultats, la partie de la mémoire du SI utilisée et la procédure de traitement qui assure la transformation. Un **modèle des ressources** pour caractériser les processeurs qui exécutent les procédures de traitement et les ressources nécessaires. Les modèles IDA sont développés en Annexe A.

- langages: La méthode IDA utilise le langage DSL (Dynamic Specification Language) de la famille PSL (Problem Statement Language) définie dans le projet ISDOS-PRISE [TEI77]. C'est un langage non procédural au sens où il permet la spécification en ordre quelconque et de manière progressive du SI. Il est construit à partir des notions de type d'objet, type d'association et des propriétés associées à ces types d'objet ou d'association. Il existe aussi un langage graphique pour chacun des modèles.

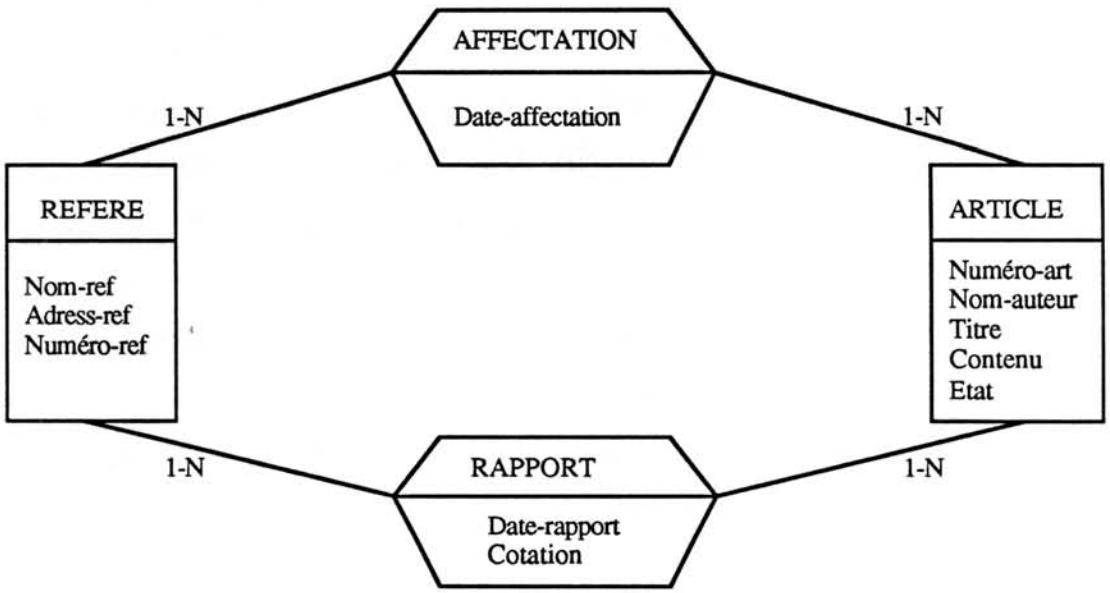


Figure I.7: Exemple de description graphique selon le modèle de structuration des informations [BOD86]

- démarche: La méthode IDA prévoit une démarche en deux étapes: étude d'opportunité et analyse conceptuelle. L'étude d'opportunité a pour objectif de préparer un avant-projet de solution à partir de besoins exprimés par l'organisation. L'analyse conceptuelle comprend deux sous-étapes

majeures: (i) l'élaboration et la validation des sous-schémas conceptuels pour chaque "phase" (cf. Annexe) du "projet" et (ii) l'élaboration et la validation du schéma conceptuel global par consolidation des différentes "phases" du "projet".

- outils: Une base de données des spécifications constitue le cœur du système. Elle regroupe l'ensemble des spécifications du SI introduites et devient la source unique de toute la documentation. Des spécifications stockées dans cette base sont rédigées à l'aide du langage DSL et sont conformes aux modèles énoncés ci-dessus. Un langage interactif d'interrogation de la base de données permet la sélection d'objets répondant à certains critères et permet d'effectuer visuellement des contrôles de cohérence et de complétude des descriptions introduites. Un éditeur syntaxique du langage DSL, intégré dans un poste de travail sur micro-ordinateur, facilite la saisie et la mise à jour des spécifications. Un ensemble de rapports documentaires et d'analyse présente sous plusieurs formes (narratives, listes, tableaux, graphiques) différents aspects du système spécifié, à l'usage de diverses personnes impliquées dans le projet: responsables, analystes, programmeurs, utilisateurs. Un générateur automatique d'un programme de simulation sert à évaluer le caractère réalisable du système décrit. Un générateur d'une maquette programmée du système futur a pour but de tester le caractère effectif des spécifications.

I.2.3 Conception assistée par ordinateur

Après avoir précisé la notion de système d'information, schématisé son cycle de vie, caractérisé les quatre composantes essentielles des méthodes de modélisation de ces systèmes et présenté brièvement quelques méthodes, nous pouvons évoquer la notion de conception assistée par ordinateur (CAO) appliquée au domaine de la modélisation des SI.

N. Giambiasi [GIA85] définit la CAO, indépendamment du domaine d'application, comme

> "l'ensemble des outils et procédures utilisant l'informatique et permettant d'établir une synergie entre l'homme et l'ordinateur en mettant au mieux à profit leurs qualités complémentaires: d'une part en confiant à l'ordinateur les travaux de stockage d'information et d'analyse routinière fastidieux et automatisables . . . et d'autre part, en laissant à l'homme les travaux de synthèse créatrice".

Parmi les objectifs principaux de la CAO dans le domaine de la conception des SI nous pouvons citer:

- Améliorer la qualité et la fiabilité des applications en mettant à la disposition du concepteur des moyens perfectionnés d'analyse des caractéristiques des spécifications;
- Réduire les délais et les coûts de conception en assurant l'automatisation de tâches comme la documentation et certains contrôles;
- Augmenter la créativité en permettant l'investigation d'un nombre plus élevé de solutions sans recourir à une expérimentation longue ou à des réalisations coûteuses;
- Pallier le manque de main-d'œuvre par l'augmentation de la productivité individuelle;
- Faciliter l'archivage et la circulation de l'information.

De l'analyse des systèmes de CAO appliqués au domaine de la modélisation des SI (CAO.SI) nous pouvons faire ressortir trois générations d'outils. Les outils de la première génération sont conçus pour traiter des tâches spécifiques et ponctuelles, sans liaisons avec d'autres outils. La structure rigide de ces outils les rend peu évolutifs et, par la suite, sujets à une obsolescence rapide. Ils ne peuvent pas être rapidement et facilement adaptés aux évolutions des procédures de conception et des technologies. La deuxième

génération d'outils est caractérisée par le développement de systèmes complexes possédant dans son noyau une base de données des spécifications; ce sont les systèmes intégrés de CAO.SI. Ces systèmes ont une structure modulaire qui autorise (au moins en théorie) les évolutions inévitables de tout outil de CAO. Le dialogue avec l'utilisateur est textuel. Les outils de la troisième génération ont comme caractéristique principale l'utilisation de postes de travail avec interface graphique pour le dialogue et pour la spécification des modèles.

Dans leur majorité, les logiciels de CAO.SI actuels servent à décrire, manipuler, analyser, simuler ou documenter les multiples représentations des applications à concevoir. Leur objectif principal est d'aider à spécifier et à valider les solutions proposées par le concepteur; en fait, ils ne participent pas activement aux nombreuses prises de décision que comporte tout processus de conception et, surtout sont incapables de modifier par eux-mêmes la structure de l'application à concevoir en fonction d'objectifs fixés a priori.

Cette constatation est à la base des outils de la quatrième génération qui feront appel aux techniques de l'intelligence artificielle.

I.3 Approche système expert

Jusqu'à présent la grande majorité des environnements de spécification des SI sont basés sur des techniques de Bases de Données classiques et offrent des services spécifiques, c'est-à-dire utilisent des outils pour faciliter l'exécution des tâches pratiques de l'analyste-concepteur sans prendre en compte les problèmes de décision et d'organisation des tâches. Mais cela est insuffisant pour assurer la qualité des systèmes.

Pour pallier les limites de ces outils il est envisageable de mettre en place des environnements de résolution de problèmes (problem-solving environments) centrés sur une Base de Connaissances des objectifs, des tâches et des stratégies de développement [GID84] adaptée au domaine de la modélisation des SI.

Partant de cet état de faits, plusieurs projets utilisent une nouvelle approche d'outils pour l'analyse et la conception basée sur des techniques de l'Intelligence Artificielle, et plus spécialement sur les Systèmes Experts.

Le système OICSI [PRO86, CAU88] utilise une approche système expert pour assister ses utilisateurs dans l'analyse et la conception des SI. OICSI a la tâche de prendre en compte les phrases émises en langue naturelle (français) et de produire le schéma conceptuel Remora correspondant.

Le système CAPRI [FRA86] est un exemple de l'utilisation de l'approche système expert pour l'estimation des ressources requises pour réaliser un projet de développement de logiciel dans un environnement donné.

Le système Analyst [STE85] est un système d'aide à l'analyse et à la conception développé dans le cadre de la méthode CORE [MUL79] qui utilise aussi une approche système expert.

L'utilisation de cette approche système expert pour la modélisation d'une base de données est aussi l'objet de plusieurs projets [BOU83, BRI85, GIR85, BER86].

H. Farreny [FAR85] définit fonctionnellement les systèmes experts comme

> "des logiciels (peut-être bientôt aussi des matériels) destinés à remplacer ou assister l'homme dans des domaines où est reconnue une expertise humaine:
> - insuffisamment structurée pour constituer une méthode de travail précise, sûre, complète, directement transposable sur ordinateur,
> - sujette à révisions ou compléments (selon l'expérience accumulée)".

L'architecture classique d'un système expert est représentée dans la figure I.8. Ses composantes de base sont:

- Une **base de connaissances** qui contient tout le savoir nécessaire au système pour être expert dans son domaine d'application. Couramment on distingue dans cette base des "connaissances assertionnelles" (**faits**) et des "connaissances opératoires" (**règles**).
- Un **moteur d'inférences** qui est chargé d'exploiter les connaissances afin de résoudre un problème donné.
- Des **interfaces** qui assurent, d'une part, le dialogue avec l'expert pour l'acquisition et la validation de connaissances et, d'autre part, le dialogue avec les utilisateurs du système.

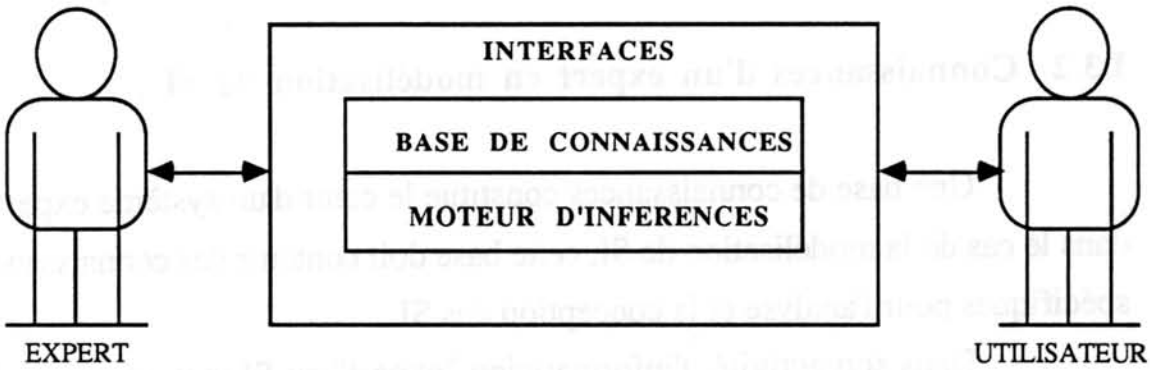


Figure I.8: Architecture d'un système expert classique

I.3.1 Avantages escomptés

Les principaux avantages escomptés d'une telle approche sont:

- l'existence d'une base de connaissances formelle sur le sujet en question, la modélisation des SI, intégrant les concepts sur des modèles, des langages et des démarches mais aussi des connaissances expérimentales de chaque concepteur de SI;
- la spécification des SI avec un grand degré d'expertise par des informaticiens moins expérimentés ou même par des utilisateurs, dès que les connaissances de chaque expert seront partageables et communicables;

- la facilité d'intégration de nouveaux concepts, nouvelles règles formelles et de nouvelles connaissances expérimentales spécifiques de chaque concepteur;
- un environnement interactif convivial pour piloter le processus de modélisation à partir d'une telle base de connaissances;
- la formation des informaticiens et leur perfectionnement à la modélisation de systèmes grâce aux connaissances disponibles dans cette base;
- la possibilité d'adaptation du système au profil de l'utilisateur;
- la facilité d'adaptation du système aux caractéristiques particulières de l'application à spécifier.

I.3.2 Connaissances d'un expert en modélisation de SI

Une base de connaissances constitue le cœur d'un système expert et dans le cas de la modélisation de SI, cette base doit contenir des connaissances spécifiques pour l'analyse et la conception des SI.

Dans son activité, l'informaticien "expert" en SI met en œuvre des connaissances que l'on peut regrouper en trois catégories:

- connaissances en analyse et conception;
- connaissances d'un domaine spécifique d'application;
- connaissances d'une méthode.

I.3.2.1 Connaissances en analyse et conception

Ces connaissances sont très variées. Certaines sont d'ordre général et sont indispensables pour le travail d'analyse, comme par exemple: techniques d'interview, utilisation de plusieurs sources d'information, élaboration de questionnaires, etc... Il existe d'autres connaissances plus spécifiques telles que:

- **procédures pour livrer un système plus sûr**: mots de passe, encryptage de quelques informations critiques, limitation physique de l'accès des personnes, procédures de reprise en cas de panne matérielle;
- **considérations sur la souplesse d'un système**: recherche lexicographique complétée par des recherches phonétiques ou de proximité pour certains systèmes intelligents de sélection d'informations;

I.3.2.2 Connaissances d'un domaine spécifique d'application

Des études [SAC68] ont montré qu'une équipe d'informaticiens fait un deuxième système du même type beaucoup plus vite que le premier. Nous supposons que cela est dû aux connaissances obtenues et à l'expérience acquise sur le domaine. Alors, on pourrait faire des bases de connaissances par types d'applications comme comptabilité, paye ou contrôle de stocks. Pour chacun de ces domaines on aurait des caractéristiques essentielles, comme par exemple pour le contrôle de stocks:

- structure des principaux objets du domaine;
- liaison avec d'autres systèmes: comptabilité, achat, vente;
- mode de calcul du "seuil de recommande";
- édition d'un rapport périodique sur les produits qui sont au dessous du stock minimum (seuil de recommande) ou émission directe des ordres d'achat?
- édition des principaux rapports statistiques.

Ces bases de connaissances serviraient de support d'apprentissage, elles permettraient en particulier de suggérer des structures d'objets dans un domaine spécifique [MIT82, MIC83, FIS85].

On peut considérer que les progiciels (comptabilité, paye, stock,...) commercialisés dans les années 70 constituent une forme primitive de bases de connaissances de ces domaines.

I.3.2.3 Connaissances d'une méthode

Ce type de connaissances correspond aux quatre notions évoquées ci-dessus: modèles, langages, démarches et outils.

Les connaissances relatives aux modèles et aux langages consistent à décrire leurs formalismes: schémas, grammaires et contraintes. Les connaissances relatives à une démarche portent sur la manière d'accomplir un travail de modélisation, c'est à dire la prise en compte de quatre interrogations permanentes en analyse pour chaque tâche à assurer: **quoi** (concepts à utiliser et activités à réaliser); **comment** (formalismes à respecter); **quand et qui** (ordonnancement et affectation des activités). Les connaissances sur des outils portent sur la fonction et l'usage de chaque outil.

Notons que la difficulté essentielle se situe au niveau de la démarche par manque de formalisation et surtout à cause de la flexibilité nécessaire. Une démarche doit être personnalisée: comment voir un travail d'analyse et de conception comme un acte de production combiné avec une imagination créatrice indispensable?

La connaissance requise pour choisir les modèles, les langages, les démarches et les outils à utiliser dans une application spécifique et avec une certaine équipe d'informaticiens peut être vue comme une méta-connaissance: elle est basée en grande partie sur l'expérience.

I.4 Conclusion

Ce premier chapitre a permis de fixer le domaine général concerné par notre travail.

Le chapitre suivant propose une démarche rigoureuse pour représenter les connaissances liées à une méthode: notre étude ne tiendra compte que de ces connaissances. Les connaissances en analyse et conception et les connaissances d'un domaine spécifique d'application ne seront pas abordées.

CHAPITRE II

REPRESENTATION D'UNE METHODE DE MODELISATION

REPRESENTATION D'UNE METHODE DE MODELISATION

II.1 Introduction

L'objectif de ce chapitre est de fixer un formalisme précis de représentation d'une méthode de modélisation des SI afin de pouvoir aborder proprement le problème du pilotage.

Notons que notre but est de représenter formellement une méthode existante et non de définir une nouvelle méthode.

Nous présentons d'abord une idée générale de la façon de caractériser une méthode. Nous proposons ensuite un formalisme de représentation, puis nous définissons une méthode selon ce formalisme et nous abordons finalement les problèmes de complétude et de cohérence de spécifications.

II.2 Stratégie de représentation d'une méthode

Au chapitre I, nous avons caractérisé une méthode de modélisation de S.I. par quatre composantes: modèles, langages, démarches et outils. Nous proposons de représenter rigoureusement les trois premières composantes avec un formalisme unique.

Un modèle est décrit par un ensemble de règles qui définissent le sous-ensemble du langage de spécification utilisé et par les conditions de complétude et de cohérence des spécifications.

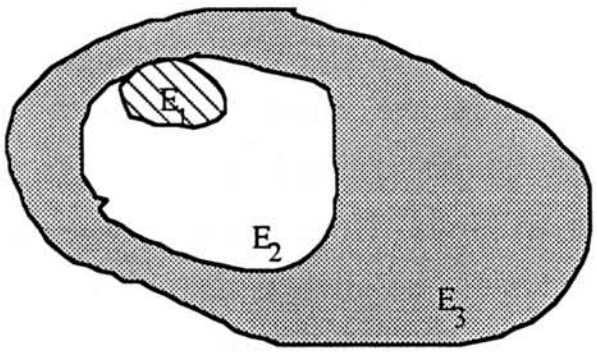
Nous ne considérons pas le processus de modélisation des SI du point de vue de la gestion d'un projet mais seulement en cherchant à établir les résultats; ainsi nous étudions la démarche comme une séquence d'états de la spécification à partir d'un état initial (par exemple l'ensemble vide) jusqu'à l'état final où la modélisation est considérée comme terminée: la spécification

doit être complète et cohérente. Dans ce cadre, nous caractérisons une étape de la démarche comme la transition d'un "état i" à un "état j" de la spécification.

Exemple: Soit une démarche D organisée en trois étapes:

- à la fin de l'étape 1, la spécification doit être dans l'état E_1;
- à la fin de l'étape 2, la spécification doit être dans l'état E_2;
- à la fin de l'étape 3, la spécification doit être dans l'état E_3;

Nous pouvons représenter graphiquement l'évolution d'une spécification d'une application A selon cette démarche D:



La partie grisée correspond à l'étape 3 de la démarche, qui est le passage de l'état E_2 à l'état E_3 de la modélisation.

Un état potentiel de la spécification peut être défini par un ensemble d'objets, de relations et de règles qui caractérisent le contenu possible d'une spécification.

Le formalisme de représentation que nous utilisons est un sous-ensemble du langage Z [ABR74, ABR78, DEL82, MEY78]. Nous retenons dans ce sous-ensemble les concepts pour définir:

- des ensembles d'entités;
- des relations binaires entre entités;
- des formules logiques.

Nous aurions pu utiliser d'autres techniques voisines [BRA76, CHE76, VER82], mais notre choix a été fortement influencé par la simplicité et la puissance d'expression de Z, ainsi que par son utilisation dans notre

environnement et par son adéquation à une transformation en spécification exécutable.

L'idée de base est de décrire les composantes d'une méthode par des concepts spécifiques de Z.

- langage:

  La formalisation d'un langage est intégralement réalisée par des définitions d'ensembles d'entités et de relations binaires. Le but est de constituer un ensemble d'énoncés qui caractérisent le vocabulaire et la syntaxe du langage de spécification.

- modèle:

  La formalisation d'un modèle s'appuie sur des définitions d'ensembles d'entités et de relations binaires complétées par des formules. Le but est de décrire un ensemble d'énoncés qui caractérisent un modèle par des restrictions appliquées au langage de spécification. Nous montrons par la suite l'existence d'une dépendance état-modèle (cf. §II.4.6).

- démarche:

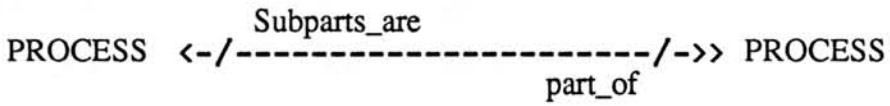
  La formalisation d'une démarche est constituée de définitions d'ensembles d'entités et de relations binaires complétées par des formules. Le but est de grouper des ensembles d'énoncés pour caractériser chaque état de la démarche; une démarche est décrite par une séquence d'états.

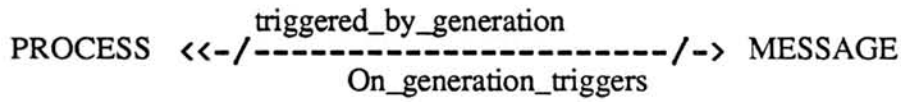
Notons que l'ordre usuel de définition d'une méthode est de commencer par les modèles, puis de décrire les langages associés aux modèles, de déterminer ensuite une démarche de modélisation et finalement de choisir des outils. C'est pour faciliter la représentation formelle d'une méthode que nous commençons par la description du langage avant d'aborder diverses restrictions pour caractériser des modèles selon une démarche spécifique.

II.3 Formalisme utilisé

Le principe de Z est de permettre de décrire des classes d'information appelées des **ensembles d'entités** et de définir entre deux ensembles d'entités des relations binaires appelées **associations** caractérisées par des sortes de fonctions qui précisent les rôles joués par les ensembles d'entités arguments des relations et qui correspondent à des interprétations de ces relations. Ces définitions peuvent être complétées par des **formules logiques** pour exprimer des contraintes sur les ensembles ou sur les associations.

Exemple: Représentation en Z de la décomposition hiérarchique des processus et du déclenchement d'un processus par l'arrivée d'un message (cf. langage DSL/IDA):





PROCESS et MESSAGE sont les deux ensembles d'entités introduits par cette représentation. La première association exprime le fait qu'un processus peut (--/-) être décomposé en plusieurs (-->>) processus fils et qu'à un processus peut correspondre au plus un (-->) processus père. La deuxième association indique qu'un processus peut être déclenché par l'arrivée d'un message et que l'arrivée d'un message peut déclencher plusieurs processus.

II.3.1 Ensembles d'entités

Un ensemble d'entités peut être désigné par:

- un nom caractérisé par un identificateur formé de lettres majuscules;

Exemples: PROCESS, MESSAGE, CALENDAR, ENTIER;

- une définition en extension;

Exemple: {projet, application, phase, fonction}

- des opérations ensemblistes union, intersection, différence et produit cartésien sur des ensembles d'entités;

Exemples: MESSAGE x CONDITION x ENTIER

GROUP ∪ ELEMENT

- une définition en intention;

Exemple: {p ∈ PROCESS | card(Subparts_are(p) ≤ 1)}

- une combinaison des cas ci-dessus.

Exemple: MESSAGE x (GROUP ∪ ELEMENT) x {one, many}

Nous appelons **ensemble composite** un ensemble d'entités formé par un produit cartésien où au moins un des ensembles est optionnel. Un ensemble E optionnel est noté E°.

Exemple: Soient $A=\{a_1,a_2\}$, $B=\{b_1,b_2\}$ et $C=\{c_1,c_2,c_3\}$.

L'ensemble $X = A \times B^{\circ} \times C$ est un ensemble composite et par exemple

$(a_2,-,c_3) \in X$, $(a_1,b_2,c_2) \in X$ mais, $(-,b_2,c_2) \notin X$

Ce concept d'ensemble composite est une extension au formalisme Z pour faciliter l'écriture des énoncés de définition des langages que nous traitons.

Notons que dans un ensemble composite au moins un des ensembles doit être obligatoire (non optionnel).

II.3.2 Associations

La description d'une **association** est réalisée par:

- la désignation des ensembles d'entités;
- la définition précise des fonctions: nom, portée partielle ou totale sur l'ensemble source et cardinalité monovaluée ou multivaluée sur l'ensemble cible.

Conventions de dessin et d'écriture des fonctions:

------>	fonction monovaluée totale
----/->	fonction monovaluée partielle
----->>	fonction multivaluée totale
---/->>	fonction multivaluée partielle

Le nom d'une fonction monovaluée est un identificateur formé par des caractères minuscules et le nom d'une fonction multivaluée est un identificateur commençant par une lettre majuscule.

Sémantique des fonctions:

Si $E \subset E_1 \times E_2 \times ... \times E_n$ et $x_i \in E_i$ on peut noter:

$\mathbf{x} = (x_1, x_2, ..., x_n)$, $\mathbf{x} \in E$ ou $(x_1, x_2, ..., x_n) \in E$.

Dire que la fonction f est monovaluée et totale ($E_1 \xrightarrow{f} E_2$)

signifie que: $\forall \mathbf{x_1} \in E_1 \,.\, \exists^1 \mathbf{x_2} \in E_2 \,.\, \mathbf{x_2} = f(\mathbf{x_1})$

La fonction f monovaluée et partielle ($E_1 \xrightarrow{f} E_2$) signifie que:

$\forall \mathbf{x_1} \in E_1 \,.\, (f(\mathbf{x_1}) = \text{nil} \vee (\exists^1 \mathbf{x_2} \in E_2 \,.\, \mathbf{x_2} = f(\mathbf{x_1})))$

Si F est une fonction multivaluée et totale ($E_1 \xrightarrow{F} E_2$) alors:

$\forall \mathbf{x_1} \in E_1 \,.\, \exists\, \mathbf{x_2} \in E_2 \,.\, \mathbf{x_2} \in F(\mathbf{x_1})$

Une fonction F multivaluée et partielle ($E_1 \xrightarrow{F} E_2$) signifie que:

$\forall \mathbf{x_1} \in E_1 \,.\, (F(\mathbf{x_1}) = \phi \vee (\exists\, \mathbf{x_2} \in E_2 \,.\, \mathbf{x_2} \in F(\mathbf{x_1})))$

Conventions complémentaires utilisées:

nil est l'entité indéfinie

ϕ est l'ensemble vide

$\exists^1$ signifie "il existe un et un seul"

Une association entre deux ensembles d'entités n'est pas complétement décrite par une seule fonction monovaluée ou multivaluée, totale ou partielle. Pour lever toute ambiguïté sur les liens possibles entre entités de ces ensembles arguments il est indispensable de caractériser de la même manière la fonction inverse. Prenons deux formes d'associations intégralement décrites:

$$E_1 \overset{F_1}{\underset{F_2}{<\!\!-/\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-/\!\!-\!\!>\!\!>}} E_2$$

$$\forall x_1 \in E_1 \,.\, \forall x_2 \in E_2 \,.\, (x_2 \in F_1(x_1) \;\equiv\; x_1 \in F_2(x_2))$$

$$E_1 \overset{F_1}{\underset{f_2}{<\!\!-/\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-/\!\!-\!\!>\!\!>}} E_2$$

$$\forall x_1 \in E_1 \,.\, \forall x_2 \in E_2 \,.\, (x_2 \in F_1(x_1) \;\equiv\; x_1 = f_2(x_2))$$

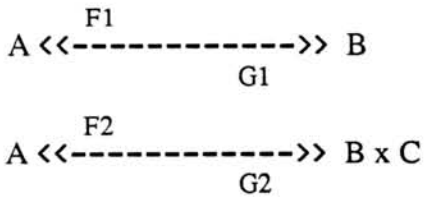
Nous appelons **association composite** une association où au moins un des arguments est un ensemble composite.

Exemple:

$$A \overset{F}{\underset{G}{<\!\!<\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!-\!\!>\!\!>}} B \times C^{\circ}$$

Notons que nous pouvons représenter une association composite avec plusieurs associations sans arguments optionnels.

Exemple: L'association composite de l'exemple précédent peut être représentée par les deux associations



Néanmoins, cette multiplication d'associations peut rendre le schéma inutilement complexe. En effet, le nombre d'associations nécessaires pour représenter une association composite est 2^n où n est le nombre d'arguments optionnels dans l'association composite.

Lien [LIE79] et Zaniolo [ZAN84] ont étudié formellement le problème de la valeur nulle interprétée comme "valeur inexistante" par opposition à l'interprétation "valeur inconnue" dans le cadre du modèle relationnel de données.

Qu'est qu'une spécification?

Une spécification selon un modèle et un langage peut être vue comme un ensemble de faits engendrés à partir d'expressions Z de ce modèle et de ce langage. Les **faits** sont de la forme: $\mathbf{x} \in E$, $\mathbf{x}=f(\mathbf{y})$ ou $\mathbf{x} \in F(\mathbf{y})$.

Exemple: Soit la représentation d'un processus Inscription qui a pour synonyme Inscription-de-l'électeur et qui fait partie d'un système électoral. Dans un langage de spécification comme DSL/IDA on peut écrire:

```
DEFINE PROCESS inscription;                (1)
       SYNONYM inscription-de-l'électeur;  (2)
       PART OF système-électoral;          (3)
```

(1), (2) et (3) sont des faits qui peuvent être représentés par:

(1) inscription ∈ PROCESS

(2) inscription-de-l'électeur ∈ Synonym (inscription)

(3) part-of (inscription) = système-électoral

si le modèle et le langage DSL/IDA ont été décrits en Z par:

$$\text{PROCESS} \;\; \underset{\text{part\_of}}{\overset{\text{Subparts\_are}}{\text{<-/----------/->>}}} \;\; \text{PROCESS} \qquad \text{(a)}$$

$$\text{PROCESS} \;\; \overset{\text{Synonym}}{\text{<-/----------/->>}} \;\; \text{PROCESS} \qquad \text{(b)}$$

Notons que dans la description de la deuxième association (b), nous avons volontairement omis de donner un nom à la fonction inverse de la fonction "Synonym" mais nous en avons donné ses caractéristiques fondamentales (monovaluée et partielle). Ce choix indique que le langage n'autorise qu'un seul mode d'introduction des faits relatifs à cette association. Dans la première association, la suppression de la fonction "Subparts_are" correspond à une technique de spécification ascendante, alors que la suppression de la fonction "part_of" contraint à une approche descendante.

II.3.3 Expressions logiques

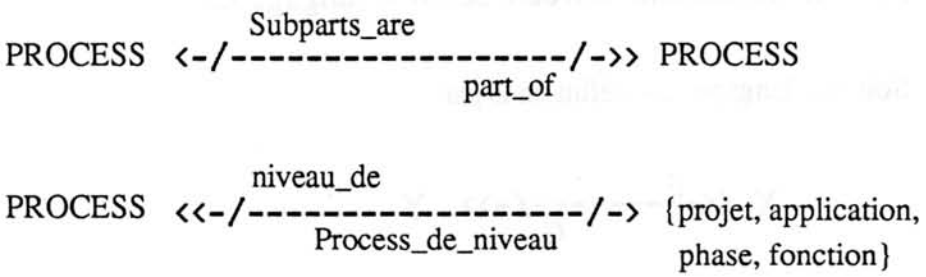
Pour décrire formellement une méthode il faut non seulement spécifier des ensembles d'entités et des associations entre entités pour classer les objets et leurs liens de dépendance, mais aussi des règles pour contrôler la complétude et la cohérence d'une spécification et organiser le tout selon une démarche. Ces règles peuvent être énoncées comme des contraintes d'intégrité exprimables dans le langage Z par des expressions logiques.

Ces expressions logiques sont des formules qui obéissent aux règles classiques suivantes [DEL82]:

- (r1): les symboles désignant des entités élémentaires, les identificateurs de variables d'entités sont des termes entités (nil est un terme entité); les identificateurs d'ensembles sont des termes ensembles (ϕ est un terme ensemble);
- (r2): si g est un symbole de fonction monovaluée ayant n paramètres, et si $t_1, t_2, ..., t_n$ sont des termes entités, alors $g(t_1, t_2, ..., t_n)$ est un terme entité;

- (r3): si $t_1, t_2, ..., t_n$ sont des termes entités, le n-uplet $(t_1, t_2, ..., t_n)$ est un terme entité;
- (r4): si G est un symbole de fonction multivaluée ayant n paramètres et $t_1, t_2, ..., t_n$ sont des termes entités, alors $G(t_1, t_2, ..., t_n)$ est un terme ensemble;
- (r5): si G est un symbole de fonction multivaluée et $t_1, t_2, ..., t_n$ des termes entités, alors $(t_{k+1}, t_{k+2}, ..., t_n) \in G(t_1, t_2, ..., t_k)$ est une formule atomique à condition que G admette k paramètres et prenne son résultat dans n-k valeurs;
- (r6): si t_1 et t_2 sont des termes entités et θ un opérateur de comparaison: $=, \neq, <, \leq, >, \geq$, alors $t_1 \theta t_2$ est une formule atomique;
- (r7): si T_1 et T_2 sont des termes ensembles et ψ un opérateur de comparaison: $=, \neq, \subset, \subseteq, \supset, \supseteq$ alors $T_1 \psi T_2$ est une formule atomique;
- (r8): une formule atomique est une formule;
- (r9): si F_1 et F_2 sont des formules alors $F_1 \wedge F_2$, $F_1 \vee F_2$, $F_1 \Rightarrow F_2$ et $\neg F1$ sont des formules;
- (r10): si F est une formule, où x est une variable libre qui prend ses valeurs dans l'ensemble E, alors $\forall x \in E.\ F$ et $\exists x \in E.\ F$ sont des formules;
- (r11): si F est une formule, alors (F) est une formule; les parenthèses peuvent être omises à condition de définir un ordre de priorité sur les opérateurs.

Exemple: Soit la modélisation Z d'une partie très réduite du langage DSL/IDA



$\forall p \in PROCESS \; . \; (part\_of(p) \neq nil \; \vee \; niveau\_de(p) = projet)$ (1)

$\forall p \in PROCESS \; . \; niveau\_de(p) \neq nil$ (2)

(1) et (2) sont des formules.

Notons que la deuxième formule impose une restriction sur le langage de base en indiquant que tout PROCESS doit avoir un niveau défini; c'est une sorte de redéfinition de la fonction monovaluée partielle "niveau_de" qui devient totale pour une certaine étape souhaitée de la spécification.

Dans les chapitres suivants, à la place de **formule** ou d'**expression logique**, nous employons les termes **règle** ou **contrainte d'intégrité** conformément au vocabulaire des domaines des systèmes experts et des bases de données.

II.4 Description formelle d'une méthode de modélisation

Ce même formalisme Z est utilisable pour caractériser complétement une méthode en décrivant successivement le langage de spécification, les modèles de représentation et la démarche associée.

Cependant des définitions complémentaires sont nécessaires pour décrire plus formellement une démarche.

II.4.1 Ensemble maximal de faits

Nous appelons **ensemble maximal de faits d'un langage L**, noté **EMF(L)**, l'ensemble de tous les faits qu'il est possible d'engendrer par

les énoncés Z qui décrivent ce langage. Ces faits correspondent à une spécification syntaxiquement correcte selon le langage L.

Exemple: Soit un langage L_1 défini en Z par

$$X \;\langle\!\langle \xrightarrow[G]{F} \!\!/\!\rightarrow\rangle\!\rangle\; Y$$

avec $X \subseteq \{n \in \mathbb{N} \mid 0<n\leq 3\}$

et $Y \subseteq \{n \in \mathbb{N} \mid 0<n<3\}$

Il engendre des faits de la forme $x \in X$, $y \in Y$ et $y \in F(x)$

(notons que le fait $y \in F(x)$ est identique au fait $x \in G(y)$ (cf. §II.3.2)).

Alors l'EMF associé est:

$EMF(L_1) = \{1 \in X, 2 \in X, 3 \in X, 1 \in Y, 2 \in Y, 1 \in F(1), 2 \in F(1), 1 \in F(2), 2 \in F(2),$

$1 \in F(3), 2 \in F(3)\}$

II.4.2 Modèle d'un langage

Un **modèle d'un langage L**, noté **M(L)**, peut être défini par les énoncés Z (ensembles d'entités et associations) relatifs au langage L complétés par un ensemble de formules qui expriment des restrictions sur l'utilisation du langage L.

Exemple: Soit le modèle $M(L_1)$ défini par les énoncés du langage L_1 de l'exemple précédent et par les formules complémentaires:

$$\forall x \in X \,.\, \forall y \in Y \,.\, (y \in F(x) \Rightarrow x=y) \qquad (a)$$
$$card(Y) = 2 \qquad (b)$$

(**card** est une fonction monovaluée totale pré-définie qui associe à un terme ensemble le nombre d'entités (d'éléments) de cet ensemble)

Notons que la deuxième formule introduit une restriction de l'ensemble Y:

$Y \subseteq \{n \in \mathbb{N} \mid 0<n<3\}$ et $card(Y) = 2$ est équivalent à $Y = \{1, 2\}$

Cette définition de modèle est conforme à la notion présentée au chapitre I (cf. §I.2.2) mais le point de vue est différent. Dans le chapitre I

nous considérons un modèle associé à un "aspect" du SI, comme par exemple "les données", "les traitements", "les ressources". Par la suite, nous utilisons un modèle d'un point de vue plus évolutif selon le déroulement d'une modélisation, c'est-à-dire pour définir les étapes d'une démarche. Néanmoins, un tel modèle peut aussi correspondre à un aspect particulier du SI.

II.4.3 Sous-ensemble de faits selon un modèle

Nous notons **SE(M(L))** un sous-ensemble de l'EMF(L) où tous les faits satisfont les formules d'un modèle M(L) défini par rapport au langage L.

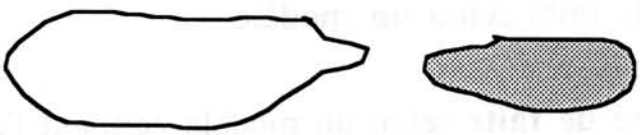
Exemple: Soit le modèle $M(L_1)$ de l'exemple précédent. On peut en déduire un sous-ensemble:

$$SE(M(L_1)) = \{1 \in X, 2 \in X, 3 \in X, 1 \in Y, 2 \in Y, 1 \in F(1), 2 \in F(2)\}$$

Sur un même langage L, il est possible de définir plusieurs modèles $M_i(L)$, $M_j(L)$, ... qui se distinguent par des restrictions (formules) différentes. Les sous-ensembles de faits correspondants peuvent être:

- indépendants:

$$SE(M_i(L)) \cap SE(M_j(L)) = \phi$$



- dépendants:

$$SE(M_i(L)) \supset SE(M_j(L)) \quad \vee \quad SE(M_j(L)) \supset SE(M_i(L)) \quad \vee$$

$$(SE(M_i(L)) \cap SE(M_j(L)) \neq \phi \wedge SE(M_i(L)) \cap SE(M_j(L)) \neq SE(M_i(L))$$

$$\wedge SE(M_i(L)) \cap SE(M_j(L)) \neq SE(M_j(L)))$$

En général, sur un même langage, nous construisons des modèles successifs par ajout, suppression et/ou modification de formules qui engendrent des sous-ensembles de faits dépendants.

II.4.4 Sous-ensemble minimal de faits selon un modèle

On appellera $\mathbf{SE_{min}(M(L))}$ un ensemble de faits d'un modèle M(L) tel que le retrait de l'un des faits est impossible sans contredire l'attachement de cet ensemble de faits au modèle M(L).

Exemple: Le sous-ensemble de faits

$$SE(M(L_1)) = \{1 \in X, 2 \in X, 3 \in X, 1 \in Y, 2 \in Y, 1 \in F(1), 2 \in F(2)\}$$

donné dans l'exemple précédent n'est pas minimal:

le fait $3 \in X$ peut être supprimé. Mais le sous-ensemble

$$SE'(M(L_1)) = \{1 \in X, 2 \in X, 1 \in Y, 2 \in Y, 1 \in F(1), 2 \in F(2)\}$$

est un sous-ensemble minimal de faits de $M(L_1)$ puisque la cardinalité de Y doit être 2 (formule b) et que la fonction G est totale.

II.4.5 Base de faits selon un modèle

La **base de faits** selon un modèle contient l'ensemble des faits existants à un moment donné. Nous notons $\mathbf{BF_i}$ l'état de la base de faits à un moment **i**.

Ces faits sont des spécifications d'une application selon le modèle choisi.

II.4.6 Etapes et démarche d'une méthode

Un **état i** d'une spécification tel que nous l'avons introduit précédemment (cf. §II.2) est donc dépendant d'un **modèle** au sens précis des définitions ci-dessus. Une **démarche** dans le cadre d'une méthode peut être ainsi caractérisée par une séquence de modèles et une **étape** d'une démarche est la transition entre deux modèles successifs.

Exemple: Soit une méthode définie par:

le **langage** L:

$$X \xleftarrow{\ F\ } Y \qquad X \xleftarrow{\ h\ } Z$$
$$\qquad\quad G \qquad\qquad\quad H$$

et par les **modèles**:

modèle $M_1(L)$: f1: $card(X) \geq 1$

f2: $card(Y) \geq 1$

f3: $card(Z) = 0$

f4: $\forall x \in X \,.\, F(x) \neq \phi$

Les formules f1, f2 et f3 complètent la définition des ensembles X, Y et Z en imposant que X et Y soient non vides et que Z soit vide. La formule f4 est une restriction de la fonction F qui devient ainsi totale.

modèle $M_2(L)$: f1: $card(X) \geq 1$

f2: $card(Y) \geq 1$

f4: $\forall x \in X \,.\, F(x) \neq \phi$

f5: $card(Z) \geq 1$ (remplace la formule f3)

f6: $\forall y \in Y \,.\, G(y) \neq \phi$

f7: $\forall x \in X \,.\, h(x) = nil$

Les formules f5, f6 et f7 introduisent des restrictions complémentaires sur l'ensemble Z et sur les fonctions G et h.

modèle $M_3(L)$: f1: $card(X) \geq 1$

f2: $card(Y) \geq 1$

f4: $\forall x \in X \,.\, F(x) \neq \phi$

f6: $\forall y \in Y . G(y) \neq \phi$

f8: $\forall x \in X . h(x) \neq nil$ (remplace les formules f5 et f7)

L'évolution de la base de faits pour une application A pourrait être, par exemple, représentée par trois états:

états:

$BF_1 = \{x_1 \in X, x_2 \in X, y_1 \in Y, y_2 \in Y, y_1 \in F(x_1), y_1 \in F(x_2)\}$

$BF_2 = \{x_1 \in X, x_2 \in X, y_1 \in Y, y_2 \in Y, y_1 \in F(x_1), y_1 \in F(x_2), z_1 \in Z, y_2 \in F(x_1)\}$

$BF_3 = \{x_1 \in X, x_2 \in X, y_1 \in Y, y_2 \in Y, y_1 \in F(x_1), y_1 \in F(x_2), z_1 \in Z, y_2 \in F(x_1), z_1 = h(x_1), z_1 = h(x_2)\}$

Dans le cadre de cette méthode, une telle évolution serait conforme à une démarche en trois étapes pour satisfaire successivement les trois modèles ci-dessus.

une démarche:

étape1: création d'une nouvelle base de faits type BF_1 par l'introduction d'entités et de liens entre entités en respectant les énoncés du langage L et les formules du modèle $M_1(L)$;

étape2: transition d'une base de faits type BF_1 vers une base de faits type BF_2 qui respecte le modèle $M_2(L)$. Ajout des faits $z_1 \in Z$ et $y_2 \in F(x_1)$ pour satisfaire les formules f5, f6 et f7 sans contredire les formules précédentes f1, f2 et f4;

étape3: transition d'une base de faits type BF_2 vers une base de faits type BF_3 qui respecte le modèle $M_3(L)$. Ajout des faits $z_1 = h(x_1)$ et $z_1 = h(x_2)$ pour satisfaire la formule f8 en respectant les formules f1, f2, f4 et f6.

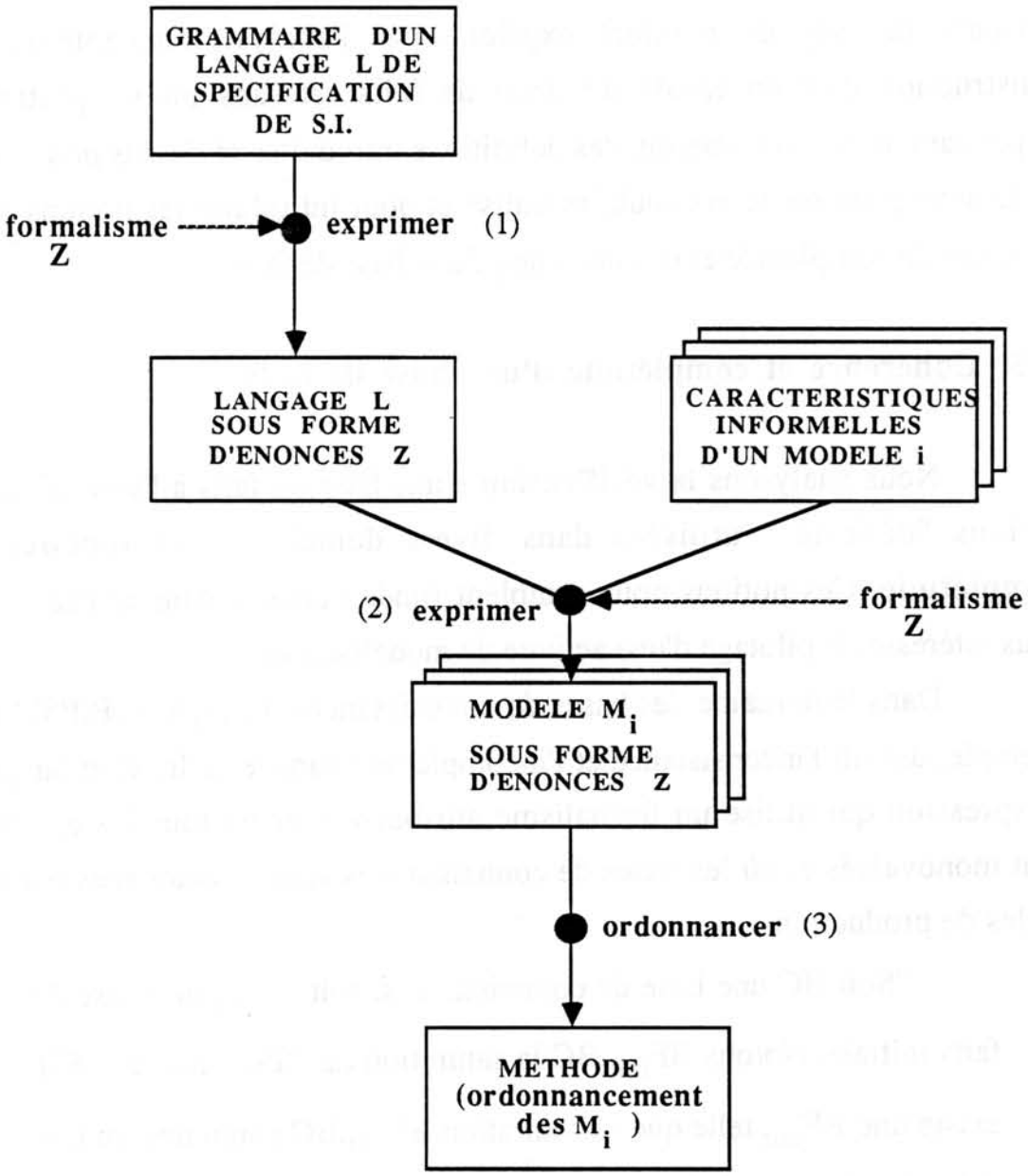
Ces définitions permettent de décrire des méthodes avec le "degré de finesse" souhaité en faisant varier le contenu et le nombre d'étapes.

Nous pouvons "affiner" une démarche générale introduite par un petit nombre d'étapes en définissant des sous-étapes intermédiaires.

Le formalisme Z permet donc de représenter successivement et de manière cohérente les aspects langage, modèle et démarche d'une méthode. Le contrôle de la spécification d'un S.I. selon une méthode ainsi décrite est assuré par la validation de formules.

La figure II.1 schématise la stratégie que nous proposons pour

décrire formellement une méthode.



(1) Ecriture d'énoncés Z à base d'ensembles d'entités et d'associations

(2) Ecriture d'énoncés Z à base d'ensembles d'entités, d'associations et de formules

(3) Organisation des modèles sous forme de démarche

Figure II.1: Schéma d'une stratégie générale pour décrire une méthode

Notons que dans une application réelle nous ne considérons que des bases de faits finies: nous nous limitons à valider, à un moment donné, une base concrète de faits conformément à un modèle. L'introduction de nouvelles entités ou de nouveaux liens entre entités (faits élémentaires) est toujours réalisée de manière explicite par l'analyste-concepteur. La construction d'un ensemble maximal de faits est sans intérêt pratique. Cependant, nous avons besoin des définitions introduites ci-dessus pour lever toute ambiguïté sur le vocabulaire utilisé et pour introduire les notions plus délicates de complétude et de cohérence d'une base de faits.

II.5 Cohérence et complétude d'une base de faits

Nous analysons la **vérification** d'une base de faits à l'aide de deux notions "délicates" utilisées dans divers domaines: **cohérence** et **complétude**. Ces notions nous semblent fondamentales dans le cadre qui nous intéresse, le pilotage d'une activité de modélisation.

Dans le domaine des bases de connaissances, E. Pipard [PIP87] par exemple, définit l'inconsistance et l'incomplétude dans le cadre d'un langage d'expression qui utilise un formalisme attribut-valeur où tous les attributs sont monovalués et où les bases de connaissances sont caractérisées par des règles de production.

> "Soit BC une base de connaissances, soit BF_{init} une base de faits initiale. Notons BF_{init}.BC la saturation de BF_{init} par BC. S'il existe une BF_{init} telle que sa saturation BF_{init}.BC contienne un fait et la négation de celui-ci alors nous dirons que BC est inconsistante; si pour toute BF_{init} les saturations BF_{init}.BC ne contiennent pas un fait pourtant déductible alors nous dirons que BC est incomplète."

Dans cette définition, la saturation d'une base de faits par une base de connaissances correspond au déclenchement de toutes les règles applicables, c'est-à-dire dont les prémisses sont vraies. E. Pipard définit la consistance et

la complétude d'une base de connaissances et non d'une base de faits. En effet, il considère que l'utilisateur connaît la situation du monde réel à fournir au système et donc, lors d'une session, il n'introduira pas de bases de faits inconsistantes. Ainsi, il s'intéresse surtout aux connaissances déductives.

M. Ayel [AYE87] n'aborde que le problème de la cohérence d'une base de connaissances. Il considère qu'une base de connaissances est incomplète par nature puisqu'elle traduit une expertise souvent partielle. Il considère qu'une base de faits est cohérente

"si l'ensemble de faits saturé par des mécanismes de complétion, $EF_{saturé}$, est un ensemble de faits cohérent vis à vis du modèle conceptuel

et si toutes les contraintes d'intégrité sont satisfaites sur cet ensemble $EF_{saturé}$."

Dans cette définition, les mécanismes de complétion correspondent à des simplifications pour la saisie des faits, ils expriment sous forme déductive des relations structurelles entre attributs.

Exemple: SI A est père de B ALORS B est fils de A

Le modèle conceptuel correspond à un ensemble de classes d'objets où chaque classe est définie par un nom, par une liste d'attributs intrinsèques et une liste de leurs propriétés et par des relations d'exclusion entre les attributs. Une contrainte d'intégrité établit que la conjonction de certains faits constitue une incohérence.

Exemple:
```
SI longueur de X est Y
   ET SI largeur de X est Z
   ET SI Y < Z
      ALORS incohérence
```

Un fait est un triplet sans variable (objet, attribut, valeur).

Exemple: (bateau, couleur, blanc)

Dans le domaine de la modélisation de S.I. il y a une notion plus ou moins intuitive de cohérence et de complétude. Une spécification est dite cohérente si elle ne présente pas de contradictions. Une spécification est dite complète s'il n'y a pas omission d'éléments vis à vis de la définition de complétude attachée au modèle. Nous proposons donc les définitions précises suivantes, mieux adaptées à nos objectifs.

Cohérence et complétude dans notre étude

Les définitions relatives à la description précise d'une démarche (cf. §II.4) peuvent être complétées par l'introduction des notions de cohérence et de complétude d'une base de faits BF selon un modèle M(L), notée BF(M(L)).

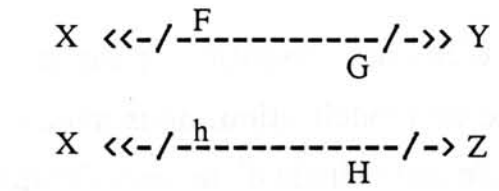
Nous définissons les notions d'incohérence et d'incomplétude pour mettre en évidence qu'un bon système doit, après avoir décelé des situations d'incohérence ou d'incomplétude, nous informer sur l'activité à réaliser pour passer d'une situation d'incohérence à une situation de cohérence ou d'une situation d'incomplétude à une situation de complétude.

• une base de faits BF(M(L)) est **incohérente** SSi il existe des formules du modèle M(L) en échec qui ne peuvent être satisfaites que par la suppression de faits existants.

• une base de faits BF(M(L)) est **incomplète** SSi il existe des formules du modèle M(L) en échec qui peuvent être satisfaites par l'ajout de nouveaux faits. Une BF(M(L)) complète contient implicitement un sous-ensemble minimal de M(L):

$$BF(M(L))\text{"complète"} \supseteq SE_{min}(M(L))$$

Exemple: Soit le langage L:



et le modèle $M_1(L)$:

f1: $card(X) \geq 1$

f2: $card(Y) \geq 1$

f3: $\forall y \in Y . G(y) \neq \phi$

f4: $\forall x \in X . (F(x)=\phi \vee h(x)=nil)$

Un sous-ensemble minimal de $M_1(L)$ est composé par trois faits sous la forme

$SE_{min}(M_1(L)) = \{x \in X,\ y \in Y,\ x \in G(y)\}$

Soit $BF_1 = \{x_1 \in X, y_1 \in Y, y_2 \in Y, z_1 \in Z, x_1 \in G(y_1), h(x_1)=z_1\}$

BF_1 est incomplète (f3 non satisfaite: $G(y_2)=\phi$) et incohérente (f4 non satisfaite: $h(x_1) \neq nil$ et $F(x_1) \neq \phi$).

Soit $BF_2 = BF_1 - \{h(x_1)=z_1\} = \{x_1 \in X, y_1 \in Y, y_2 \in Y, z_1 \in Z, x_1 \in G(y_1)\}$

BF_2 est incomplète (f3 non satisfaite: $G(y_2)=\phi$) et cohérente.

Soit $BF_3 = BF_1 \cup \{x_1 \in G(y_2)\} = \{x_1 \in X, y_1 \in Y, y_2 \in Y, z_1 \in Z, x_1 \in G(y_1), x_1 \in G(y_2), h(x_1)=z_1\}$

BF_3 est complète et incohérente (f4 non satisfaite: $h(x_1) \neq nil$ et $F(x_1) \neq \phi$).

Soit $BF_4 = BF_3 - \{h(x_1)=z_1\} = \{x_1 \in X, y_1 \in Y, y_2 \in Y, z_1 \in Z, x_1 \in G(y_1), x_1 \in G(y_2)\}$

BF_4 est complète et cohérente.

Soit $BF_5 = BF_2 \cup \{x_1 \in G(y_2)\} = \{x_1 \in X, y_1 \in Y, y_2 \in Y, z_1 \in Z, x_1 \in G(y_1), x_1 \in G(y_2)\}$

BF_5 est complète et cohérente.

Notons que BF5 = BF4: cet état est obtenu par des séquences différentes de spécification.

II.6 Conclusion

Après avoir donné les éléments essentiels pour assurer une description formelle d'une méthode de modélisation, nous avons défini les notions de contrôle de cohérence et de complétude d'une spécification.

Ces définitions portent sur la vérification de la cohérence et de la complétude d'une spécification par rapport à la définition formelle d'un modèle.

Elles ne permettent pas de mesurer l'adéquation d'une spécification relativement à la réalité à modéliser, en particulier à cause de l'absence actuelle de formalisation du domaine de l'application (cf. §I.3.2.2). Les approches dites de prototypage permettent d'aborder cet aspect [BOA84, BUD84].

Dans le chapitre suivant nous montrons comment trouver et classer les formules (règles) à associer à un modèle.

CHAPITRE III

TYPOLOGIE DES REGLES DE VERIFICATION

TYPOLOGIE DES REGLES DE VERIFICATION

III.1 Introduction

La représentation formelle d'une méthode d'analyse et de conception de S.I. s'appuie sur des règles de définition des modèles (cf. II.4). Dans le cadre d'un pilotage d'une modélisation, ces règles sont vues comme des contraintes d'intégrité sur des spécifications et sont appelées **règles élémentaires de gestion.**

L'objectif principal de ce chapitre est l'introduction de différentes catégories de règles et de la manière de les décrire. Cette caractérisation déterminée par l'utilité et la portée de chaque règle associera descriptions formelles et descriptions informelles, prémisses d'un logiciel autodocumenté.

Par un exemple, nous montrons comment cette caractérisation des règles permet d'en faciliter leurs "découverte" dans le cadre d'un sous-ensemble du langage DSL/IDA.

Dans une dernière partie, nous étudions les dépendances entre règles.

III.2 Typologie des règles élémentaires de gestion

Ces règles sont conformes à certains types généraux qui peuvent être sommairement classés en deux grandes catégories: règles de vérification de cohérence et règles de vérification de complétude. Nous proposons de partitionner ces deux catégories en dix-sept types élémentaires de règles qui permettent de **guider la découverte des contraintes d'intégrité** à associer à un modèle.

Trois critères ou points de vue sont sous-jacents à cette caractérisation:

• la **nature de la règle**, c'est-à-dire si c'est une règle de contrôle de cohérence ou de complétude par rapport au modèle considéré (cf. §II.5, §III.4 et §III.5);

• le **type d'action à entreprendre** pour satisfaire une règle en échec: supprimer, modifier ou ajouter des spécifications (cf. "recommandation" §III.3);

• la **portée de la règle** par rapport à la syntaxe d'expression du langage de spécification (cf. "sous-schéma" §III.3): traiter les ensembles avant les associations et les groupes d'association avant les associations particulières.

III.3 Contenu de l'énoncé d'une règle

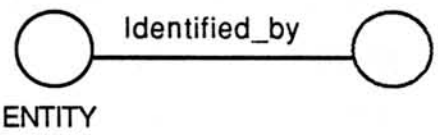
Chaque règle est décrite selon cinq composantes. Gattelier et Bertinchamps [GAT86] décrivent les contraintes d'intégrité dans le domaine de la conception des bases de données selon trois composantes: signification de la contrainte en langage naturel, diagnostic commentant le problème détecté et recommandation pour expliquer comment corriger ou compléter les spécifications. Nous proposons deux composantes supplémentaires pour l'énoncé de chaque règle: un schéma caractéristique et une expression formelle.

Exemple:

• expression informelle:

Chaque entité doit avoir au moins un identifiant.

• sous-schéma:



• expression formelle:

$\forall e \in \text{ENTITY} \;.\; \text{Identified\_by}(e) \neq \phi$

• diagnostic:

Absence d'identifiant de l'entité "X".

• recommandation:

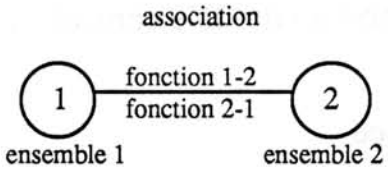
Il faut utiliser au moins une clause "IDENTIFIED BY ..." pour spécifier un identifiant de l'entité.

L'écriture de chaque phrase en langue naturelle (expression informelle, diagnostic ou recommandation) est réalisée selon une syntaxe générale.

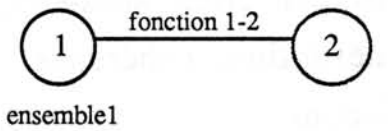
Conventions:

- **sous-schéma**: représentation graphique sommaire d'un sous-schéma pour localiser la portée de la règle. Un rond représente un ensemble d'entité et un trait représente une association.

Exemple:



Ce sous-schéma est formé de deux ensembles d'entités liés par une association. Les expressions formelles des règles associées à ce sous-schéma n'utiliseront que les noms des ensembles et des fonctions introduits graphiquement.



Par ce deuxième sous-schéma on indique que le nom de l'ensemble2 n'est pas explicitement utilisé dans l'expression formelle.

- **expression formelle**: forme générale de la description formelle d'une règle de ce type. Evidemment plusieurs expressions formelles pour une même règle sont possibles mais l'expression présentée a été jugée la plus adéquate. Les notations utilisées sont celles du chapitre II. Malgré l'utilisation conjointe de fonctions monovaluées et de fonctions multivaluées, nous ne décrivons souvent les règles qu'avec la notation des fonctions multivaluées pour ne pas alourdir le texte avec deux représentations assez semblables. Pour les exemples, par contre, la notation utilisée correspond au type de la fonction considérée. Par convention on admettra qu'une entité définie est équivalente à un ensemble à un élément et une entité indéfinie correspond à l'ensemble vide.
- **expression informelle**: forme générale de l'écriture en langage naturel d'une règle de ce type.
- **diagnostic**: forme générale de l'expression du message à communiquer à l'utilisateur pour expliquer le problème détecté.

- **recommandation**: forme générale de l'expression en langage naturel du message à donner, pour expliquer à l'utilisateur comment corriger ou compléter les spécifications par rapport au problème détecté.

Notons que les composantes diagnostic et recommandation sont dépendantes du mode d'activation de la règle et du type d'interface utilisateur choisi. Dans ce chapitre, pour la facilité de l'exposé, nous considérons une interface traditionnelle purement textuelle où les règles sont déclenchées à la demande de l'utilisateur qui veut s'assurer, à un moment qu'il choisit, que la spécification est cohérente et complète par rapport au modèle considéré.

III.4 Règles de vérification de cohérence

Nous avons rassemblé ces règles dans sept groupes: ensemble interdit, connectivité maximale d'une association, association interdite, sous-ensemble interdit dans une association, unicité de définition, cohérence d'une décomposition et autres restrictions inter-associations.

Ces règles servent à déceler des incohérences ou des contradictions des spécifications et conduisent généralement à supprimer ou à modifier des entités ou des liaisons entre entités.

III.4.1 Ensemble interdit

Pour décrire une démarche il faut définir l'activité à réaliser à chaque étape particulière. Par exemple, il faut définir quels sont les ensembles à spécifier à chaque étape. Mais, il est nécessaire aussi de formuler des règles pour interdire l'utilisation de certains ensembles pendant quelques étapes de la démarche ou même pendant toute la modélisation.

- sous-schéma:

• expression formelle:
card(<ensemble>) = 0

Ex: card(ELEMENT) = 0

• expression informelle:
- la spécification de(s) <forme nominale caractérisant l'ensemble> est interdite pendant <nom de l'étape>.

Ex: la spécification d'éléments est interdite pendant l'étude d'opportunité.

• diagnostic:
- <forme nominale caractérisant l'ensemble> "<entité>" a été décrit (la spécification <forme nominale caractérisant l'ensemble> est interdite pendant <étape>).

Ex: l'élément "X" a été décrit (la spécification d'éléments est interdite pendant l'étude d'opportunité).

• recommandation:
- supprimer la spécification de <forme nominale caractérisant l'ensemble> "<entité>".

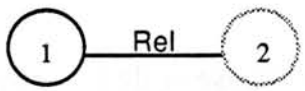
Ex: supprimer la spécification de l'élément "X".

Sur cet aspect de limitation d'utilisation d'ensembles, nous ne présentons qu'un exemple qui correspond à l'interdiction d'utilisation d'un ensemble parce que nous n'avons pas trouvé d'exemple réel sur la limitation de la cardinalité d'un ensemble à un nombre fini différent de zéro.

III.4.2 Connectivité maximale d'une association

Pour des raisons d'ordre méthodologique il peut être nécessaire de limiter le nombre maximal de liaisons selon une association par rapport à un ensemble.

• sous-schéma:



• expression formelle:
$\forall e \in$ <ensemble1> . card(Rel(e)) $\leq x$ ($x \in$ ENTIER)

Ex: $\forall p \in$ PROCESS . card(Subparts_are(p)) ≤ 10

• expression informelle:
- <forme nominale caractérisant l'ensemble1> ne peut pas avoir plus de <entier> <forme nominale caractérisant l'association>.

Ex: un processus ne peut pas avoir plus de 10 processus fils.

• diagnostic:
- <forme nominale caractérisant l'ensemble1> "<entité1>" a plus de <entier> <forme nominale caractérisant l'association>.

Ex: le processus "X" a plus de 10 processus fils.

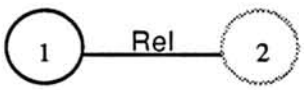
• recommandation:
- supprimer des spécifications pour que <forme nominale caractérisant l'ensemble1> "<entité1>" ait au plus <entier> <forme nominale caractérisant l'association>.

Ex: supprimer des spécifications pour que le processus "X" ait au plus 10 processus fils.

III.4.3 Association interdite

La restriction la plus forte sur l'utilisation d'associations est l'interdiction complète. Une règle de ce type peut être utilisée pendant toute la modélisation ou seulement dans certaines étapes de la démarche.

• sous-schéma:



• expression formelle:
$\forall e \in$ <ensemble1>. Rel(e) = ϕ

Ex: $\forall m \in$ MESSAGE . layout(m)=nil

• expression informelle:
- la spécification <forme nominale caractérisant l'ensemble1> <forme nominale caractérisant l'association> est interdite.

Ex: la spécification de messages en termes de présentation de contenu est interdite.

• diagnostic:
- <forme nominale caractérisant l'ensemble1> "<entité1>" a été <forme nominale caractérisant l'association>.

Ex: le message "X" a été spécifié en termes de présentation du contenu.

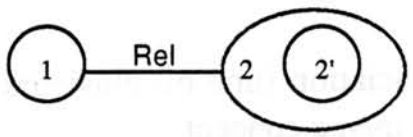
• recommandation:
- supprimer la clause "<association>" correspondant <forme nominale caractérisant l'ensemble> "<entité>".

Ex: supprimer la clause "LAYOUT; ..." correspondant au message "X".

III.4.4 Sous-ensemble interdit dans une association

Dans le cas d'associations où l'un des ensembles est construit par union ensembliste, il est nécessaire éventuellement de permettre l'application de règles pour interdire certaines formes d'usage de cette association vis à vis de sous-ensembles particuliers.

• sous-schéma:



"union d'ensembles"

• expression formelle:
$\forall e_1 \in$ <ensemble1> . $\forall e'_2 \in$ <ensemble2'> . $\neg(e'_2 \in Rel(e_1))$

Ex: $\forall e \in ENTITY . \forall n \in (SYS\text{-}PAR \cup ENTIER)^{\circ} . \forall g \in GROUP .$
$\neg(g \in Consists\_of(e,n))$

Nous pouvons remarquer que dans cet exemple:
ensemble1 = ENTITY x (SYS-PAR ∪ INTEGER)°
et que GROUP est un sous-ensemble de l'union ensemble2

• expression informelle:
- <forme nominale caractérisant l'ensemble1> ne peut pas <forme verbale caractérisant l'association> <forme nominale caractérisant l'ensemble2'>.

Ex: une entité ne peut pas contenir de groupes.

• diagnostic:
- <forme nominale caractérisant l'ensemble1> "<entité1>" <forme verbale caractérisant l'association> <forme nominale caractérisant l'ensemble2'>.

Ex: l'entité "X" contient des groupes.

• recommandation:
- supprimer le(s) <forme nominale caractérisant l'ensemble2'> <forme nominale caractérisant l'ensemble1> "<entité1>" (<alternative>).

Ex: supprimer le(s) groupe(s) de l'entité "X" (en le(s) transformant éventuellement en entité(s)).

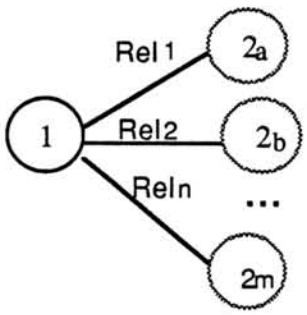
III.4.5 Unicité de définition

Dans ce cadre, nous distinguons trois catégories de règles selon que l'on traite une ou plusieurs associations et que, dans le cas de plusieurs associations, on se limite à une seule liaison ou à des liaisons selon une seule association.

III.4.5.1 Unicité de mode de définition

Il s'agit d'utiliser une seule association (une ou plusieurs liaisons) pour une entité donnée pour exprimer un même concept.

• sous-schéma:



• expression formelle:
$\forall e \in$ <ensemble1> . $(Rel_1(e)=\phi \vee Rel_2(e)=\phi)$ (entre 2 associations)

Ex1: $\forall e \in$ ELEMENT. (Domain_of_values(e)=ϕ $\vee$ same_domain(e)= nil)

$\forall e \in$ <ensemble1>. (entre plusieurs associations)

$((Rel_1(e)=\phi \wedge Rel_2(e)=\phi \wedge Rel_3(e)=\phi \wedge ... \wedge Rel_n(e)=\phi) \vee$
$(Rel_1(e)\neq\phi \wedge Rel_2(e)=\phi \wedge Rel_3(e)=\phi \wedge ... \wedge Rel_n(e)=\phi) \vee$
$(Rel_1(e)=\phi \wedge Rel_2(e)\neq\phi \wedge Rel_3(e)=\phi \wedge ... \wedge Rel_n(e)=\phi) \vee$
$(Rel_1(e)=\phi \wedge Rel_2(e)=\phi \wedge Rel_3(e)\neq\phi \wedge ... \wedge Rel_n(e)=\phi) \vee$
.....
$(Rel_1(e)=\phi \wedge Rel_2(e)=\phi \wedge ... \wedge Rel_{n-1}(e)=\phi \wedge Rel_n(e)\neq\phi))$

Ex2: $\forall m \in$ MESSAGE . $\forall n \in$ (SYS-PAR $\cup$ ENTIER)° .
((Consists_of(m,n)=ϕ $\wedge$ includes_content_of(m)=nil$\wedge$ Subsets_are(m)=ϕ) $\vee$
(Consists_of(m,n)$\neq\phi$ $\wedge$ includes_content_of(m)=nil $\wedge$ Subsets_are(m)=ϕ) $\vee$
(Consists_of(m,n)=ϕ $\wedge$ includes_content_of(m)$\neq$nil $\wedge$ Subsets_are(m)=ϕ) $\vee$
(Consists_of(m,n)=ϕ $\wedge$ includes_content_of(m)=nil $\wedge$ Subsets_are(m)$\neq\phi$))

• expression informelle:
- <forme nominale caractérisant l'ensemble1> ne peut être <forme nominale caractérisant les associations> que d'une seule manière.

Ex1: un élément ne peut être spécifié en termes de domaine que d'une seule manière.

Ex2: un message ne peut être spécifié que d'une seule manière.

• diagnostic:
- <forme nominale caractérisant l'ensemble1> "<entité1>" a été <forme nominale caractérisant les associations> de plusieurs manières différentes.

Ex1: l'élément "X" a été spécifié en termes de domaine de plusieurs manières différentes.

Ex2: le message "Y" a été spécifié de plusieurs manières différentes.

• recommandation:
- modifier les spécifications <forme nominale caractérisant l'ensemble1> "<entité1>" pour n'utiliser qu'un(e) seul(e) <forme nominale caractérisant les associations>.

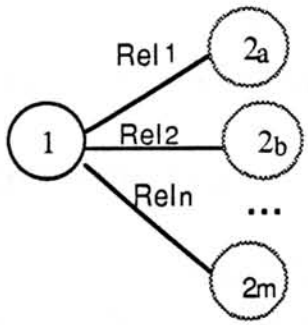
Ex1: modifier les spécifications de l'élément "X" pour n'utiliser qu'une seule manière d'en décrire le domaine.

Ex2: modifier les spécifications du message "Y" pour n'utiliser qu'une seule manière d'en décrire le contenu.

III.4.5.2 Unicité absolue de définition

Une contrainte d'utilisation de plusieurs associations pour représenter un même concept peut imposer qu'une entité ait au plus une liaison selon une seule association selon ce concept.

• sous-schéma:



• expression formelle:

$\forall e \in$ <ensemble1> . $(card(Rel_1(e)) + card(Rel_2(e)) + ... + card(Rel_n(e)) \leq 1)$

Ex: $\forall p \in$ PROCESS.

(card(triggered_by(p)) + card(triggered_by_generation(p)) +
card(triggered_by_realization(p)) ≤ 1)

• expression informelle:

- <forme nominale caractérisant les associations> <forme nominale caractérisant l'ensemble1> doit être unique.

Ex: le déclenchement d'un processus doit être unique.

• diagnostic:

- <forme nominale caractérisant les associations> <forme nominale caractérisant l'ensemble1> "<entité1>" n'est pas unique.

Ex: le déclenchement du processus "X" n'est pas unique.

• recommandation:

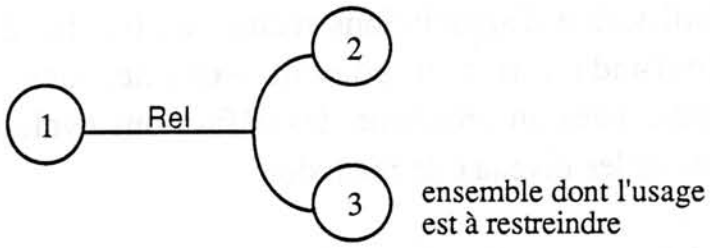
- modifier les spécifications pour obtenir <forme nominale caractérisant les associations> unique.

Ex: modifier les spécifications pour obtenir un déclenchement unique.

III.4.5.3 Unicité d'usage d'un ensemble via une association

Dans le cas d'une association entre plus de deux ensembles il est nécessaire éventuellement de formuler des règles pour s'assurer de l'unicité de la définition d'un ensemble par rapport à d'autres ensembles.

• sous-schéma:



• expression formelle:

$\forall e_1 \in$ <ensemble1> . $\forall e_2 \in$ <ensemble2> . $\forall e_{31}, e_{32} \in$ <ensemble3> .

$$(((e_2, e_{31}) \in Rel(e_1) \wedge (e_2, e_{32}) \in Rel(e_1)) \Rightarrow (e_{31} = e_{32}))$$

Ex: $\forall p \in PROCESSOR . \forall x \in PROCESS^{\circ} . \forall c_1, c_2 \in CALENDAR .$

$$(((c_1, x) \in Available\_during(p) \wedge (c_2, x) \in Available\_during(p)) \Rightarrow c1 = c2)$$

• expression informelle:

- <forme nominale caractérisant l'ensemble3> <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble1> doit être unique <forme nominale caractérisant l'ensemble2>.

Ex: le calendrier d'utilisation d'un processeur doit être unique pour chaque processus.

• diagnostic:

- <forme nominale caractérisant l'ensemble3> <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble1> "<entité1>" n'est pas unique <forme nominale caractérisant l'ensemble2> "<entité2>" .

Ex: le calendrier d'utilisation du processeur "X" n'est pas unique pour le processus "Y".

• recommandation:

- modifier les spécifications pour que <forme nominale caractérisant l'ensemble3> <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble1> "<entité1>" soit unique <forme nominale caractérisant l'ensemble2> "<entité2>" .

Ex: modifier les spécifications pour que le calendrier d'utilisation du processeur "X" soit unique pour le processus "Y".

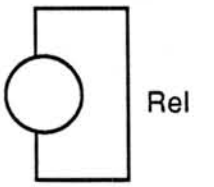
III.4.6 Contrôle d'une décomposition

Lors de l'utilisation d'associations récursives (où les 2 ensembles arguments sont confondus) il faut pouvoir exprimer des contraintes spécifiques éventuelles liées au problème des définitions cycliques ou à la nécessité de différencier les niveaux de récursion.

III.4.6.1 Définition circulaire interdite

Une forme de restriction souvent utilisée dans le cas d'associations récursives est l'interdiction des cycles.

• sous-schéma:



"association récursive"

• expression formelle:

$\forall e_1, e_2 \in$ <ensemble> . $\forall n_1, n_2 \in ENTIER^+ . \neg(e_1 \in Rel^{n1}(e_2) \wedge e_2 \in Rel^{n2}(e_1))$

Ex1: $\forall e_1, e_2 \in ELEMENT . \forall n_1, n_2 \in ENTIER^+ . \neg(e_1 = same\_domain^{n1}(e_2) \wedge e_2 = same\_domain^{n2}(e_1))$

Ex2: $\forall p_1, p_2 \in PROCESS . \forall n_1, n_2 \in ENTIER^+ . \neg(p_1 \in Subparts\_are^{n1}(p_2) \wedge p_2 \in Subparts\_are^{n2}(p_1))$

NB: $Rel^i(e) = \underbrace{Rel(Rel(\ldots(Rel(e))\ldots))}_{i \text{ fois}}$

• expression informelle:

- <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble> ne doit pas être défini(e) circulairement.

Ex1: l'équivalence de domaines des éléments ne doit pas être définie circulairement.

Ex2: la hiérarchie des traitements ne doit pas être définie circulairement.

• diagnostic:
- définition circulaire <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble> "<entité1>" et "<entité2>".

Ex1: définition circulaire entre les domaines des éléments "X" et "Y".

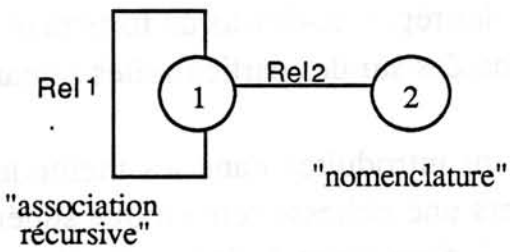
Ex2: définition circulaire de décomposition des processus "X" et "Y".

• recommandation:
- modifier les spécifications de manière à éliminer la définition circulaire.

III.4.6.2 Contrôle de décomposition selon une nomenclature

Dans le cas de l'existence d'une nomenclature standard associée à chaque entité d'un ensemble décomposable selon une association récursive, il faut pouvoir exprimer des contraintes pour assurer la cohérence de la décomposition de ces entités par rapport à la nomenclature.

• sous-schéma:



• expression formelle: $(n_{21},n_{22} \in Rel2(<ensemble1>))$

$\forall e,e' \in$ <ensemble1> .
((e ∈ Rel1 (e') ∧ n_{21} ∈ Rel2(e)) ⇒
(n_{22} ∈ Rel2(e') ∨ Rel2(e') = ϕ))

Ex: ∀p,p' ∈ PROCESS .
((p = part_of(p') ∧ niveau_de(p) = 'projet') ⇒
(niveau_de(p') = 'application' ∨ niveau_de(p') = nil))

• expression informelle:
- chaque <ensemble1> <forme nominale caractérisant l'association1> <forme nominale caractérisant l'association2> <occurrence n_{21} de la nomenclature> doit être <forme nominale caractérisant l'association2> <occurrence n_{22} de la nomenclature>.

Ex: chaque processus fils d'un processus de niveau projet doit être de niveau application .

• diagnostic:

- <forme nominale caractérisant l'ensemble1> <"entité1>" <forme nominale caractérisant l'association2> <occurrence n_{21} de la nomenclature> ne peut pas avoir comme <forme nominale caractérisant l'association1> <forme nominale caractérisant l'ensemble1> "<entité2>" <forme nominale caractérisant l'association2> "<occurrence n_{22} de la nomenclature>" .

Ex: le processus "X" de niveau projet ne peut pas avoir comme fils le processus "W" de niveau "Z".

• recommandation:

- modifier les spécifications pour que <forme nominale caractérisant l'association2> <forme nominale caractérisant l'ensemble1> "<entité1>" et "<entité2>" soient cohérents (cohérentes).

Ex: modifier les spécifications pour que les niveaux de processus "X" et "W" soient cohérents.

III.4.7 Autres restrictions inter-associations

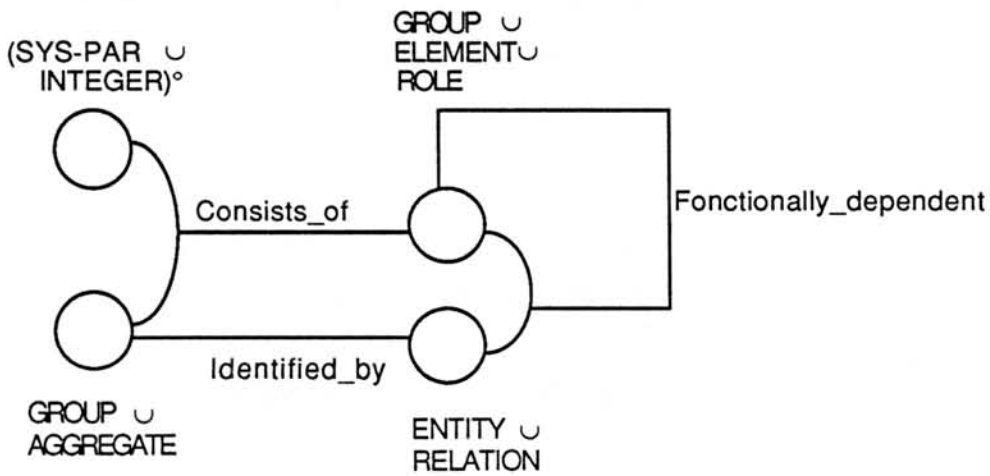
Nous pouvons qualifier les règles ci-dessus de fortement syntaxiques par opposition à d'autres règles basées sur des particularités sémantiques d'un langage de spécification.

Ces règles, nouvellement introduites dans les méthodes, sont très importantes puisqu'elles vont vers une richesse sémantique supérieure. Elles relèvent d'un "style d'abstraction", d'une sorte de "normalisation" du schéma. A défaut d'une meilleure prise en compte du réel on essaie d'obtenir une meilleure forme canonique.

Pour cette classe de règles délicates à caractériser, nous nous limitons à donner deux exemples.

Exemple1:

• sous-schéma:



- expression formelle:

  $\forall a \in (GROUP \cup AGGREGATE)\ .\ \forall g_1 \neq g_2 \in (GROUP \cup ELEMENT \cup ROLE)\ .$

  $\forall e \in (ENTITY \cup RELATION)\ .\ \forall n_1, n_2 \in (SYS\text{-}PAR \cup INTEGER)^{\circ}\ .$

  $\neg\ (a \in Identified\_by(e)\ \wedge\ g_1 \in Consists\_of(a, n_1)\ \wedge\ g_2 \in Consists\_of(a, n_2)\ \wedge$

  $(g_1, e) \in fonctionally\_dependent(g_2))$

- expression informelle:

  il ne doit pas y avoir de dépendance fonctionnelle entre attributs d'un groupe qui identifie une entité ou une association.

- diagnostic:

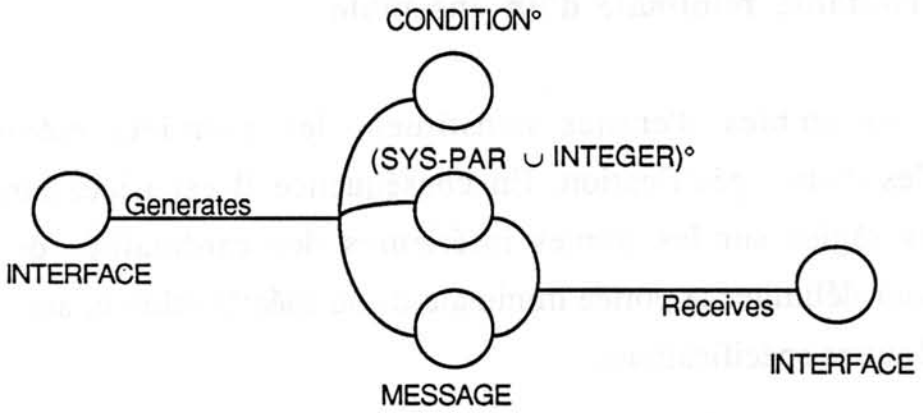
  il y a une dépendance fonctionnelle entre les attributs "X" et "Y" de l'identifiant de "Z".

- recommandation:

  modifier les spécifications pour que le groupe identifiant soit sans dépendance fonctionnelle entre ses attributs.

Exemple2:

- sous-schéma:



- expression formelle:

  $\forall m \in MESSAGE\ .\ \forall i_1, i_2 \in INTERFACE\ .\ \forall n_1, n_2 \in (SYS\text{-}PAR \cup INTEGER)^{\circ}\ .$

  $\forall c \in CONDITION^{\circ}\ .$

  $\neg\ ((m, n_1, c) \in Generates(i_1)\ \wedge\ (m, n_2) \in Receives(i_2))$

- expression informelle:

  un message ne peut pas être à la fois généré et reçu par des interfaces.

- diagnostic:

  le message "X" est généré par l'interface "Y" et reçu par l'interface "Z".

- recommandation:

  modifier les spécifications pour que le message ne soit pas généré et reçu par des interfaces.

III.5 Règles de vérification de complétude

Une spécification est jugée complète lorsque, par rapport à un modèle de spécification donné, il ne manque aucune définition obligatoire d'entités ou de liaisons. Nous rassemblons les différentes règles de vérification de complétude de spécifications dans six groupes: cardinalité minimale d'un ensemble, connectivité minimale d'un groupe d'associations, connectivité minimale d'une association, ensemble obligatoire dans une association composite, contrôle d'une hiérarchie et autres dépendances inter-associations.

A moins d'une transformation profonde des spécifications, ces règles conduisent en général à ajouter de nouvelles entités ou de nouvelles liaisons entre entités.

III.5.1 Cardinalité minimale d'un ensemble

Les ensembles d'entités constituent les premiers éléments indispensables d'une spécification. En conséquence il est nécessaire de formuler des règles sur les bornes inférieures des cardinalités de ces ensembles pour délimiter la portée minimale d'une spécification et autoriser par la suite d'autres spécifications.

• sous-schéma:

• expression formelle :

card(<ensemble>) ≥ x (x ∈ ENTIER^{+})

Ex1: card(ENTITY) ≥ 1

Ex2: card(PROCESS) ≥ 2

• expression informelle:

- il doit y avoir au moins <entier> <forme nominale caractérisant l'ensemble> défini(e)(s).

Ex1: il doit y avoir au moins une entité définie.

Ex2: il doit y avoir au moins deux processus définis.

• diagnostic:

- absence de spécification <forme nominale caractérisant l'ensemble>.

Ex1: absence de spécification d'entités.

Ex2': absence de spécification de processus.

- il y a seulement <entier> <forme nominale caractérisant l'ensemble> spécifié(e)(s).

Ex2": il y a seulement un processus spécifié.

• recommandation:

- il faut utiliser au moins <entier> clause(s) "DEFINE <ensemble> ..." <explication de la finalité de la clause>.

Ex1: il faut utiliser au moins une clause "DEFINE ENTITY ..." pour caractériser les entités du système.

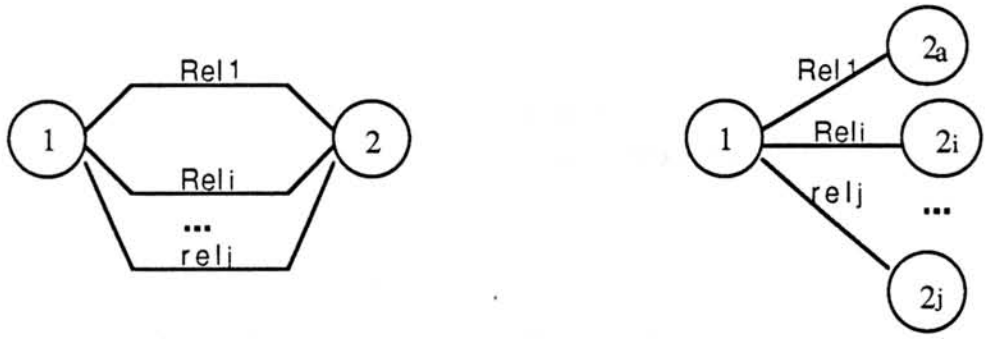
Ex2': il faut utiliser au moins une clause "DEFINE PROCESS ..." pour caractériser les traitements du système.

Ex2": il faut utiliser au moins deux clauses "DEFINE PROCESS ..." pour caractériser les traitements du système.

III.5.2 Connectivité minimale d'un groupe d'associations

Un même concept général peut être représenté par plusieurs associations spécifiques. Par exemple, l'"utilisation" de la mémoire du S.I. peut être caractérisée par des associations telles que "modifier", "lire" et "ajouter". Il est nécessaire de pouvoir formuler des règles sur l'usage indispensable d'au moins une association parmi un groupe d'associations pour indiquer que le concept général est pris en compte par au moins une association spécifique.

• sous-schéma:



• expression formelle:

$\forall e \in$ <ensemble1>. $(Rel_1(e)\neq\phi \vee ... Rel_i(e)\neq\phi \vee ... \vee rel_j(e) \neq nil \vee ...)$

Ex1: $\forall p \in$ PROCESS.

(Uses(p)≠ϕ ∨ Derives(p)≠ϕ ∨ Adds(p)≠ϕ ∨ References(p)≠ϕ ∨

Modifies(p)≠ϕ)

Ex2: $\forall e \in$ ELEMENT. $\exists r \in$ (RELATION ∪ ENTITY).

$\exists s \in$ (SET ∪ ENTITY ∪ RELATION ∪ GROUP ∪ ELEMENT)°.

$\exists n \in$ (SYS-PAR ∪ ENTIER)° .

(Used_by(e,s) ≠ ϕ ∨ Derived_by(e,s) ≠ ϕ ∨ Referenced_by(r,e,n)≠ ϕ ∨

Modified_by(r,e,n)≠ϕ)

Ex3: $\forall m \in$ MESSAGE . $\exists n \in$ (SYS-PAR ∪ ENTIER)° .

(Subsets_are(m) ≠ ϕ ∨ Consists_of(m,n) ≠ ϕ ∨

includes_content_of(m) ≠ nil)

• expression informelle:

- chaque <ensemble1> doit <forme verbale caractérisant les types d'association> au moins <forme nominale caractérisant l'ensemble2> ...

Ex1: chaque processus doit utiliser au moins un élément de la mémoire du SI.

Ex2: chaque élément doit être utilisé par au moins un processus .

- chaque <ensemble1> doit <forme verbale caractérisant les types d'association>.

Ex3: chaque message doit être décomposé.

• diagnostic:

- absence <forme nominale caractérisant les types d'association> <forme nominale caractérisant l'ensemble2> <forme nominale caractérisant l'ensemble1> "<entité1>".

Ex1: absence d'utilisation de la mémoire du SI par le processus "X".

Ex2: absence d'utilisation par un processus de l'élément "X".

- <forme nominale caractérisant l'ensemble1> "<entité1>" n'est pas <forme nominale caractérisant le groupe d'associations>.

Ex3: le message "X" n'est pas décomposé.

• recommandation:

- il faut utiliser les clauses "<association>" [ou "<association>"...] <explication de la finalité des associations> <forme nominale caractérisant l'ensemble1 ou l'ensemble2>.

Ex1: il faut utiliser les clauses "USES ..." ou "DERIVES ..." ou "ADDS..." ou "REMOVES ..." ou "REFERENCES ..." ou "MODIFIES ..." pour spécifier l'utilisation de la mémoire du SI.

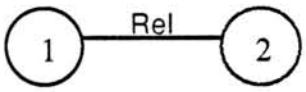
Ex2: il faut utiliser les clauses "REFERENCES..." ou "MODIFIES ..." pour spécifier l'utilisation d'un élément, d'une entité ou d'une relation par un processus.

Ex3: il faut utiliser les clauses "SUBSETS ARE..." ou "CONSISTS OF ..." ou "INCLUDES CONTENT OF ..." pour décomposer un message.

III.5.3 Connectivité minimale d'une association

Après l'introduction d'ensembles d'entités, on doit spécifier les associations entre ces ensembles. Il est nécessaire de formuler des règles pour définir quelles sont les associations à spécifier dans un modèle et leur connectivité minimale obligatoire.

• sous-schéma:



• expression formelle: nous distinguons trois cas pour fournir des phrases plus explicites:

1- ∀e ∈ <ensemble1> . rel(e) ≠ nil

Ex1: ∀e ∈ ELEMENT . format(e) ≠ nil

2- ∀e ∈ <ensemble1> . Rel(e) ≠ ϕ

Ex2: ∀e ∈ ENTITY . Identified_by(e) ≠ ϕ

3- ∀e ∈ <ensemble1> . card(Rel(e)) ≥ x (x ∈ ENTIER ∧ x>1)

Ex3: ∀r ∈ RELATION . card(Relates(r)) ≥ 2

• expression informelle:

- chaque <ensemble1> doit avoir un(e) <forme nominale caractérisant l'association>.

Ex1: chaque élément doit avoir un format.

- chaque <ensemble1> doit avoir au moins <entier> <forme nominale caractérisant l'association>.

Ex2: chaque entité doit avoir au moins 1 identifiant.

- chaque <ensemble1> doit <forme verbale caractérisant l'association> au moins <entier> <forme nominale caractérisant l'ensemble2>.

Ex3: chaque relation doit être associée à au moins 2 entités.

• diagnostic:

- absence <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble1> "<entité1>".

Ex1: absence de format de l'élément "X".

Ex2: absence d'identifiant de l'entité "X".

- absence <forme nominale caractérisant l'ensemble2> <forme verbale caractérisant l'association> <forme nominale caractérisant l'ensemble1> "<entité1>".

Ex3: absence d'entités associées à la relation "X".

- il y a seulement <entier> <ensemble2> <forme verbale caractérisant l'association> <forme nominale caractérisant l'ensemble1> "<entité1>".

Ex3': il y a seulement 1 entité associée à la relation "X".

• recommandation:

- il faut utiliser la clause "<association>" <explication de la finalité de la clause>.

Ex1: il faut utiliser la clause "FORMAT IS ..." pour spécifier le format de l'élément.

- il faut utiliser au moins <entier> clause(s) "<association>" <explication de la finalité de la clause>.

Ex2: il faut utiliser au moins 1 clause "IDENTIFIED BY..." pour spécifier un identifiant de l'entité.

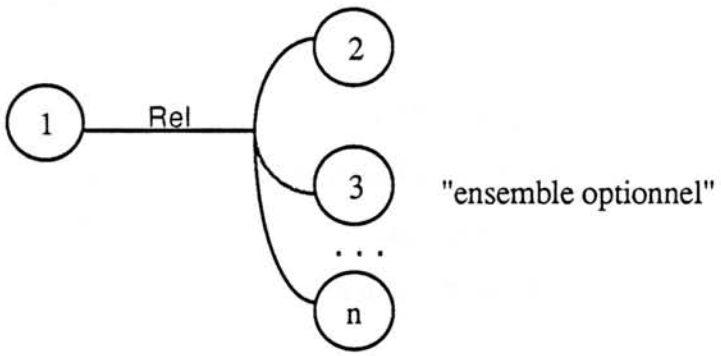
Ex3: il faut utiliser au moins 2 clauses "RELATES ..." pour spécifier les entités de la relation.

Ex3': il faut utiliser au moins 1 clause "RELATES ..." pour spécifier les entités de la relation.

III.5.4 Ensemble obligatoire dans une association composite

Dans le cas d'associations qui comportent des ensembles optionnels dans leurs définitions générales il faut avoir la possibilité, à certaines étapes d'une modélisation, d'appliquer des règles qui imposent un usage plus restrictif de ces associations.

• sous-schéma:



• expression formelle:

$\forall e_1 \in$ <ensemble1> . $\forall e_2 \in$ <ensemble2> . $\forall e_3 \in$ <ensemble3>° .

$((e_2,e_3) \in Rel(e_1) \Rightarrow e_3 \neq nil)$

Ex: $\forall r \in RELATION . \forall e \in ENTITY . \forall x \in ROLE° . \forall s \in (SYS\text{-}PAR \cup STRING)° .$

$((e,x,s) \in Relates(r) \Rightarrow s \neq nil)$

• expression informelle:

- dans <forme nominale caractérisant l'association> <forme nominale caractérisant l'ensemble3> doit être défini(e).

Ex: dans une association entre une entité et une relation la connectivité doit être définie.

• diagnostic:

- absence de <forme nominale caractérisant l'ensemble3> dans <forme nominale caractérisant l'association> entre <forme nominale caractérisant l'ensemble2> "<entité2>" et <forme nominale caractérisant l'ensemble1> "<entité1>".

Ex: absence de "connectivité" dans l'association entre l'entité "X" et la relation "Y".

• recommandation:

- il faut décrire <forme nominale caractérisant l'ensemble3> dans la clause "<association>".

Ex: il faut décrire la connectivité dans la clause "RELATES...".

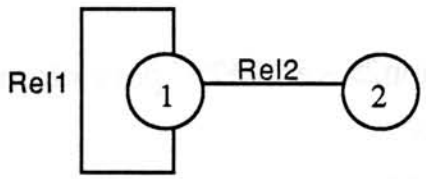
III.5.5 Contrôle d'une hiérarchie

Les spécifications ne correspondent pas toujours à un même niveau de détail et elles sont obtenues par décomposition en spécifications plus élémentaires ou par composition de spécifications déjà réalisées. Il faut des règles pour vérifier que la spécification d'un niveau est complète par rapport au niveau supérieur ou au niveau inférieur.

III.5.5.1 Contrôle ascendant

Il s'agit d'énoncer des règles pour contrôler la complétude d'une entité à un niveau par rapport au niveau directement inférieur.

• sous-schéma:



"décomposition"

• expression formelle:

$\forall e_1, e'_1 \in$ <ensemble1> . $\forall e_2 \in$ <ensemble2 > .

$$((e'_1 \in \text{Rel1}(e_1)) \wedge (e_2 \in \text{Rel2}(e'_1)) \Rightarrow e_2 \in \text{Rel2}(e_1))$$

Ex: $\forall p,p' \in$ PROCESS . $\forall c \in$ CONS-RESOURCE . $\forall t \in$ TIME-CONSTANT .

$\forall n \in$ (SYS-PAR $\cup$ REAL $\cup$ ENTIER)° .

((p' $\in$ Subparts_are(p) $\wedge$ (c,n,t) $\in$ Receives(p')) $\Rightarrow$

(c,n,t) $\in$ Receives(p))

• expression informelle:

- <forme nominale caractérisant l'ensemble2> <forme verbale caractérisant l'association2> par <forme nominale caractérisant l'ensemble1> doit être aussi <forme verbale caractérisant l'association2> par <forme nominale caractérisant l'ensemble1> père s'il existe.

Ex: une ressource consommable utilisée par un processus doit être aussi utilisée par son processus père s'il existe.

• diagnostic:

- <forme nominale caractérisant l'ensemble2> "<entité2>" est <forme verbale caractérisant l'association2> par <forme nominale caractérisant l'ensemble1> "<entité1>" mais pas par <forme nominale caractérisant l'ensemble1> père "<entité1>".

Ex: la ressource consommable "X" est utilisée par le processus "Y" mais pas par son processus père "Z".

• recommandation:

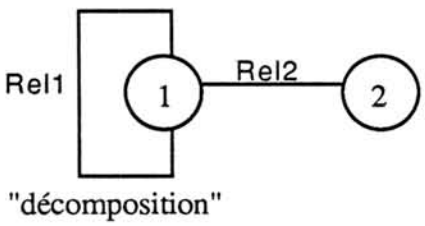
- il faut utiliser une clause "<association2> <entité2>" pour <forme nominale caractérisant l'ensemble1> "<entité1>".

Ex: il faut utiliser une clause "CONSUMES X" pour le processus "Z" .

III.5.5.2 Contrôle descendant

La complétude d'une spécification à un certain niveau doit être contrôlée par rapport au niveau directement supérieur.

• sous-schéma:



"décomposition"

• <u>expression formelle</u>:

$\forall e_1 \in$ <ensemble1> . $\forall e_2 \in$ <ensemble2> . $\exists e'_1 \in$ <ensemble1> .

$((Rel1(e_1) \neq \phi \ \wedge e_2 \in Rel2(e_1)) \Rightarrow$

$(e'_1 \in Rel1(e_1) \wedge e_2 \in Rel2(e'_1)))$

Ex1: $\forall p \in$ PROCESS . $\forall c \in$ CONS-RESOURCE . $\forall t \in$ TIME-CONSTANT .

$\forall n \in (SYS\text{-}PAR \cup REAL \cup ENTIER)^\circ$. $\exists p' \in$ PROCESS .

$((Subparts\_are(p) \neq \phi \ \wedge (m,n,t) \in Consumes(p)) \Rightarrow$

$(p' \in Subparts\_are(p) \wedge (m,n,t) \in Consumes(p')))$

• <u>expressions informelles</u>:

- <forme nominale caractérisant l'ensemble1> <forme verbale caractérisant l'association2> par <forme nominale caractérisant l'ensemble2> décomposé(e) doit être aussi <forme verbale caractérisant l'association2> par au moins un(e) des <ensemble2> fils <forme nominale caractérisant l'ensemble2> décomposé(e) .

Ex1: une ressource consommable utilisée par un processus décomposé doit être aussi utilisé par au moins un des processus fils du processus décomposé.

• <u>diagnostic</u>:

- <forme nominale caractérisant l'ensemble2> "<entité2>" est <forme verbale caractérisant l'association2> par <forme nominale caractérisant l'ensemble1> "<entité1>" mais n'est <forme verbale caractérisant l'association2> par aucun(e) de <forme nominale caractérisant l'ensemble1> fils(filles).

Ex1: la ressource consommable "X" est utilisée par le processus "Y" mais n'est utilisée par aucun de ses processus fils.

• <u>recommandation</u>:

- il faut utiliser une clause "<type d'association> <entité2>" pour au moins un(e) <forme nominale caractérisant l'ensemble1> fils(filles) <forme

nominale caractérisant l'ensemble1> "<entité1>".

Ex1: il faut utiliser une clause "CONSUMES X" pour au moins un des processus fils du processus "Y".

Notons que d'autres règles plus complexes peuvent être formulées pour le contrôle d'une hiérarchie, en considérant notamment la possibilité de décomposition conjointe de l'ensemble2.

III.5.6 Autres dépendances inter-associations

Certaines liaisons entre entités imposent l'existence d'autres liaisons. Ces dépendances entre liaisons sont formulées par des règles qui peuvent être complexes si elles portent sur plusieurs associations. Nous présentons à titre d'exemple des expressions qui correspondent à un cas simple avec seulement deux associations. Cependant d'autres expressions peuvent être formulées.

- <u>sous-schéma</u>:

- <u>expression formelle</u>:

$\forall e_1 \in$ <ensemble1> . $\forall e_2 \in$ <ensemble2> . $(e_2 \in Rel_1(e_1) \Rightarrow e_2 \in Rel_2(e_1))$

Ex: $\forall e \in (ENTITY \cup RELATION \cup MESSAGE \cup PANEL)$,

$\forall s \in SET . \forall n \in (SYS\text{-}PAR \cup ENTIER) . \forall c \in (CALENDAR \cup STRING)^{\circ}$.

$((s,n,c) \in Cardinality(e) \Rightarrow s \in Collected\_in(e))$

• expression informelle:

- <forme nominale caractérisant l'ensemble1> lié(e) à <forme nominale caractérisant l'association1> doit aussi être lié(e) à <forme nominale caractérisant l'ensemble2> <forme verbale caractérisant l'association2> .

Ex: un objet lié à une collection par la notion de cardinalité doit aussi être lié à cette même collection par la notion d'appartenance.

• diagnostic:

- <forme nominale caractérisant l'ensemble1> "<entité1>" doit <forme verbale caractérisant l'association2> <forme nominale caractérisant l'ensemble2> "<entité2>".

Ex: l'objet "X" doit appartenir à la collection "Y".

• recommandation:

- il faut utiliser la clause "<association2> <entité2>" pour <forme nominale caractérisant l'ensemble1> "<entité3>" .

Ex: il faut utiliser la clause "COLLECTED IN Y" pour l'objet "X".

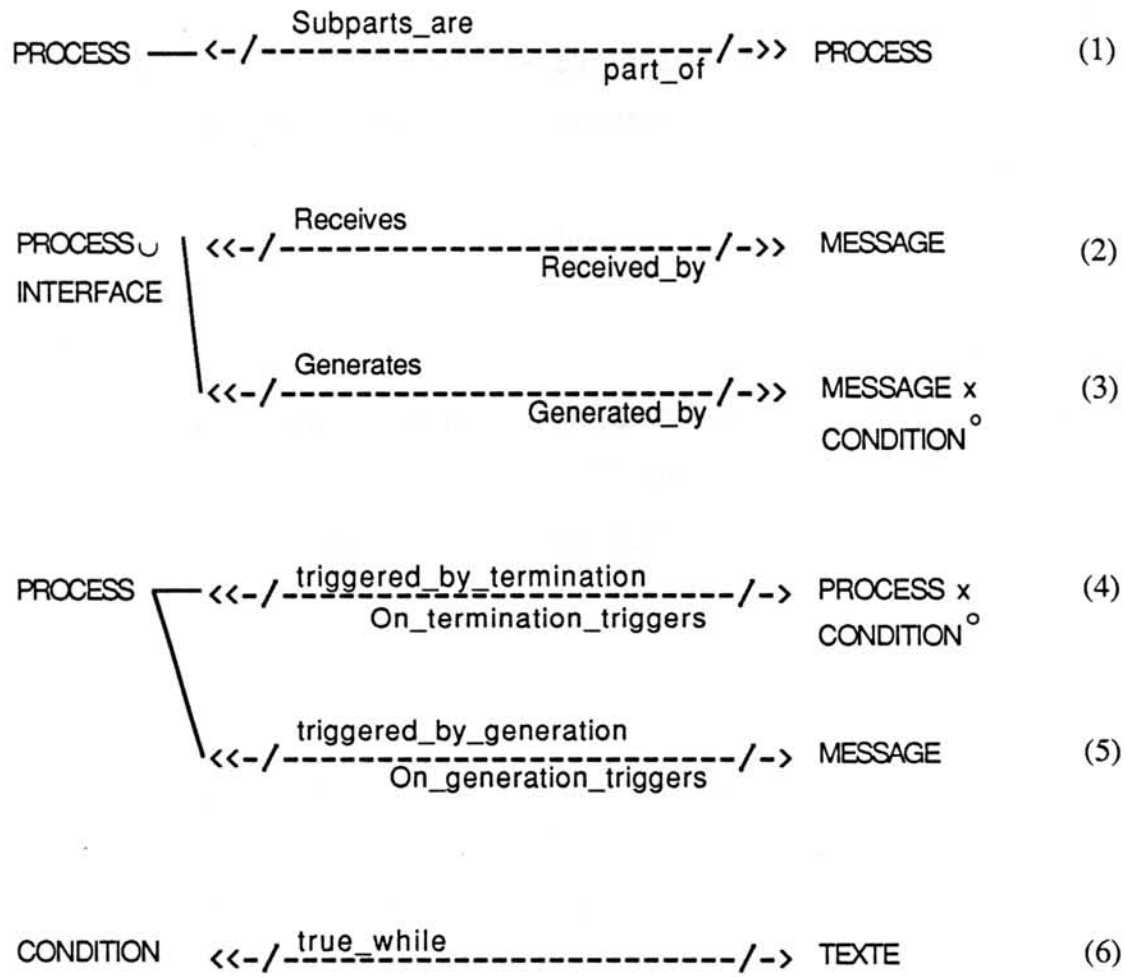
Toutes ces règles correspondent à des C.I. statiques. Nous n'avons pas abordé les C.I. dynamiques, c'est-à-dire qui prennent en compte l'évolution de la spécification, les changements d'états de la spécification.

Le problème fréquent est que, dans une base de spécifications, une spécification ou une partie d'une spécification est jugée incorrecte et il faut la corriger, à la différence d'une base de données où toute donnée stockée est considérée comme correcte. On ne tient pas compte de l'état antérieur de la base de spécifications, on modifie simplement la partie incorrecte .

III.6 Exemple d'un ensemble de règles pour une méthode

Nous illustrons l'utilisation de la typologie de règles définie ci-dessus dans le cadre d'un sous-ensemble de DSL/IDA[BOD88]. Ce sous-ensemble est décrit sous forme d'énoncés Z.

Schémas:



(1) décomposition hiérarchique des processus

(2) réception d'un message par un processus ou une interface

(3) génération d'un message par un processus ou une interface et éventuellement la condition de génération

(4) déclenchement d'un processus par la terminaison d'un autre processus et éventuellement la condition de déclenchement

(5) déclenchement d'un processus par la génération d'un message

(6) définition des conditions

Soit une méthode particulière de modélisation composée de deux étapes. Dans la première étape on doit spécifier les interfaces, la hiérarchie des traitements et les messages échangés entre les interfaces et les traitements. Dans la deuxième étape on doit compléter les spécifications pour préciser le déclenchement de chaque traitement et définir la réalisation de chaque condition.

A partir des énoncés Z du langage de spécification et de la définition informelle ou intuitive de la méthode de modélisation à utiliser, nous proposons une liste de règles élémentaires de gestion qui vont compléter la représentation formelle de cette méthode (cf. §II.4).

La recherche et la formulation de ces règles ont été facilitées par le travail de caractérisation précédent. Mais plusieurs stratégies peuvent être utilisées pour l'obtention des règles. Nous avons utilisé une stratégie basée sur les ensembles d'entités. On sélectionne un ensemble et pour cet ensemble on essaie de formuler des règles selon chaque type décrit en §III.4 et §III.5. Cette stratégie est repétée pour chaque ensemble d'entités du langage de spécification.

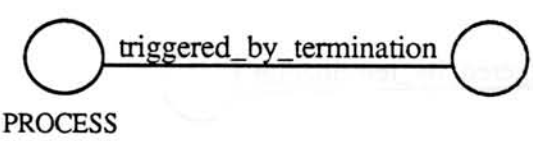
Les règles énoncées ci-dessous sont de type assez varié (10 types différents par rapport aux 17 types possibles) et elles constituent une base de règles pour piloter la méthode souhaitée. D'autres règles peuvent être énoncées sur le sous-ensemble DSL/IDA choisi dans le cadre de différentes méthodes.

R1:

- expression informelle:

  la spécification de processus en termes de déclenchement par la terminaison d'un autre processus est interdite.

- sous-schéma:



- expression formelle:

$\forall p \in PROCESS \; . \; triggered\_by\_termination(p) = nil$

- diagnostic:

  le processus "X" a été spécifié en termes de déclenchement par la terminaison d'un autre processus.

- recommandation:

  supprimer la clause "TRIGGERED BY TERMINATION..." correspondant au processus "X".

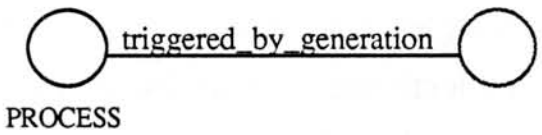
- type de la règle: association interdite.

R2:

- expression informelle:

  la spécification de processus en termes de déclenchement par la génération d'un message est interdite.

- sous-schéma:



- expression formelle:

  $\forall p \in$ PROCESS . triggered_by_generation(p) = nil

- diagnostic:

  le processus "X" a été spécifié en termes de déclenchement par la génération d'un message.

- recommandation:

  supprimer la clause "TRIGGERED BY GENERATION..." correspondant au processus "X".

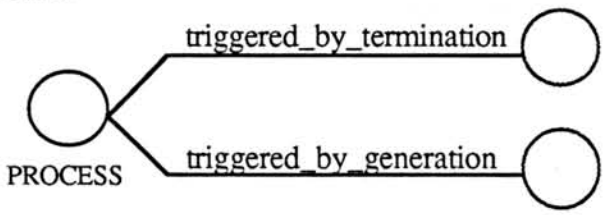
- type de la règle: association interdite.

R3:

- expression informelle:

  le déclenchement d'un processus doit être unique.

- sous-schéma:



- expression formelle:

$$\forall p \in \text{PROCESS} .$$

$$(\text{card}(\text{triggered\_by\_termination}(p)) + \text{card}(\text{triggered\_by\_generation}(p)) \leq 1)$$

- diagnostic:

le déclenchement du processus "X" n'est pas unique.

- recommandation:

modifier les spécifications pour obtenir un déclenchement unique.

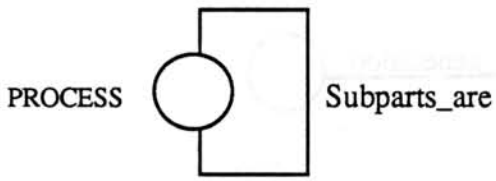
- type de la règle: unicité absolue de définition.

R4:

- expression informelle:

la hiérarchie des traitements ne doit pas être définie circulairement.

- sous-schéma:



- expression formelle:

$$\forall p_1, p_2 \in \text{PROCESS} . \ \forall n_1, n_2 \in \text{ENTIER}^+ .$$

$$\neg(p_1 \in \text{Subparts\_are}^{n1}(p_2) \wedge p_2 \in \text{Subparts\_are}^{n2}(p_1))$$

- diagnostic:

définition circulaire de décomposition des processus "X" et "Y".

- recommandation:

modifier les spécifications de manière à éliminer la définition circulaire.

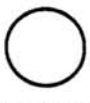
- type de la règle: définition circulaire interdite.

R5:

- expression informelle:

il doit y avoir au moins un processus défini.

- sous-schéma:



- expression formelle:

  card(PROCESS) ≥ 1

- diagnostic:

  absence de spécification de processus.

- recommandation:

  il faut utiliser au moins une clause "DEFINE PROCESS..." pour caractériser les traitements du système.

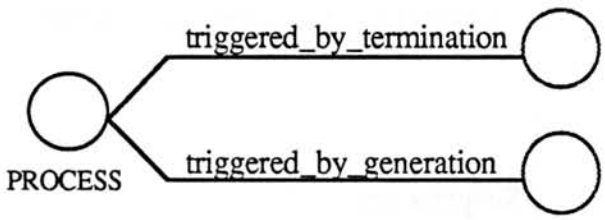
- type de la règle: cardinalité minimale d'un ensemble.

R6:

- expression informelle:

  chaque processus doit avoir un déclenchement.

- sous-schéma:



- expression formelle:

  ∀p ∈ PROCESS .

  (triggered_by_termination(p) ≠ nil ∨ triggered_by_generation(p) ≠ nil)

- diagnostic:

  le processus "X" n'est pas déclenché.

- recommandation:

  il faut utiliser de clause "TRIGGERED BY TERMINATION ..." ou "TRIGGERED BY GENERATION..." pour spécifier le déclenchement d'un processus.

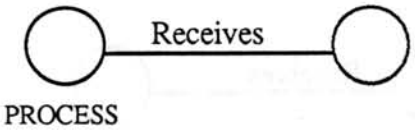
- type de la règle: connectivité minimale d'un groupe d'associations.

R7:

- expression informelle:

  chaque processus doit avoir au moins une entrée.

- sous-schéma:



- expression formelle:

$\forall p \in PROCESS \; . \; Receives(p) \neq \phi$

- diagnostic:

absence d'entrée du processus "X".

- recommandation:

il faut utiliser au moins une clause "RECEIVES..." pour spécifier les entrées du processus.

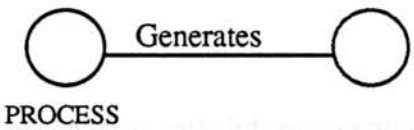
- type de la règle: connectivité minimale d'une association.

R8:

- expression informelle:

chaque processus doit avoir au moins une sortie.

- sous-schéma:



- expression formelle:

$\forall p \in PROCESS \; . \; Generates(p) \neq \phi$

- diagnostic:

absence de sortie du processus "X".

- recommandation:

il faut utiliser au moins une clause "GENERATES..." pour spécifier les sorties du processus.

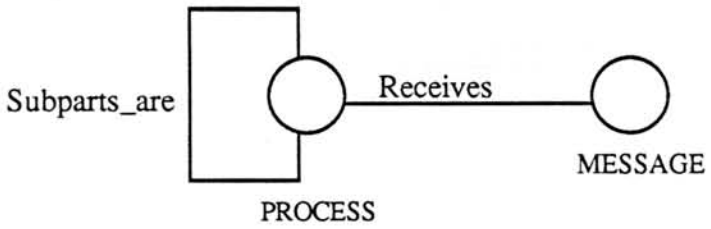
- type de la règle: connectivité minimale d'une association.

R9:

- expression informelle:

un message reçu par un processus doit être aussi reçu par son processus père s'il existe.

• sous-schéma:



• expression formelle:

$\forall p,p' \in PROCESS \; . \; \forall m \in MESSAGE \; .$

$(p' \in Subparts\_are(p) \; \wedge \; m \in Receives(p')) \; \Rightarrow \; m \in Receives(p))$

• diagnostic:

le message "X" est reçu par le processus "Y" mais pas par son processus père "Z".

• recommandation:

il faut utiliser une clause "RECEIVES X" pour le processus "Z".

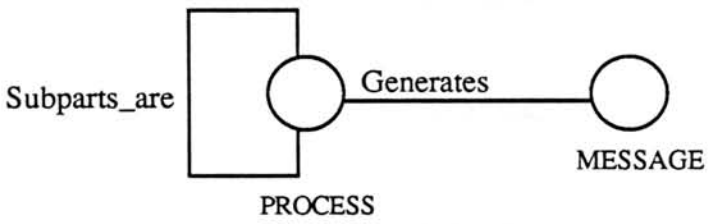
• type de la règle: contrôle ascendant.

R10:

• expression informelle:

un message généré par un processus doit être aussi généré par son processus père s'il existe.

• sous-schéma:



• expression formelle:

$\forall p,p' \in PROCESS \; . \; \forall m \in MESSAGE \; .$

$(p' \in Subparts\_are(p) \; \wedge \; m \in Generates(p')) \; \Rightarrow \; m \in Generates(p))$

• diagnostic:

le message "X" est généré par le processus "Y" mais pas par son processus père "Z".

• recommandation:

il faut utiliser une clause "GENERATES X" pour le processus "Z".

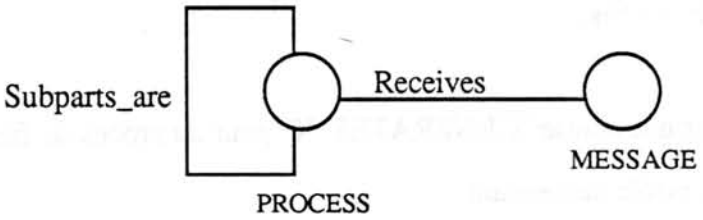
• type de la règle: contrôle ascendant.

R11:

• expression informelle:

un message reçu par un processus décomposé doit être aussi reçu par au moins un des processus fils du processus décomposé.

• sous-schéma:



• expression formelle:

$\forall p \in PROCESS\ .\ \forall m \in MESSAGE\ .\ \exists p' \in PROCESS\ .$

$(Subparts\_are(p) \neq \phi \ \wedge\ m \in Receives(p)) \Rightarrow$

$(p' \in Subparts\_are(p) \ \wedge\ m \in Receives(p'))$

• diagnostic:

le message "X" est reçu par le processus "Y" mais n'est reçu par aucun de ses processus fils.

• recommandation:

il faut utiliser une clause "RECEIVES X" pour un processus fils de "Z".

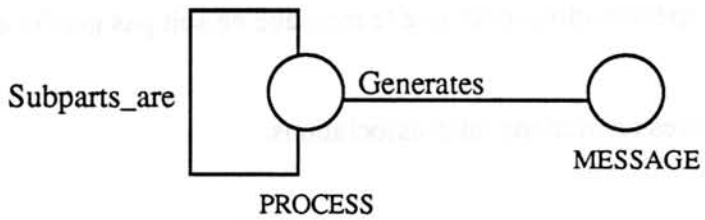
• type de la règle: contrôle descendant.

R12:

• expression informelle:

un message généré par un processus décomposé doit être aussi généré par au moins un des processus fils du processus décomposé.

• sous-schéma:



• expression formelle:

$\forall p \in PROCESS\ .\ \forall m \in MESSAGE\ .\ \exists p' \in PROCESS\ .$

$(Subparts\_are(p) \neq \phi \ \wedge\ m \in Generates(p)) \Rightarrow$

$(p' \in Subparts\_are(p) \ \wedge\ m \in Generates(p'))$

• diagnostic:

le message "X" est généré par le processus "Y" mais n'est pas généré par aucun de ses processus fils.

• recommandation:

il faut utiliser une clause "GENERATES X" pour un processus fils de "Z".

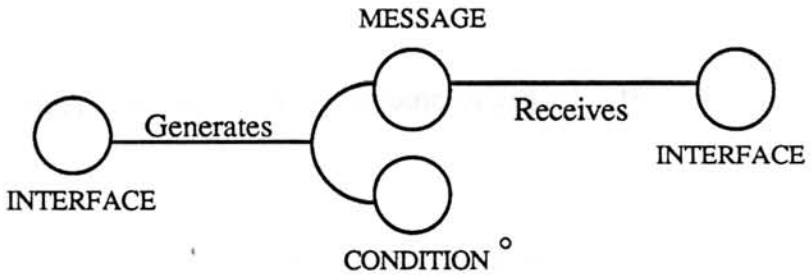
• type de la règle: contrôle descendant.

R13:

• expression informelle:

un message ne peut pas être à la fois généré et reçu par des interfaces.

• sous-schéma:



• expression formelle:

$\forall m \in MESSAGE\ .\ \forall i_1, i_2 \in INTERFACE\ .\ \forall c \in CONDITION^{\circ}\ .$

$\neg((m,c) \in Generates(i_1) \ \wedge\ m \in Receives(i_2))$

• diagnostic:

le message "X" est généré par l'interface "Y" et est reçu par l'interface "Z".

• recommandation:

modifier les spécifications pour que le message ne soit pas généré et reçu par des interfaces.

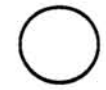
• type de la règle: autres restrictions inter-associations.

R14:

• expression informelle:

il doit y avoir au moins deux messages définis.

- sous-schéma:



- expression formelle:

$card(MESSAGE) \geq 2$

- diagnostics:

absence de spécification de messages.

il y a seulement un message spécifié.

- recommandation:

il faut utiliser au moins <entier> clause(s) "DEFINE MESSAGE..." pour caractériser les messages du système.

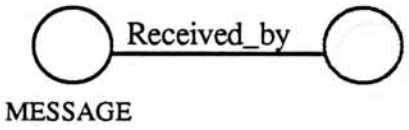
- type de la règle: cardinalité minimale d'un ensemble.

R15:

- expression informelle:

chaque message doit être reçu par un processus ou une interface.

- sous-schéma:



- expression formelle:

$\forall m \in MESSAGE \; . \; Received\_by(m) \neq \phi$

- diagnostic:

absence de réception du message "X".

- recommandation:

il faut utiliser la clause "RECEIVED BY..." pour spécifier la réception d'un message par un processus ou une interface.

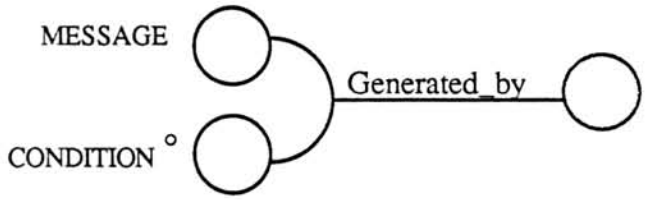
- type de la règle: connectivité minimale d'une association.

R16:

- expression informelle:

chaque message doit être généré par un processus ou une interface.

• sous-schéma:



• expression formelle:

$\forall m \in$ MESSAGE . $\exists c \in$ CONDITION° . Generated_by(m,c) $\neq \phi$

• diagnostic:

absence de génération du message "X".

• recommandation:

il faut utiliser la clause "GENERATED BY..." pour spécifier la génération d'un message par un processus ou une interface.

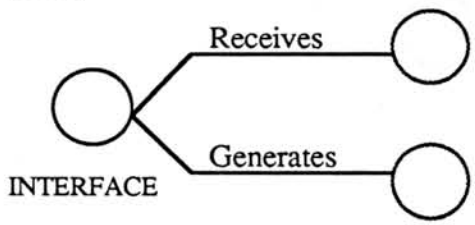
• type de la règle: connectivité minimale d'une association.

R17:

• expression informelle:

chaque interface doit être utilisée.

• sous-schéma:



• expression formelle:

$\forall i \in$ INTERFACE . (Receives(i) $\neq \phi \vee$ Generates(i) $\neq \phi$)

• diagnostic:

absence d'utilisation de l'interface "X".

• recommandation:

il faut utiliser les clauses "RECEIVES..." ou "GENERATES..." pour spécifier l'utilisation de l'interface.

• type de la règle: connectivité minimale d'un groupe d'associations.

R18:

• expression informelle:

il doit y avoir au moins une interface définie.

- sous-schéma:

INTERFACE

- expression formelle:

card(INTERFACE) ≥ 1

- diagnostic:

absence de spécification d'interfaces.

- recommandation:

il faut utiliser au moins une clause "DEFINE INTERFACE..." pour caractériser les interfaces du système.

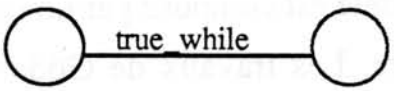
- type de la règle: cardinalité minimale d'un ensemble.

R19:

- expression informelle:

chaque condition doit avoir une définition.

- sous-schéma:



CONDITION

- expression formelle:

∀c ∈ CONDITION . true_while(c) ≠ nil

- diagnostic:

absence de définition de la condition "X".

- recommandation:

il faut utiliser la clause "TRUE WHILE; ..." pour définir la condition.

- type de la règle: connectivité minimale d'une association.

III.7 Dépendances entre règles

La phase d'acquisition des connaissances c'est-à-dire, l'extraction ainsi que la formalisation de celles-ci pour constituer un modèle (cf. §II.4) est une opération très délicate qui peut introduire des anomalies.

Exemple: Soit l'ensemble des règles définies en §III.6. Les trois règles R1, R2 et R6 sont non applicables conjointement car R6 impose qu'un processus doit avoir un déclenchement alors que les règles R1 et R2 interdisent la spécification de déclenchements des processus.

Il est nécessaire d'étudier des dépendances entre règles élémentaires de gestion pour former un modèle "sain". Nous considérons ces dépendances selon quatre critères: incompatibilité, redondance, regroupement et ordre d'exécution.

Il s'agit de détecter les incompatibilités entre règles d'un même modèle. Dans notre cas, un modèle est composé par un ensemble de formules de la logique du premier ordre. Les travaux de Gödel ont montré que le problème de la détection d'inconsistances, c'est-à-dire prouver qu'il existe une formule f telle que $(f \wedge \neg f)$ soit un théorème, est semi-décidable en logique du premier ordre.

Le traitement de ce problème par des spécialistes n'admet aujourd'hui que des solutions très restrictives [AYE87, PIP87, ROU87]; aussi, abordons-nous ce problème selon deux approches complémentaires.

III.7.1 Approche informelle

Un premier axe, très intuitif, est la validation des ensembles de règles qui forment une méthode par des experts. Ces experts doivent utiliser le système de pilotage avec d'une part, des spécifications préparées spécialement pour la validation et d'autre part, des spécifications réelles d'applications existantes.

Exemple: Le problème d'incompatibilité entre les règles R1, R2 et R6 de l'exemple précédent est détecté immédiatement à condition d'avoir au moins un processus défini.

Si dans les spécifications analysées il n'y a pas de déclenchement de processus, la règle R6 est en échec. En ajoutant une spécification (déclenchement de processus) pour satisfaire cette règle, la règle R1 ou la règle R2 serait en échec... Réciproquement, si dans les spécifications analysées il y a un déclenchement de processus la règle R1 ou la règle R2 est en échec. En supprimant le déclenchement du processus pour satisfaire cette règle la règle R6 serait en échec...

L'absence d'un contrôle formel est limitée par le fait que les règles assurent des vérifications. Ainsi, l'existence d'une incohérence entre les règles d'un modèle a pour seule conséquence un faux diagnostic qui ne met pas en cause toute la spécification. La situation serait plus délicate si les règles engendraient des faits nouveaux, c'est-à-dire s'il y avait une déduction de nouveaux faits par les règles.

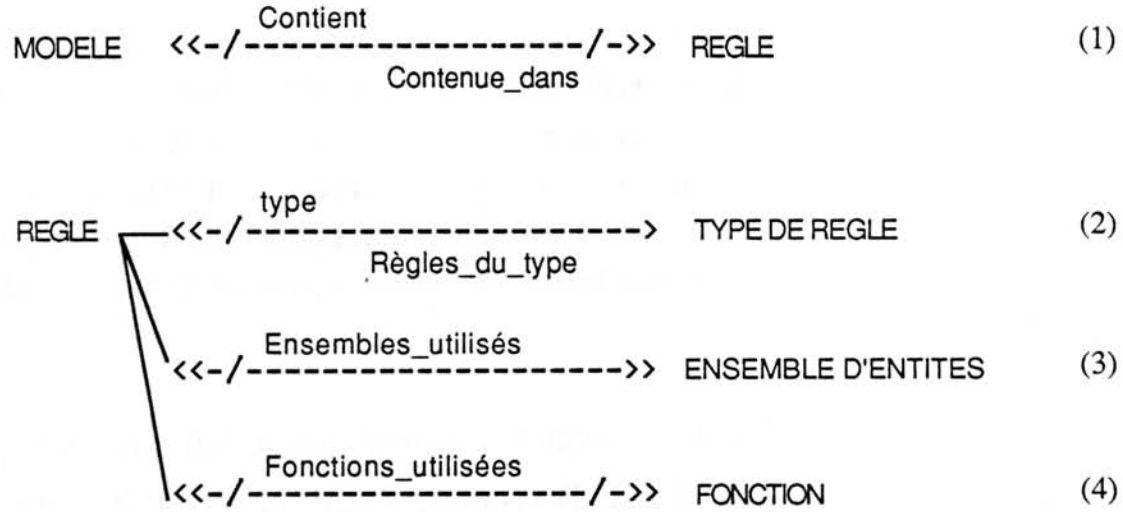
Sans déduction de faits, si au cours d'une modélisation, on détecte un problème dans un ensemble de règles, il suffit de le corriger, sans autres conséquences pour la spécification existante.

III.7.2 Méta-règles

Un second axe plus rigoureux est centré sur le contrôle des règles élémentaires de gestion par des méta-règles. Ce type de contrôle n'est pas complet au sens qu'il ne permet pas d'assurer la cohérence totale d'un ensemble quelconque de règles, mais il peut détecter des situations particulières d'incompatibilité.

La description de méta-règles peut s'appuyer sur le schéma ci-dessous.

III.7.2.1 Schéma général



(1) spécification des règles qui forment un modèle ou un ensemble de règles
(2) caractérisation d'une règle (cf. §III.4 et §III.5)
(3) ensembles d'entités utilisés dans une règle
(4) fonctions utilisées dans une règle

A partir de cette description nous pouvons formuler des méta-règles de contrôle que nous regroupons en trois catégories: exclusion, inclusion et union.

III.7.2.2 Exclusion de règles

Un premier type de dépendance est l'exclusion mutuelle de règles, c'est-à-dire que si une règle est satisfaite, d'autres règles sont en échec. Nous disons qu'il y a exclusion entre les règles r_1 et r_2 , SSi elles ne peuvent pas être satisfaites simultanément.

MR1: Un modèle ne peut pas contenir des règles d'une part, interdisant l'utilisation d'un ensemble d'entités et d'autre part, imposant l'utilisation de ce même ensemble d'entités.

$\forall m \in$ MODELE . $\forall r_1, r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1)='ensemble interdit' $\wedge$

type(r_2)='cardinalité minimale d'un ensemble' $\wedge$

(Ensembles_utilisés(r_1) $\cap$ Ensembles_utilisés(r_2) $\neq \phi$))

MR2: Un modèle ne peut pas contenir des règles d'une part, interdisant l'utilisation d'une association et d'autre part, imposant l'utilisation de cette même association.

$\forall m \in$ MODELE . $\forall r_1, r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1)='association interdite' $\wedge$

type(r_2)='connectivité minimale d'une association' $\wedge$

(Fonctions_utilisées(r_1) $\cap$ Fonctions_utilisées(r_2) $\neq \phi$))

III.7.2.3 Inclusion de règles

Un deuxième type de dépendance est l'inclusion de règles, c'est-à-dire que la restriction imposée par une règle élémentaire de gestion est contenue dans la restriction imposée par d'autres règles. La règle r_1 est dite incluse dans la règle r_2 SSi la "validité" de la règle r_2 implique la "validité" de la règle r_1.

Exemple: La règle "une entité ne doit pas avoir plus de 5 synonymes" est incluse dans la règle "une entité ne doit pas avoir de synonyme".

Il est souhaitable d'éliminer cette forme de redondance entre les règles d'un même modèle afin d'éviter une multiplication des messages communiqués à l'utilisateur et de minimiser les temps de vérification.

MR3: Un modèle ne peut pas avoir plus d'une règle sur la cardinalité minimale d'un ensemble.

$\forall m \in$ MODELE . $\forall r_1 \neq r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1) = type(r_2) = 'cardinalité minimale d'un ensemble' $\wedge$

(Ensembles_utilisés(r_1) $\cap$ Ensembles_utilisés(r_2) $\neq \phi$))

MR4: Un modèle ne peut pas avoir plus d'une règle sur la connectivité minimale d'une association.

$\forall m \in$ MODELE . $\forall r_1 \neq r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1)=type(r_2)='connectivité minimale d'une association' $\wedge$

(Fonctions_utilisées(r_1) $\cap$ Fonctions_utilisées(r_2) $\neq \phi$))

MR5: Un modèle ne peut pas contenir des règles d'une part, limitant l'utilisation d'une association et d'autre part, interdisant complétement l'utilisation de cette même association.

$\forall m \in$ MODELE . $\forall r_1, r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1)='association interdite' $\wedge$

type(r_2)='connectivité maximale d'une association' $\wedge$

(Fonctions_utilisées(r_1) $\cap$ Fonctions_utilisées(r_2) $\neq \phi$))

MR6: Un modèle ne peut pas contenir des règles d'une part, limitant l'utilisation de certains ensembles dans une association et d'autre part, interdisant complétement l'utilisation de cette même association.

$\forall m \in$ MODELE . $\forall r_1, r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1)='association interdite' $\wedge$

type(r_2)='sous-ensemble interdit dans une association' $\wedge$

(Fonctions_utilisées(r_1) $\cap$ Fonctions_utilisées(r_2) $\neq \phi$))

MR7: Un modèle ne peut pas contenir des règles d'une part, imposant l'utilisation de certains ensembles dans une association et d'autre part, interdisant complétement l'utilisation de cette même association.

$\forall m \in$ MODELE . $\forall r_1, r_2 \in$ REGLE .

$\neg(r_1 \in$ Contient(m) $\wedge$ $r_2 \in$ Contient(m) $\wedge$

type(r_1)='association interdite' $\wedge$

type(r_2)='ensemble obligatoire dans une association composite' $\wedge$

(Fonctions_utilisées(r_1) $\cap$ Fonctions_utilisées(r_2) $\neq \phi$))

III.7.2.4 Union de règles

Il est souhaitable d'appliquer conjointement certaines règles. Ce type de dépendance peut être aussi contrôlé par des méta-règles. Ces méta-règles sont de la forme générale:

MR8: Un modèle qui contient la règle "X" doit contenir aussi la règle "Y".

$\forall m \in$ MODELE .

('règle1' $\in$ Contient(m) $\Rightarrow$ 'règle2' $\in$ Contient(m))

Ces trois catégories de méta-règles introduites pour constituer des ensembles cohérentes de règles sont utilisées dans un module de définition de méthodes qui décrit rigoureusement une méthode spécifique à suivre dans un projet particulier.

III.7.3 Ordre d'application des règles

L'ordre d'application des règles de vérification est certainement l'un des aspects le plus important dans un système de pilotage. Nous considérons deux niveaux principaux dans cet ordonnancement des règles:

• un **niveau externe** qui est fonction de la forme choisie de pilotage. Ce niveau sera étudié au chapitre IV.

• un **niveau interne** qui considère l'ordre d'exécution des règles de vérification comme résultat d'une action de l'utilisateur. Ce niveau est analysé ci-dessous.

Dans un tel niveau interne, il faut d'abord définir l'objectif de l'ordonnancement des vérifications. Nous concevons deux objectifs principaux, pas toujours compatibles: privilégier le nombre de messages à communiquer à l'utilisateur ou privilégier la durée des vérifications.

Dans le domaine des bases de données nous constatons des préoccupations analogues. Dans le système TAXIS [NIX87] par exemple, les contraintes sont classées en trois types: intra-classe, inter-classe et temporelle. Une contrainte intra-classe ne porte que sur des attributs d'une même classe, alors qu'une contrainte inter-classe considère des attributs de plusieurs classes. Une contrainte temporelle considère des entités associées au temps. Si plusieurs contraintes sont associées à un attribut, TAXIS active d'abord les contraintes intra-classes, ensuite les contraintes inter-classes et termine par les contraintes temporelles. L'objectif est de minimiser la durée des vérifications.

Sur ce problème d'ordre d'exécution des règles de vérification, notre objectif principal est de minimiser le nombre de messages à donner à l'utilisateur. Dans cette optique, on doit exécuter les règles de vérification de la cohérence avant les règles de vérification de la complétude: le but est de montrer à l'utilisateur les erreurs déjà commises avant de demander de nouvelles spécifications. Un autre critère est de regrouper les règles par rapport à un même ensemble d'entités. Un troisième critère est d'exécuter les règles plus générales avant les règles plus spécifiques.

La typologie proposée dans ce chapitre respecte ces critères (cf. §III.2). Ainsi, l'ordre d'exécution des règles doit suivre l'ordre de définition de cette typologie pour chaque ensemble d'entités. Entre règles d'un même type le chef de projet doit prendre en compte la sémantique des associations pour décider de l'ordre d'application des règles.

Exemple: Soit deux règles élémentaires de gestion représentées par les expressions suivantes:

R_i -

expression informelle: chaque entité doit avoir au moins un attribut.

expression formelle: $\forall e \in$ ENTITY . $\exists n \in$ (SYS-PAR $\cup$ INTEGER)° . Consists_of(e,n) $\neq \phi$)

R_j -

expression informelle: chaque entité doit avoir au moins un identifiant.

expression formelle: $\forall e \in$ ENTITY . Identified_by(e) $\neq \phi$)

La règle R_i doit être activée avant la règle R_j pour respecter le troisième critère, car "avoir un attribut" est plus général qu'"avoir un identifiant": un identifiant étant un attribut ou un groupe d'attributs.

III.8 Conclusion

Les règles qui viennent d'être présentées constituent le noyau de notre approche; c'est pourquoi nous avons voulu les énoncer d'une manière systématique et exhaustive.

Le chapitre suivant permet de proposer et d'expérimenter des modes de définition, de stockage et d'exécution de ces règles comme données de base d'un système expert de pilotage de la modélisation d'un SI.

CHAPITRE IV

UN SYSTEME DE PILOTAGE

UN SYSTEME DE PILOTAGE

IV.1 Introduction

Après avoir introduit une représentation formelle des méthodes de modélisation de SI et proposé une typologie de règles associées à cette représentation nous revenons à notre objectif principal: le pilotage d'une modélisation en ne considérant que les connaissances liées aux méthodes, sans aborder le problème des connaissances générales d'analyse et des connaissances d'un domaine spécifique d'application (cf. §I.3.2.1 et §I.3.2.2).

Nous commençons par décrire le point de vue externe de l'utilisateur face à un système de pilotage puis nous proposons une architecture fonctionnelle d'un tel système et enfin nous présentons une structure d'implémentation particulière d'un prototype réalisé pour montrer la faisabilité de nos propositions.

IV.2 Aspects externes du pilotage

Un système de pilotage de modélisation doit être adapté à différentes formes de conduite de l'activité des analystes-concepteurs. Dans ce cadre nous pouvons dégager deux formes extrêmes de pilotage [GAT86]:

• **guidage permanent et contrôle ponctuel**- Le système **dirige** chaque action de l'utilisateur, il lui demande chaque fait selon la démarche choisie et l'utilisateur n'a pratiquement aucun degré de liberté.

- Les règles de vérification de la cohérence sont déclenchées en permanence sur chaque nouvelle spécification (introduction d'un nouveau fait).

  Exemple: Lors de l'introduction d'une clause SUBPARTS ARE ou PART OF en DSL/IDA, il est envisageable de déclencher une règle pour s'assurer qu'il n'y a pas de cycle dans la décomposition des processus concernés (cf. §III.4.6.1).

- Les règles de vérification de la complétude sont sous forme de demande d'informations complémentaires à l'utilisateur.

Exemple: Dans le cadre d'une spécification en DSL/IDA, une règle d'expression informelle "Chaque élément doit avoir un format" peut avoir comme effet l'affichage d'un écran de saisie du type

DEFINE ELEMENT nom-électeur;
FORMAT .....

à chaque ajout d'un nouveau fait de type ELEMENT.

NB: Dans les exemples nous ne considérons qu'une interface traditionnelle textuelle.

• **contrôle global a posteriori** - L'utilisateur est **complétement libre** au cours de cette activité de modélisation. Il introduit les spécifications dans l'ordre qu'il désire et ce n'est que lorsqu'il juge terminée cette modélisation, qu'il la soumet à une validation globale par le système qui déclenche alors l'ensemble des règles de vérification de cohérence et de complétude.

En réalité, il faut pouvoir choisir un **mode mixte de pilotage** entre ces deux extrêmes. Il s'agit d'un compromis satisfaisant entre le degré de liberté et de guidage de l'utilisateur, en fonction particulièrement de son expérience et de ses connaissances de la méthode à suivre.

Par analogie, dans le domaine de l'EAO on trouve plusieurs niveaux de conduite. Certains systèmes ont un contrôle total et permanent sur les activités de l'élève, en adaptant l'action selon les réponses de celui-ci, mais en conservant toujours le contrôle (système "maître"). D'autres systèmes, proposent un contrôle mixte de dialogue où le contrôle est partagé entre l'élève et le système par l'échange de questions et de réponses. Des systèmes basés sur l'apprentissage guidé par la découverte, laissent l'élève diriger complétement son activité (système "esclave") [WEN87].

De même, dans le domaine de la programmation, un compilateur correspond à un pilotage avec contrôle global a posteriori alors qu'un interpréteur peut être considéré comme un mode mixte situé entre les deux extrêmes introduits ci-dessus puisqu'il assure certaines vérifications pendant l'édition d'un programme.

Considérant ces différentes formes de pilotage de la modélisation de SI, l'architecture d'un tel système doit permettre plusieurs types de fonctionnement. En particulier il faut pouvoir accepter plusieurs possibilités d'activation de règles de contrôle comme par exemple:

- une activation dynamique permanente et immédiate à partir de l'éditeur du langage de spécification;
- une activation à des moments privilégiés définis a priori;
- une activation globale a posteriori.

Cette architecture doit aussi être adaptable à l'interface utilisateur choisie, comme par exemple, en cas de diagnostic d'erreur, fournir simplement le message associé ou compléter le message d'erreur par des éléments pour la correction. Le passage automatique dans l'environnement de correction doit être possible et la proposition d'une correction automatique est envisageable.

IV.3 Architecture fonctionnelle d'un système de pilotage

Par système de pilotage de la modélisation de SI nous considérons un outil capable de conduire et de vérifier le travail de modélisation.

L'idée générale est, d'une part, de permettre au chef de projet de définir et d'affiner précisément une méthode à l'aide de règles (contraintes d'intégrité par rapport au méta-modèle) et, d'autre part, pendant la modélisation, de vérifier que ces règles sont respectées. Ainsi, nous proposons une architecture fonctionnelle (cf. figure IV.1) basée sur deux sous-systèmes:

- un **sous-système de configuration** pour créer une base de connaissances sur une méthode personnalisée, c'est-à-dire définir le cadre méthodologique de déroulement de la modélisation d'applications;
- un **sous-système de pilotage** qui utilise cette base comme un mode de fonctionnement particulier pour guider et vérifier une activité de spécification.

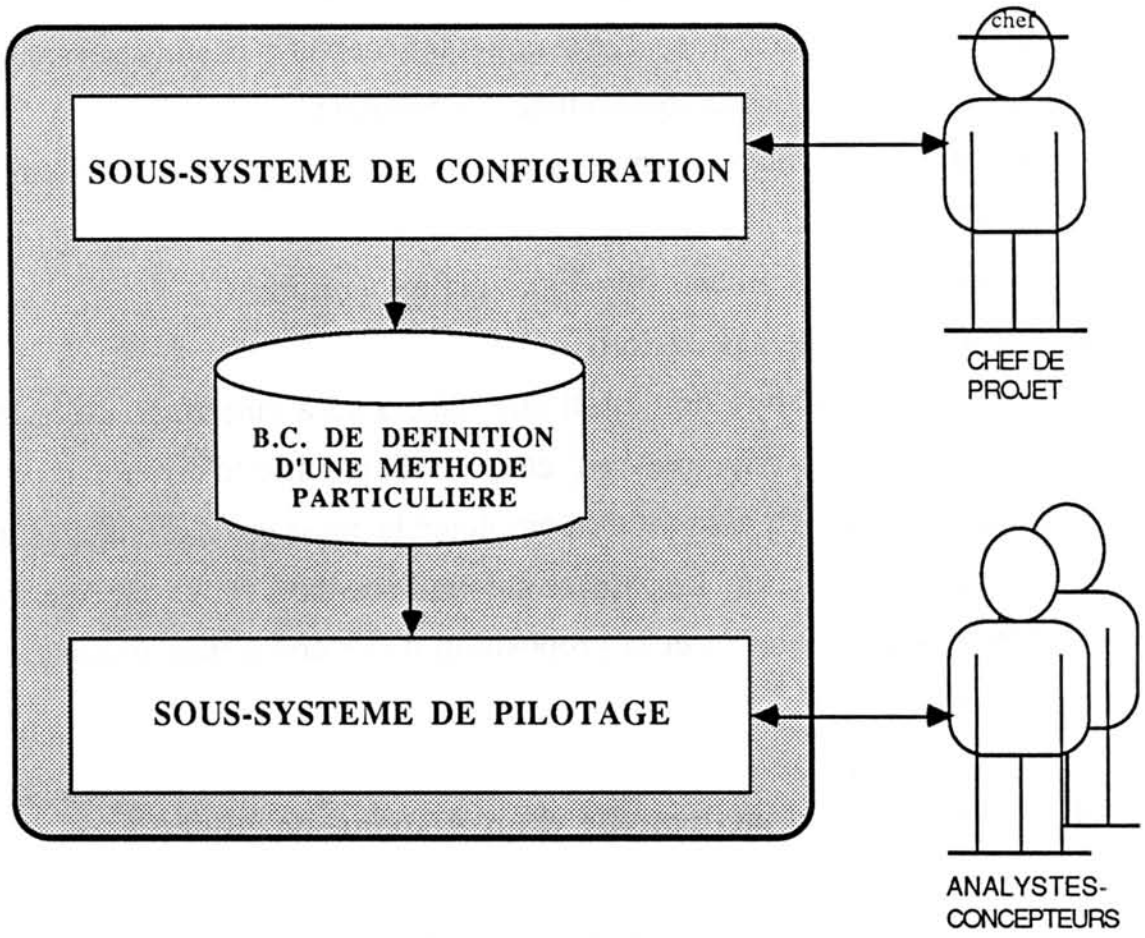


Figure IV.1: Architecture fonctionnelle d'un système général de pilotage

IV.3.1 Sous-système de configuration

Un grand défaut des systèmes actuels d'aide à la modélisation des SI est le manque de flexibilité par rapport aux caractéristiques spécifiques de chaque utilisateur et du système à modéliser. La majorité des outils d'aide ignorent l'aspect démarche ou imposent une démarche particulière et ne font pas ou peu de vérifications sur un modèle. Ils sont souvent très rigides dans leur mode d'utilisation. Pour limiter ces problèmes, nous proposons un système de configuration qui permet à un chef de projet d'adapter l'outil de pilotage aux besoins spécifiques et à l'approche particulière d'une équipe

d'analystes-concepteurs.

L'objectif de ce sous-système est de gérer la définition d'une base de connaissances caractérisant une méthode à utiliser pour une application particulière dans un environnement précis.

Fonctions

Ce sous-système met à la disposition d'un chef de projet un ensemble de fonctions pour choisir principalement:

- les règles de contrôle à activer;
- le mode d'activation souhaité;
- le type d'interface utilisateur à employer;
- les étapes de la modélisation (les règles de contrôle à appliquer à chaque étape).

Stratégie

Le sous-système de configuration est conforme à la stratégie de description d'une méthode proposée au chapitre II. Nous rappelons que cette stratégie s'appuie sur la représentation d'une méthode comme une séquence de modèles.

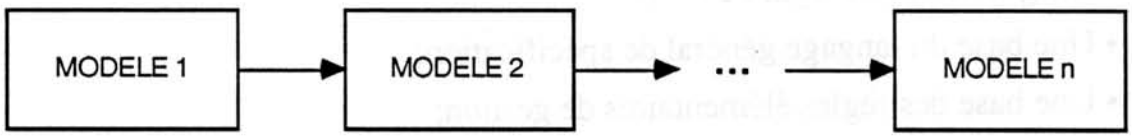


Figure IV.2: Une méthode vue comme une séquence de modèles

Un modèle M est caractérisé par un langage de spécification L et par un ensemble E de règles élémentaires de gestion. Ces règles se comportent comme des contraintes d'intégrité définies sur le langage; elles sont traitées en deux catégories distinctes: cohérence et complétude. Langage et règles sont

décrits formellement en Z (cf. figure IV.3).

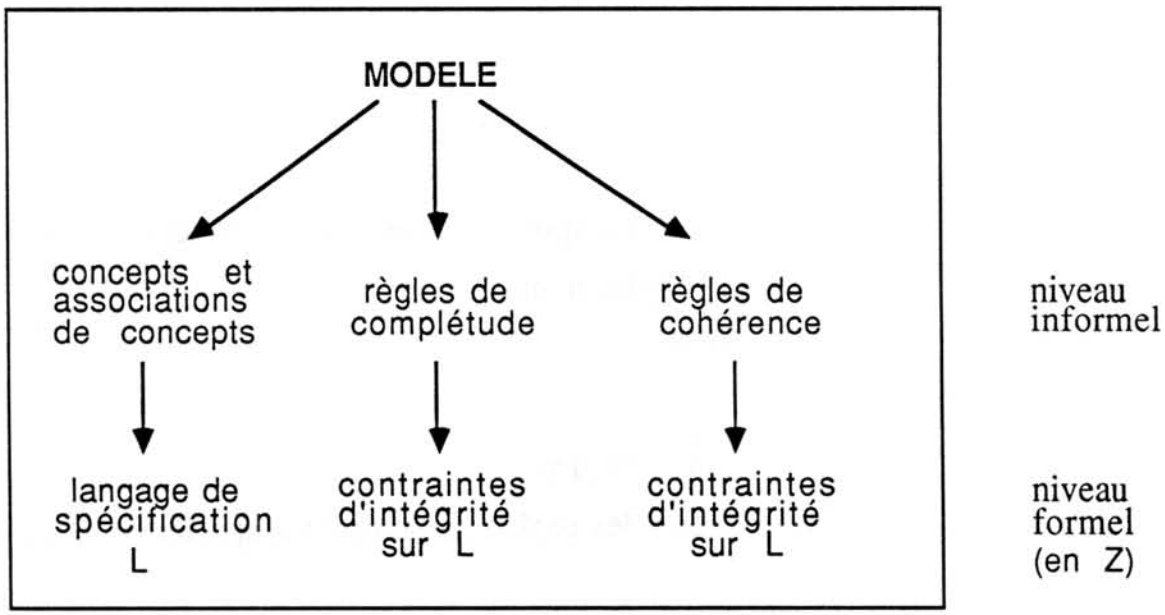


Figure IV.3: Formalisation d'un modèle

Architecture

Cinq composantes principales caractérisent ce sous-système de configuration (cf. figure IV.4):

- Une base du langage général de spécification;
- Une base des règles élémentaires de gestion;
- Une base des méta-règles de contrôle;
- Une interface de saisie et de mise à jour du langage et des règles;
- Une interface de configuration de méthodes.

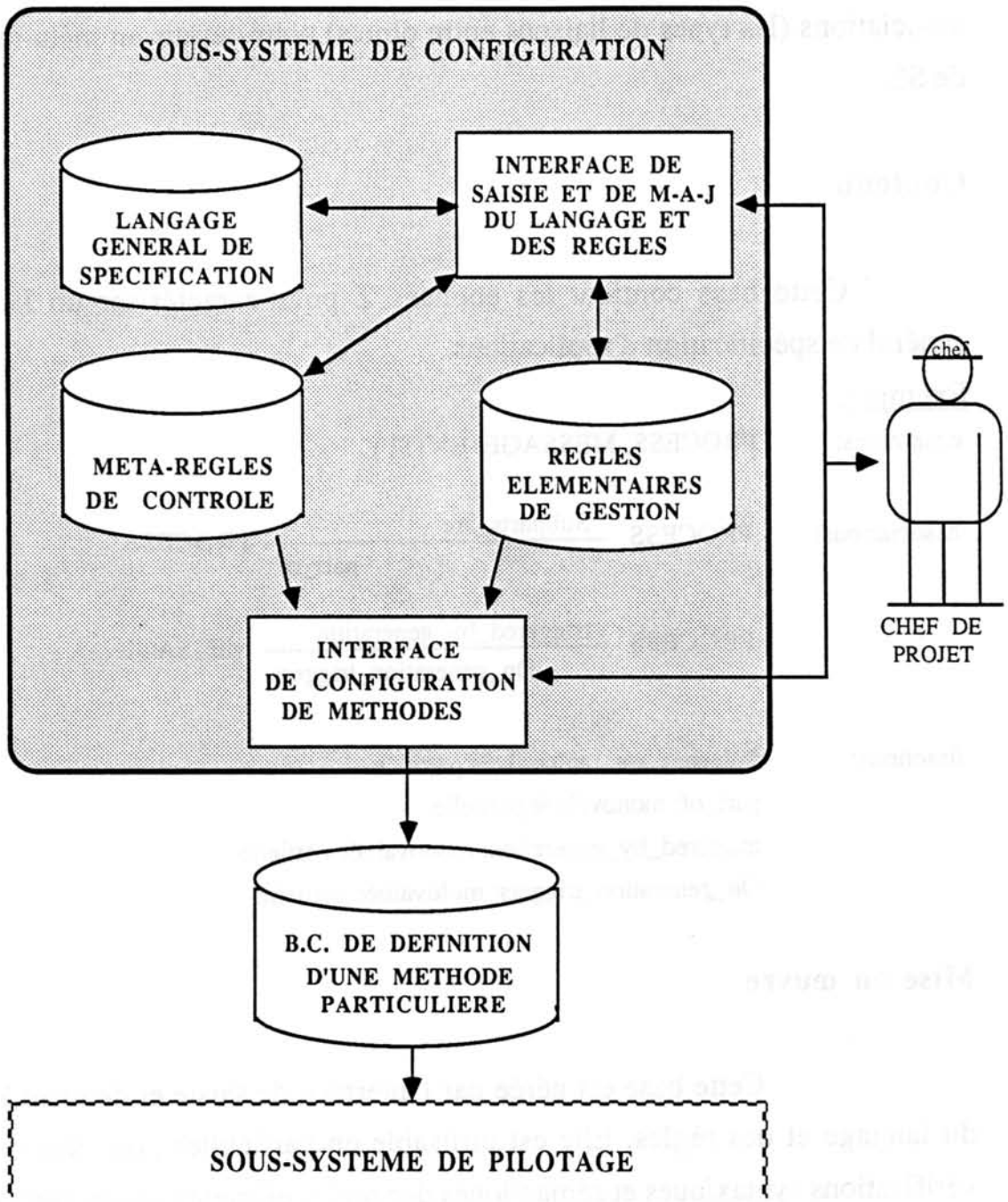


Figure IV.4: Architecture fonctionnelle d'un sous-système de configuration de méthodes

IV.3.1.1 Base du langage général de spécification

La base du langage général de spécification contient l'ensemble des définitions fixant le cadre conceptuel des modèles et des méthodes envisagés. Il s'agit essentiellement de recenser les ensembles (les types d'objets) et leurs

associations (les types de liaisons entre objets) pour définir un méta-modèle de SI.

Contenu

Cette base contient les énoncés Z pour caractériser un langage général de spécification d'applications.

Exemple:

ensembles: PROCESS, MESSAGE, ENTITY.

associations: PROCESS —Subparts_are / part_of— PROCESS

PROCESS —triggered_by_generation / On_generation_triggers— MESSAGE

fonctions: Subparts_are: multivaluée partielle
part_of: monovaluée partielle
triggered_by_generation: monovaluée partielle
On_generation_triggers: multivaluée partielle

Mise en œuvre

Cette base est gérée par l'interface de saisie et de mise à jour du langage et des règles. Elle est utilisable en particulier pour réaliser les vérifications syntaxiques et sémantiques des règles élémentaires de gestion.

IV.3.1.2 Base des règles élémentaires de gestion

La base des règles élémentaires de gestion contient l'ensemble des restrictions au langage général de spécification utilisé pour la définition formelle d'une méthode.

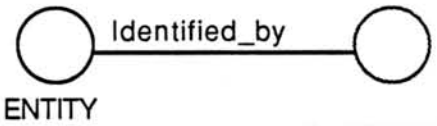
Cette base permet ainsi d'exprimer plusieurs variantes particulières d'un langage de spécification.

Contenu

Cette base contient l'ensemble des règles élémentaires de gestion définies par rapport au langage de spécification utilisé dans une méthode. Ces règles, étudiées au chapitre III, sont décrites par cinq composantes (cf. §III.3). L'exemple ci-dessous explicite cet aspect.

Exemple:

- expression informelle:
  Chaque entité doit avoir au moins un identifiant.
- sous-schéma:



- expression formelle:
  $\forall e \in ENTITY \; . \; Identified\_by(e) \neq \phi$
- diagnostic:
  Absence d'identifiant de l'entité "X".
- recommandation:
  Il faut utiliser au moins 1 clause "IDENTIFIED BY ..." pour spécifier un identifiant de l'entité.

Mise en œuvre

Cette base est gérée par l'interface de saisie et de mise à jour du langage et des règles et constitue une entrée de l'interface de configuration de méthodes.

IV.3.1.3 Base des méta-règles de contrôle de méthodes

La base des méta-règles de contrôle contient des règles utilisées pour former **une méthode "saine"**.

Schématiquement une méthode est définie par plusieurs ensembles de règles (modèles) et il est souhaitable que dans chaque ensemble de règles il n'y ait aucune incompatibilité au sens où nous l'avons décrit au chapitre III en introduisant des méta-règles (cf. §III.7.2).

Ces méta-règles initiales, introduites au chapitre III pour traiter le problème d'incompatibilité entre règles d'un modèle, sont complétées par d'autres méta-règles pour effectuer des vérifications supplémentaires de la définition d'une méthode particulière, comme par exemple, pour s'assurer que chaque méthode contient au moins un modèle, que chaque modèle a au moins une règle élémentaire de gestion associée et que l'enchainement des modèles est satisfaisant.

Notons que ces méta-règles sur chaque modèle portent directement sur les règles de cohérence et de complétude (cf. figure IV.5) et non sur le langage général de spécification.

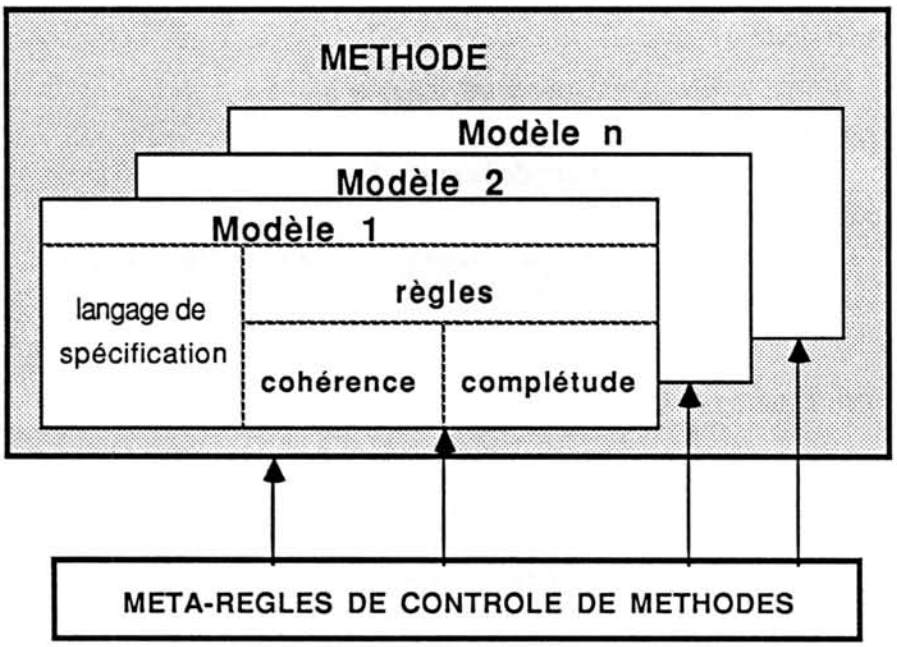


Figure IV.5: Contrôle d'une méthode par des méta-règles.

Contenu

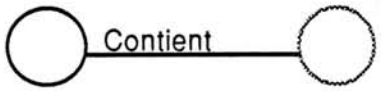
Cette base contient l'ensemble des méta-règles utilisées pour contrôler la définition d'une méthode de modélisation de SI.

Les méta-règles sont décrites par les mêmes composantes que les règles élémentaires de gestion: expression informelle, sous-schéma, expression formelle, diagnostic et recommandation.

Exemple:

expression informelle:
Chaque modèle doit contenir au moins une règle élémentaire de gestion.

sous-schéma:



MODELE

expression formelle:
$\forall m \in$ MODELE . Contient(m) $\neq \phi$

diagnostic:
Absence de règles élémentaires de gestion du modèle "X".

recommandation:
Spécifier les règles qui forment ce modèle.

Mise en œuvre

Cette base des méta-règles de contrôle, gérée par l'interface de saisie et de mise à jour du langage et des règles, est utilisée par l'interface de configuration de méthodes pour vérifier la définition d'une méthode particulière de modélisation.

IV.3.1.4 Interface de saisie et de mise à jour du langage et des règles

L'interface de saisie et de mise à jour gère les trois bases qui seront utilisées dans la définition précise d'une méthode: la base de définition du langage général, la base des règles élémentaires de gestion ainsi que la base des méta-règles de contrôle.

Fonctionnalités principales de l'interface de gestion du langage et des règles

Le dialogue avec le chef de projet permet d'activer cinq **fonctions de base** de cette interface:

- Acquisition d'énoncés Z de type:
  - langage de spécification;
  - règles élémentaires de gestion;
  - méta-règles de contrôle.
- Acquisition des énoncés informels des règles.
- Vérification des énoncés Z. Ces vérifications sont de trois types:
  - vérification syntaxique de chaque énoncé Z;
  - vérification sémantique de chaque énoncé Z du type règle élémentaire de gestion par rapport aux énoncés Z du langage de spécification;
  - vérification sémantique de chaque énoncé Z du type méta-règle de contrôle par rapport aux énoncés Z pré-définis décrivant le schéma général des méta-règles (cf. §III.7.2.1).
- Ajout, modification et suppression de règles de la base des règles élémentaires de gestion ou de la base des méta-règles de contrôle et de définitions relatives au langage. Il est important d'observer qu'une modification ou une suppression d'énoncés Z du langage de spécification implique la revalidation de toutes les règles élémentaires de gestion déjà stockées car ces validations sont faites par rapport à ces énoncés Z.
- Construction de rapports sur le contenu de ces trois bases. Deux types de rapports sont indispensables:
  - rapport documentaire et d'analyse pour présenter sous plusieurs formes (listes, tableaux, graphiques) le contenu de ces bases;
  - rapport de type diagnostic.

Autres fonctionnalités

En plus de ces fonctions de base, une interface plus évoluée pourrait avoir les **extensions** suivantes:

- détermination automatique du type (selon la typologie utilisée dans le chapitre III) de la règle en cours d'acquisition;
- comparaison d'une nouvelle règle avec les autres règles de même type et de même domaine déjà stockées avant de décider du stockage de cette nouvelle règle.

Exemple:

contenu de la base des règles élémentaires de gestion:

. . . . . .

R_i: $\forall p \in$ PROCESS . (Adds(p)$\neq\phi \vee$ References(p)$\neq\phi \vee$ Modifies(p)$\neq\phi$)

. . . . . .

la nouvelle règle

R_j: $\forall p \in$ PROCESS . (Uses(p)$\neq\phi \vee$ Adds(p)$\neq\phi \vee$ References(p)$\neq\phi \vee$ Modifies(p)$\neq\phi$)
doit être acceptée bien qu'étant de même type et portant sur les mêmes ensembles: les formules logiques de R_i et R_j ne sont pas équivalentes et supportent explicitement des méthodes de modélisation différentes.

la nouvelle règle

R_k: $\forall p \in$ PROCESS . (References(p)$\neq\phi \vee$ Modifies(p)$\neq\phi \vee$ Adds(p)$\neq\phi$)
doit être refusée: les formules logiques de R_i et R_k sont équivalentes.

- analyse des énoncés informels et des expressions formelles de plusieurs règles pour comparer leurs formes nominales et leurs domaines (ensembles et fonctions) afin d'interroger éventuellement l'utilisateur sur la "différence" constatée. Cette approche mise en œuvre dans le système TEIRESIAS [DAV77] permet par l'étude de la portée des règles de mettre en évidence des anomalies éventuelles dans un lot de règles.

Mise en œuvre

Un déroulement du fonctionnement de cette interface serait:

1- introduction des énoncés Z de définition du langage de spécification.

2- définition des règles élémentaires de gestion qui sont des restrictions à l'utilisation du langage général de spécification. La conformité des règles vis à vis du langage de spécification doit être vérifiée.

Notons que la gestion des méta-règles est analogue à la gestion des règles. La différence entre ces deux types de règles est au niveau du langage: les méta-règles portent sur un langage pré-défini indépendamment de toute méthode alors que les règles portent sur le langage propre de chaque méthode.

IV.3.1.5 Interface de configuration de méthodes

L'objectif de l'interface de configuration de méthodes est la définition précise de la méthode à suivre dans un projet particulier. Rappelons qu'une telle spécification comprend la définition et l'ordonnancement de modèles (cf. figures IV.2 et IV.3). Ainsi, l'interface de configuration de méthodes permet à un chef de projet de constituer des ensembles de règles à partir de la base générale de règles élémentaires de gestion pour constituer les modèles et permet aussi la définition de la séquence des modèles. Cette interface vérifie la méthode ainsi définie, à l'aide des méta-règles de contrôle, et génère une base de connaissances particulière à cette méthode personnalisée prise en compte dans le sous-système de pilotage.

Fonctionnalités de l'interface de configuration

A l'aide de cette interface, le chef de projet peut accomplir quatre activités importantes dans la définition d'une méthode. Chacune de ces activités peut être réalisée en utilisant plusieurs fonctionnalités plus élémentaires.

- **définition des modèles :**
  - caractérisation informelle des modèles;
  - choix des règles associées à chaque modèle (cf. §II.4.2);
  - vérification de chaque modèle à l'aide des méta-règles de contrôle pour éviter l'incompatibilité entre règles ou l'oubli de règles (cf. III.7.2);
- **définition d'une démarche :**
  - explication informelle de la démarche caractérisée par une séquence de modèles;
  - organisation de la séquence des modèles à utiliser (cf. II.4.6);
  - vérification de l'ordonnancement des modèles par des méta-règles de contrôle;
- **exploitation des règles**: choix de modes d'activation des règles;
- **vérification de la méthode** à l'aide des méta-règles de contrôle.

Mise en œuvre

Un déroulement du fonctionnement de cette interface serait:

1- caractérisation des étapes de la démarche choisie par une séquence de modèles;
2- définition des règles à appliquer dans chaque modèle;
3- spécification de modes d'activation pour chaque règle.

IV.3.2 Base de connaissances de définition d'une méthode particulière

La base de connaissances de définition d'une méthode particulière caractérise un mode de fonctionnement du sous-système de pilotage.

Contenu

Cette base contient deux types principaux de connaissances, à savoir:

- **connaissances de "contrôle"** - Ce sont les règles élémentaires de gestion associées à chaque modèle, l'ordonnancement des modèles et le mode d'activation des règles. Ces connaissances sont issues du sous-système de configuration.
- **connaissances d'"explication"** pour permettre de présenter aux utilisateurs la méthode particulière à suivre dans un projet. Ces connaissances comprennent les résumés généraux des modèles et la caractérisation de la démarche de modélisation à travers des étapes. Cette caractérisation est établie par le sous-système de configuration: ce sont les expressions informelles des règles élémentaires de gestion associées à chacune des étapes et leur ordre d'enchaînement. Notons que cette caractérisation d'explication ne correspond pas au concept courant dans le domaine des systèmes experts où l'"explication" est utilisée pour montrer l'inférence réalisée pour arriver à une conclusion.

Mise en œuvre

Cette base de connaissances de définition d'une méthode particulière est générée par le sous-système de configuration et utilisée en entrée par le sous-système de pilotage.

IV.3.3 Sous-système de pilotage

Le sous-système de pilotage utilise une base de connaissances de définition d'une méthode spécifique (cf. IV.3.2), construite à l'aide du sous-système de configuration, comme un mode de fonctionnement particulier pour guider et vérifier une activité de modélisation.

Ce sous-système est caractérisé par deux composantes principales (cf. figure IV.6):

- base de faits;
- interface de saisie et de contrôle de spécifications.

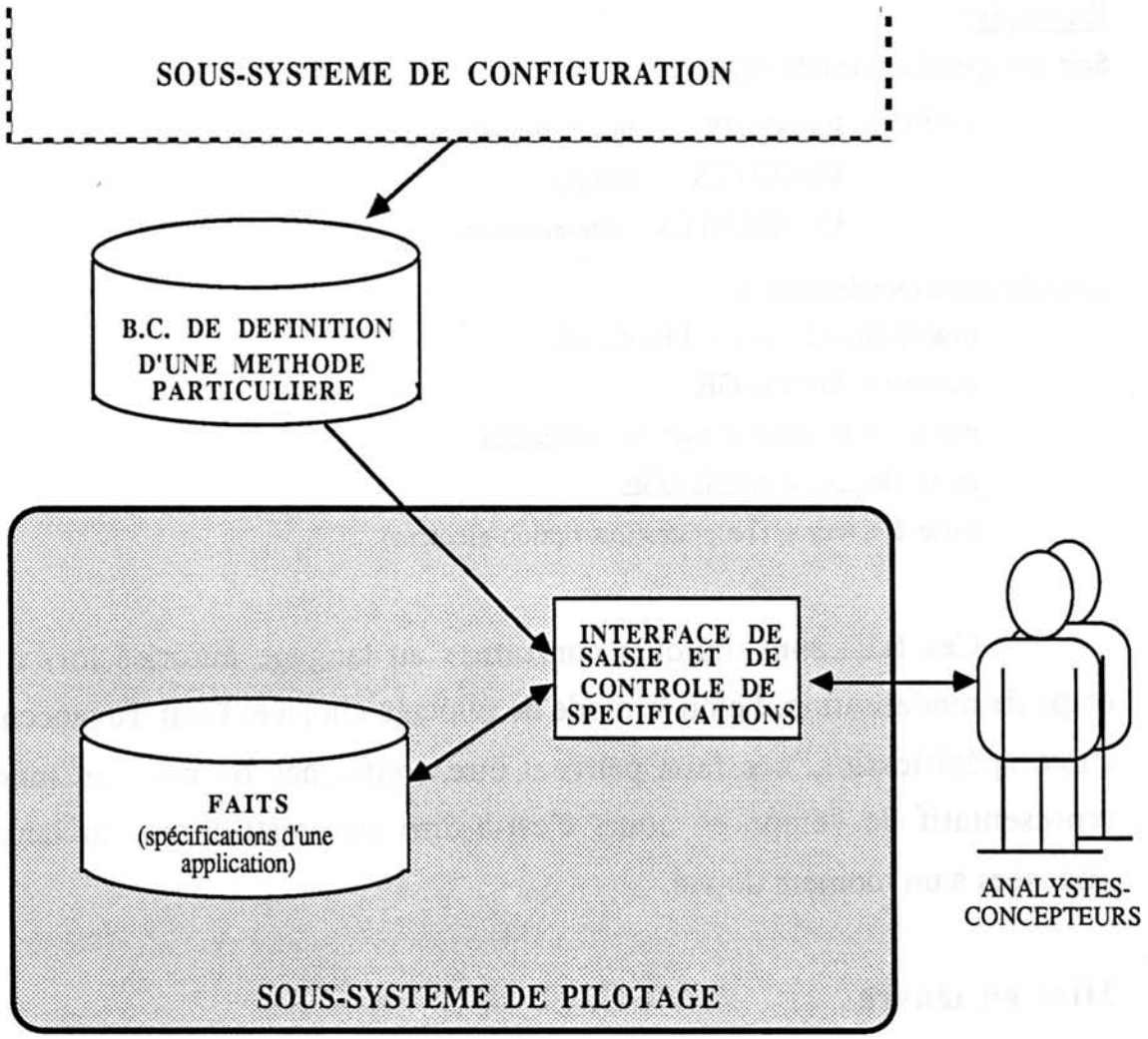


Figure IV.6: Architecture fonctionnelle générale d'un sous-système de pilotage de modélisation

IV.3.3.1 Base de faits

La base de faits rassemble les spécifications d'une application particulière.

Contenu

Cette base contient des faits de deux types (cf. §II.3.2):

- $x \in E$ (entité d'un ensemble);
- $x=f(y)$ ou $x \in F(y)$ (lien monovalué ou multivalué entre deux entités).

Exemple:

Soit une spécification DSL/IDA:

```
DEFINE PROCESS      inscription-électeur;
       RECEIVES     dossier;
       GENERATES    carte-électeur;
```

Les faits correspondants sont:

inscription-électeur ∈ PROCESS
dossier ∈ MESSAGE
dossier ∈ Receives(inscription-électeur)
carte-électeur ∈ MESSAGE
carte-électeur ∈ Generates(inscription-électeur)

Ces faits sont toujours conformes au langage autorisé lors d'une étape de modélisation. Selon le mode de pilotage choisi et l'état d'avancement d'une spécification, ces faits peuvent être conformes ou non au modèle représentatif de l'étape en cours c'est-à-dire aux restrictions du langage imposées à un moment donné.

Mise en œuvre

Une telle base est en évolution permanente durant une activité de modélisation d'une application. Les analystes-concepteurs ajoutent, suppriment ou modifient des spécifications (faits) avec l'objectif d'améliorer la modélisation en cours.

La conformité et la dynamique d'une base de faits spécifique à une application sont gérées par une interface (cf. IV.3.3.2) utilisant la base de connaissances qui caractérise la méthode à suivre (cf. IV.3.2).

IV.3.3.2 Interface de saisie et de contrôle de spécifications

Le fonctionnement du sous-système de pilotage est dirigé par une interface de saisie et de contrôle. Cette interface doit conduire, guider les analystes-concepteurs dans une activité de modélisation. Ainsi, elle doit

permettre l'introduction et la vérification syntaxique des spécifications d'une application particulière selon le langage de spécification autorisé. De même, elle doit contrôler la spécification en fonction de la méthode définie et être capable de fournir des explications sur cette méthode.

Fonctionnalités de l'interface de saisie et de contrôle

Pour atteindre ces objectifs, cette interface doit offrir les fonctionnalités suivantes aux analystes-concepteurs:

- présentation de la méthode particulière à suivre dans un projet;
- acquisition et vérification syntaxique des spécifications d'une application particulière;
- contrôle de ces spécifications selon les règles élémentaires de gestion et les formes d'activation des règles sélectionnées dans le sous-système de configuration;
- présentation de la prochaine étape de modélisation en fonction de la démarche particulière choisie;
- construction de rapports sur le contenu de la base de faits.

En effet, cette interface doit offrir les fonctionnalités existantes dans les outils logiciels actuels de spécification de SI [MAR86, RIG86, CGI87, GAL87, BOD88], avec la souplesse, la personnalisation et le contrôle introduits par le sous-système de configuration.

Mise en œuvre

Dans le cadre d'un pilotage avec **guidage et contrôle permanents**, un déroulement du fonctionnement de cette interface serait une répétition de trois actions:

1- Demande d'informations (spécifications) par le système selon les règles de contrôle de complétude;
2- Spécifications fournies par l'utilisateur;

3- Exécution par le système des règles de vérification de la cohérence concernant la spécification fournie.

Dans un cadre de pilotage avec **contrôle global a posteriori**, un déroulement serait:

1- Introduction libre de spécifications par l'utilisateur;
2- Demande explicite par l'utilisateur pour activer la vérification globale des spécifications par le système;
3- Exécution par le système des règles de vérification de cohérence et de complétude;
4- Modification par l'utilisateur des spécifications en fonction des recommandations du système.

IV.3.4 Conclusion

Cette architecture fonctionnelle hypothétique d'un système de pilotage est le résultat:

- de l'étude de plusieurs méthodes et systèmes commercialisés (cf. §I.2.2);
- de nombreuses discussions avec des chefs de projet et des analystes-concepteurs ainsi que de notre expérience en informatique de gestion;
- et d'une étude conjointe de la méthode IDA [BOD88] et des logiciels associés [IDA86].

Nous avons construit un prototype pour montrer le réalisme de nos propositions et pour prouver que les nouveaux outils logiciels dans ce domaine peuvent s'appuyer sur des règles plus rigoureuses.

IV.4 Expérimentation de l'architecture proposée

Dans cette section nous présentons la structure particulière d'implémentation d'un prototype de système de pilotage. Ce prototype, développé en Turbo-Prolog, a été réalisé avec l'objectif essentiel de **montrer la faisabilité** de nos propositions (cf. §IV.3), sans tenir compte des nombreuses techniques possibles d'implémentation qui restent à évaluer pour construire de tels systèmes.

Ainsi, ce prototype est indépendant des nouveaux environnements de développement de logiciel comme la structure d'accueil Emeraude [BOU86b] ou le projet PCTE (Portable Common Tools Environment) [GAL86a, GAL86b].

Notons que toute réalisation complète conforme à nos propositions devra être précédée d'une comparaison et d'un choix entre techniques d'implémentation.

Dans notre réalisation nous avons privilégié les **structures de données et de contrôle** et nous avons volontairement négligé l'interface utilisateur malgré l'importance de cet aspect dans un système de pilotage. Un prochain travail dans ce domaine devra prendre en compte ce problème d'interface car, aujourd'hui un "bon" système doit offrir plusieurs possibilités de communication avec l'utilisateur [COU88], comme par exemple:

- multifenêtrage pour consulter les spécifications après un message d'erreur;
- utilisation conjointe de graphes et de la souris pour faciliter la saisie et la modification des schémas;
- grille de saisie pour certains compléments d'information;
- menus pour certains utilisateurs et commandes directes pour d'autres utilisateurs plus expérimentés;
- couleurs pour mettre en évidence certains messages ou pour caractériser un aspect particulier.

Néanmoins, à partir de l'expérience acquise lors du développement d'un système d'aide à la spécification de SI [ALV82] et lors de l'implémentation d'un premier prototype de système de pilotage [ALV88], nous avons réalisé ce deuxième prototype selon l'architecture fonctionnelle proposée précédemment.

Lors de la description de cette architecture fonctionnelle générale (cf. §IV.3) nous nous sommes limités à introduire l'aspect dynamique de l'activation des règles, ici nous en proposons une structure particulière d'implémentation. Nous avons choisi comme unité d'activation un groupe de règles que nous appelons par la suite **paquet de règles.**

Unité de déclenchement: le paquet de règles

Les règles qui composent un modèle ne sont pas toujours déclenchées globalement en une seule fois, aussi groupons-nous les règles d'un modèle en plusieurs paquets, avec un déclenchement commun pour chaque paquet. Nous ne parlons pas de déclenchement d'un modèle ou d'une règle mais plutôt de déclenchement d'un paquet de règles. Un tel paquet constitue l'unité de déclenchement. Un ensemble de règles ainsi formé peut contenir une, deux, plusieurs ou, à la limite, toutes les règles d'un modèle.

Notons que le déclenchement d'un paquet de règles ne signifie pas obligatoirement leur activation en parallèle. Dans notre prototype, nous fixons la séquence d'activation des règles de chaque paquet.

Nous reprenons maintenant chaque composante introduite dans l'architecture générale pour en décrire l'implémentation expérimentale. L'annexe B présente certaines parties représentatives du prototype développé.

IV.4.1 Base du langage général de spécification

La base des caractérisations du langage de spécification doit contenir des clauses Prolog qui correspondent à la description du langage de spécification sous forme d'énoncés Z. Cette description est formée essentiellement par:

- les noms des ensembles d'entités utilisés dans le langage de spécification;
- les noms des fonctions décrivant les associations;
- la spécification du type de chaque fonction: monovaluée ou multivaluée, partielle ou totale;
- la définition des arguments de chaque fonction: définition des ensembles et une caractérisation en terme d'obligatoire ou d'optionnel.

Cette base, consultable lors de la vérification de l'expression formelle d'une règle et lors de sa transformation en clauses Prolog, n'est pas intégrée dans le prototype actuel qui ne prend pas en compte ces deux dernières fonctionnalitées (cf. §IV.4.4).

Structure

La base des caractérisations du langage de spécification peut être représentée par des instances des faits Prolog suivants:

```
ensemble_langage_spécif(nom_ensemble).
fonction_langage_spécif(nom_fonction,type_fonction).
source_fonction_langage_spécif(nom_fonction,ordre_argument,
                               nom_ensemble,type_argument).
cible_fonction_langage_spécif(nom_fonction,ordre_argument,
                               nom_ensemble,type_argument).
```

- type_fonction définit le type de la fonction parmi les quatre cas possibles: monovaluée totale, monovaluée partielle, multivaluée totale ou multivaluée partielle;

- **ordre_argument** est le numéro d'ordre de l'ensemble d'entités considéré;
- **type_argument** indique si l'ensemble considéré est obligatoire ou optionnel dans cette fonction.

Exemple: Le schéma Z

PROCESS <<-/-Receives / Received_by-/->> MESSAGE x (SYS-PAR ∪ INTEGER)°

peut être représenté selon la définition ci-dessus par les faits:

```
ensemble_langage_spécif(process).
ensemble_langage_spécif(message).
ensemble_langage_spécif(sys_par).
ensemble_langage_spécif(integer).
fonction_langage_spécif(receives,4).
fonction_langage_spécif(received_by,4).
source_fonction_langage_specif(receives,1,process,obligatoire).
cible_fonction_langage_specif(receives,2,message,obligatoire).
cible_fonction_langage_specif(receives,3,sys_par,optionnel).
cible_fonction_langage_specif(receives,3,integer,optionnel).
source_fonction_langage_specif(received_by,1,message,obligatoire).
source_fonction_langage_specif(received_by,2,sys_par,optionnel).
source_fonction_langage_specif(received_by,2,integer,optionnel).
cible_fonction_langage_specif(received_by,3,process,obligatoire).
```

IV.4.2 Base des règles élémentaires de gestion

Cette base des règles élémentaires de gestion contient l'ensemble des restrictions du langage de spécification utilisées pour la définition formelle d'une méthode.

Le stockage choisi pour cette base de règles respecte la spécification rigoureuse et systématique des règles (cf. §III.3). A chaque règle sont associées cinq clauses Prolog dont nous donnons ci-dessous la forme générale des têtes; les corps de ces clauses étant spécifiques de chaque règle

(cf. exemple suivant). La composante "sous-schéma" n'est pas prise en compte.

règle(code_règle)
expression_informelle(code_règle,expression_informelle)
expression_formelle(code_règle,arguments)
diagnostic(code_règle,arguments)
recommandation(code_règle,arguments)

- **code_règle:** code discriminant identifiant la règle.

- **code_type_règle:** caractérisation de cette règle selon la typologie (cf. §III.4, §III.5 et annexe B).

- **arguments:** arguments éventuels de la règle pour spécifier les noms des entités concernées par l'erreur.

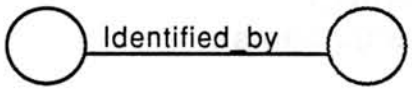
En plus de ces clauses, cette base contient des informations spécifiques pour la vérification des méthodes par des méta-règles de contrôle. Ce sont trois clauses qui indiquent pour chaque règle élémentaire de gestion: le type, les ensembles d'entités et les fonctions utilisés.

type_règle(code_règle,code_type_règle)
ensemble_utilisé(code_règle,code_ensemble)
fonction_utilisée(code_règle,code_fonction)

Exemple: Soit la règle élémentaire de gestion

• expression informelle:
Chaque entité doit avoir au moins un identifiant.

• sous-schéma:



• expression formelle:
$\forall e \in ENTITY \; . \; Identified\_by(e) \neq \phi$

• diagnostic:
Absence d'identifiant de l'entité "X".

• recommandation:
Il faut utiliser au moins 1 clause "IDENTIFIED BY ..." pour spécifier un identifiant de l'entité.

Considérant que c'est la 118ème règle de la base, les clauses Prolog correspondantes sont:

```
règle(r118).

expression_informelle(r118,"Chaque entité doit avoir au moins un identifiant.").

expression_formelle(r118,E)  :-
        entity(E),
        not(identified_by(E,Identifiant)).

diagnostic(r118,E)  :-
        write("Absence d'identifiant de l'entité'",E,"'."),nl.

recommandation(r118)  :-
        write("Il faut utiliser au moins une clause IDENTIFIED BY..."),
        write(" pour spécifier un identifiant de l'entité."),nl,nl.

type_règle(r118,230).

ensemble_utilisé(r118,entity).

fonction_utilisée(r118,identified_by).
```

Dans notre prototype nous avons écrit directement pour chaque règle son expression formelle sous forme négative pour expliciter que le diagnostic et la recommandation associés suivent la validité d'une telle formule.

Une clause **vérification_règle(code_règle)** constitue une structure générale d'exécution des règles. Elle est sous la forme générale:

```
vérification_règle(code_règle)  :-
        expression_formelle(code_règle,arguments),        % (1)
        diagnostic(code_règle,arguments),                 % (2)
        recommandation(code_règle,arguments),             % (3)
        fail.                                             % (4)
vérification_règle(code_règle)  :-  true.                 % (5)
```

où (1) Vérifie si l'expression formelle de la règle est satisfaite. En cas d'échec, "arguments" contient éventuellement les noms des entités concernées par l'erreur.

(2) Communique le diagnostic associé à la règle.

(3) Communique la recommandation associée.

(4) Force le "backtracking" pour vérifier s'il y a d'autres cas d'échec.

(5) Condition de fin d'exécution de cette règle.

Exemple: La règle de l'exemple précédent est représentée par la clause:

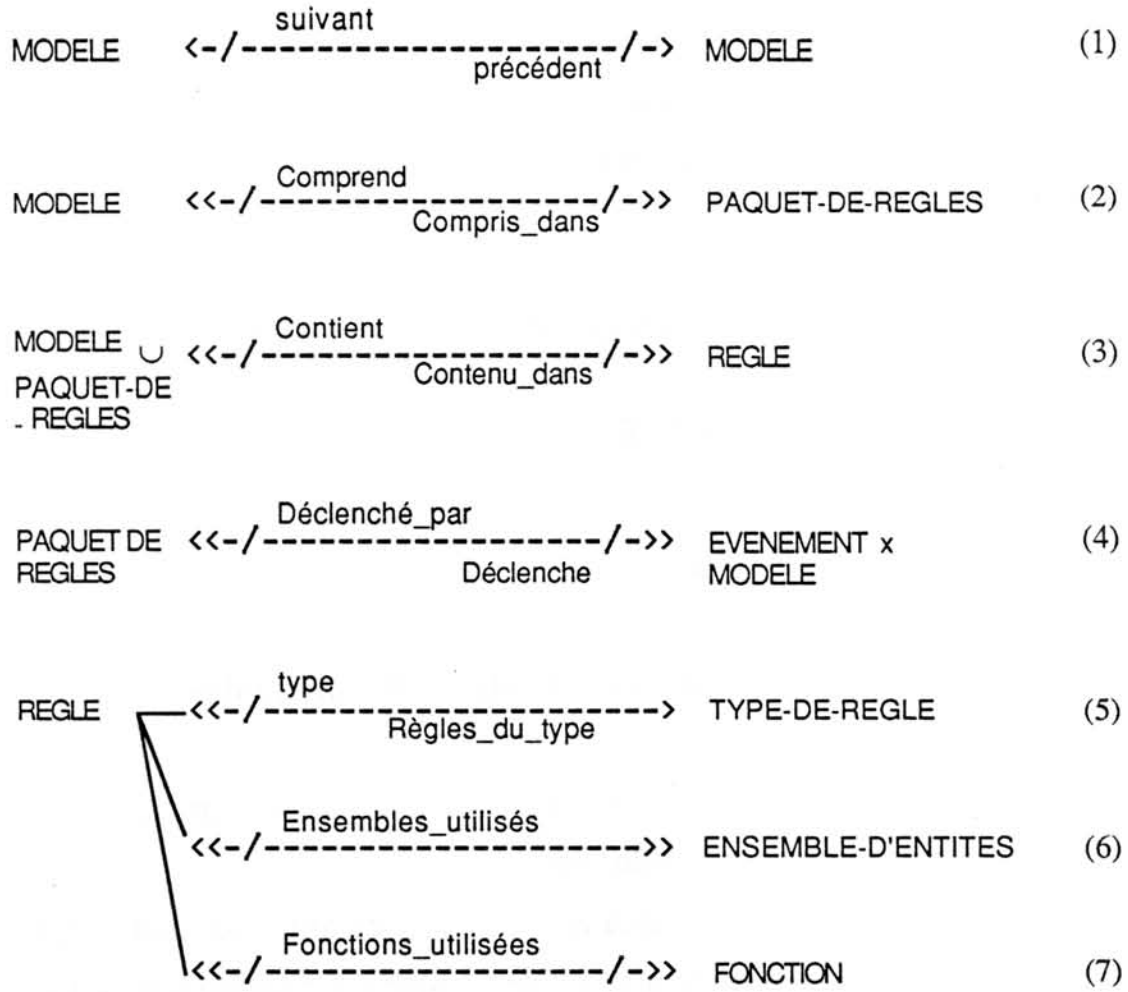
```
vérification_règle(r118)  :-
        expression_formelle(r118,E),
        diagnostic(r118,E),
        recommandation(r118),
        fail.
vérification_règle(r118)  :-  true.
```

IV.4.3 Base des méta-règles de contrôle de méthodes

La base des méta-règles de contrôle contient l'ensemble des règles utilisées pour former une méthode "saine".

Dans cette structure particulière d'implémentation, les méta-règles de contrôle doivent prendre en compte le mécanisme d'activation des règles et la séquence de modèles qui caractérisent une méthode. Ainsi, il faut compléter le schéma général présenté en §III.7.2.1 pour inclure les notions de paquet de règles, de déclenchement d'un paquet de règles par un événement (cf. §IV.4) et de caractérisation d'une méthode par une séquence de modèles (cf. §II.4.6).

Avec ces informations supplémentaires le schéma général devient:



(1) ordonnancement des modèles

(2) attachement d'un paquet de règles à un modèle

(3) attachement des règles à un modèle ou à un paquet de règles

(4) déclenchement d'un paquet de règles par un événement dans le cadre d'un modèle

(5) caractérisation d'une règle

(6) ensembles d'entités utilisés dans une règle

(7) fonctions utilisées dans une règle

A partir de cette description, en plus des méta-règles déjà introduites au chapitre III (cf. §III.7.2), nous pouvons formuler des méta-règles complémentaires:

MR1: Une méthode doit avoir au moins un modèle.
card(MODELE) ≥ 1

MR2: Chaque modèle doit comprendre au moins un paquet de règles
$\forall m \in$ MODELE . Comprend(m) $\neq \phi$

MR3: Chaque paquet de règles doit contenir au moins une règle.
$\forall p \in$ PAQUET-DE-REGLES . Contient(p) $\neq \phi$

MR4: Chaque règle d'un modèle doit être contenue dans un paquet de règles compris dans ce modèle.
$\forall r \in$ REGLE . $\forall m \in$ MODELE . $\exists p \in$ PAQUET-DE-REGLES .
$(r \in \text{Contient}(m) \Rightarrow r \in \text{Contient}(p) \wedge p \in \text{Comprend}(m))$

MR5: Chaque paquet de règles d'un modèle doit être déclenchable dans ce modèle.
$\forall m \in$ MODELE . $\forall p \in$ PAQUET-DE-REGLE . $\exists e \in$ EVENEMENT .
$p \in \text{Comprend}(m) \Rightarrow p \in \text{Déclenche}(e,m)$

MR6: Chaque règle d'un paquet de règles attaché à un modèle doit être contenu dans ce modèle.
$\forall r \in$ REGLE . $\forall p \in$ PAQUET-DE-REGLES . $\forall m \in$ MODELE .
$(r \in \text{Contient}(p) \wedge p \in \text{Comprend}(m) \Rightarrow r \in \text{Contient}(m))$

MR7: Chaque paquet de règles doit être attaché à au moins un modèle.
$\forall p \in$ PAQUET-DE-REGLES . $\exists m \in$ MODELE .
$p \in \text{Comprend}(m)$

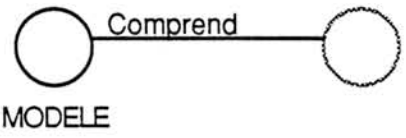
Dans notre prototype, les méta-règles de contrôle ont une représentation analogue à des règles élémentaires de gestion. Les têtes des clauses sont:

```
méta_règle(code_méta_règle)
expression_informelle_mr(code_méta_règle,expr_informelle)
expression_formelle_mr(code_méta_règle,arguments)
diagnostic_mr(code_méta_règle,arguments)
recommandation_mr(code_méta_règle,arguments)
```

Exemple: Soit la méta-règle MR2 présentée ci-dessus:

• expression informelle:
Chaque modèle doit contenir au moins un paquet de règles

• sous-schéma:



• expression formelle:

$\forall m \in$ MODELE . Comprend(m) $\neq \phi$

• diagnostic:

Absence de paquets de règles du modèle "X".

• recommandation:

Spécifier les paquets de règles qui forment ce modèle.

Clauses Prolog correspondantes:

```
méta_règle(mr2).

expression_informelle_mr(mr2,"Chaque modèle doit contenir au moins un paquet de
règles.").

expression_formelle_mr(mr2,M) :-
        modèle(M),
        not(comprend(M,Code_règle)).

diagnostic_mr(mr2,M) :-
        write("Absence de paquets de règles  du modèle '",M,"'."),nl.

recommandation_mr(mr2) :-
        write("Spécifier les paquets de règles qui forment ce modèle"),nl,nl.
```

La clause **vérif_méta_règle(Code_méta_règle)** qui exécute une méta-règle est sous la forme générale:

```
vérif_méta_règle(code_méta_règle)  :-
        expression_formelle_mr(code_méta_règle,arguments), % (1)
        diagnostic_mr(code_méta_règle,arguments),          % (2)
        recommandation_mr(code_méta_règle,arguments),      % (3)
        fail.                                              % (4)
vérif_méta_règle(code_méta_règle)  :-  true.               % (5)
```

où (1) vérifie si l'expression formelle de la règle est satisfaite. En cas d'échec, "arguments" contient éventuellement les objets non conformes.

(2) communique le diagnostic associé à la règle

(3) communique la recommandation associée

(4) force le backtracking

(5) fin de vérification de la règle

Exemple: La méta-règle MR2 de l'exemple précédent est représentée par la clause

```
vérif_méta_règle(mr2)  :-
        expression_formelle_mr(mr2,M),
        diagnostic_mr(mr2,M),
        recommandation_mr(mr2),
        fail.
vérif_méta_règle(mr2)  :-  true.
```

IV.4.4 Interface de saisie et de mise à jour du langage et des règles

L'interface de saisie et de mise à jour du langage et des règles gère les trois bases qui sont utilisées pour la définition précise d'une méthode: la base du langage général de spécification, la base des règles élémentaires de gestion et la base des méta-règles de contrôle.

Cette interface n'est pas intégrée dans notre prototype. Néanmoins, une première étude [MAR88] permet d'en établir les caractéristiques principales.

L'interface est structurée selon trois modules (cf. figure IV.7):

- module de dialogue et de contrôle;
- module de vérification d'un énoncé Z;
- module de transformation d'énoncés Z en clauses Prolog.

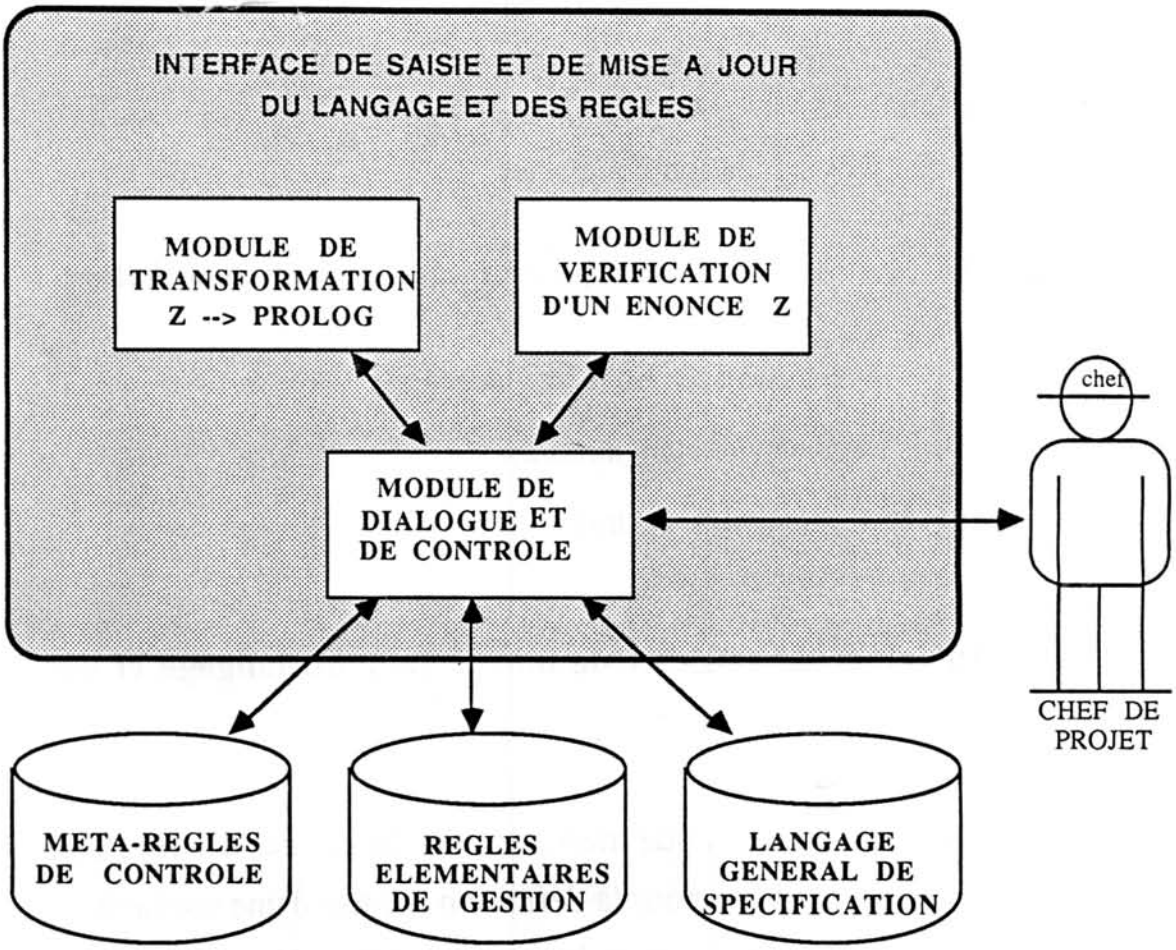


Figure IV.7: Structure de l'interface de saisie et de mise à jour du langage et des règles

IV.4.4.1 Module de dialogue et de contrôle

Le module de dialogue et de contrôle permet l'ajout, la modification et la suppression des trois types d'énoncés: langage de spécification, règles élémentaires de gestion et méta-règles. Il permet aussi la construction de rapports sur le contenu des bases de connaissances associées à ces trois types d'énoncés.

Ce module active le module de vérification d'un énoncé Z et le module de transformation d'énoncés Z en clauses Prolog.

IV.4.4.2 Module de vérification d'un énoncé Z

Le module de vérification d'un énoncé Z permet la vérification de trois types d'expressions Z: du langage de spécification, des règles élémentaires de gestion et des méta-règles de contrôle.

La vérification de l'expression Z du **langage de spécification** correspond à une vérification syntaxique usuelle.

La vérification des **règles élémentaires de gestion** peut être vue en deux parties:

- **syntaxique**: vérifier si la formule respecte la grammaire présentée au chapitre II (cf. §II.3.3);
- **sémantique**: vérifier si les ensembles et les fonctions utilisés dans la formule respectent l'expression Z du langage de spécification.

La vérification des **méta-règles** de contrôle est analogue à la vérification des règles élémentaires de gestion, mais le schéma Z de référence est pré-défini et correspond au schéma proposé en §IV.4.3.

IV.4.4.3 Module de transformation d'énoncés Z en clauses Prolog

Ce module réalise, d'une part, la traduction des énoncés qui correspondent à la représentation du langage de spécification et, d'autre part, la transformation des expressions formelles des règles et des méta-règles.

La traduction des énoncés Z qui décrivent la représentation du langage de spécification est faite directement. Les faits Prologs générés sont (cf. §IV.4.1):

```
ensemble_langage_spécif(nom_ensemble).
fonction_langage_spécif(nom_fonction,type_fonction).
source_fonction_langage_spécif(nom_fonction,ordre_argument,
                               nom_ensemble,type_argument).
cible_fonction_langage_spécif(nom_fonction,ordre_argument,
                               nom_ensemble,type_argument).
```

La transformation des règles élémentaires de gestion génère les clauses présentées ci-dessus (cf. §IV.4.2):

```
règle(code_règle)
expression_informelle(code_règle,expression_informelle)
expression_formelle(code_règle,arguments)
diagnostic(code_règle,arguments)
recommandation(code_règle,arguments)
type_règle(code_règle,code_type_règle)
ensemble_utilisé(code_règle,nom_ensemble)
fonction_utilisée(code_règle,nom_fonction)
```

La transformation des méta-règles de contrôle génère les clauses suivantes (cf. §IV.4.3):

```
méta_règle(code_méta_règle)
expression_informelle_mr(code_méta_règle,expr_informelle)
expression_formelle_mr(code_méta_règle,arguments)
diagnostic_mr(code_méta_règle,arguments)
recommandation_mr(code_méta_règle,arguments)
```

IV.4.5 Interface de configuration de méthodes

L'interface de configuration de méthodes est utilisée pour la définition précise d'une méthode à suivre dans un projet particulier.

Cette interface est structurée selon deux modules (cf. figure IV.8): un module de dialogue et de configuration et un module de vérification.

Le module de dialogue et de configuration permet de spécifier:

- les modèles qui forment la méthode;
- l'ordonnancement des modèles;
- les règles élémentaires de gestion associées à chaque modèle;
- les paquets de règles de chaque modèle (cf. IV.4);
- le déclenchement de chaque paquet de règles.

Le module de vérification certifie qu'une méthode définie respecte les méta-règles de contrôle.

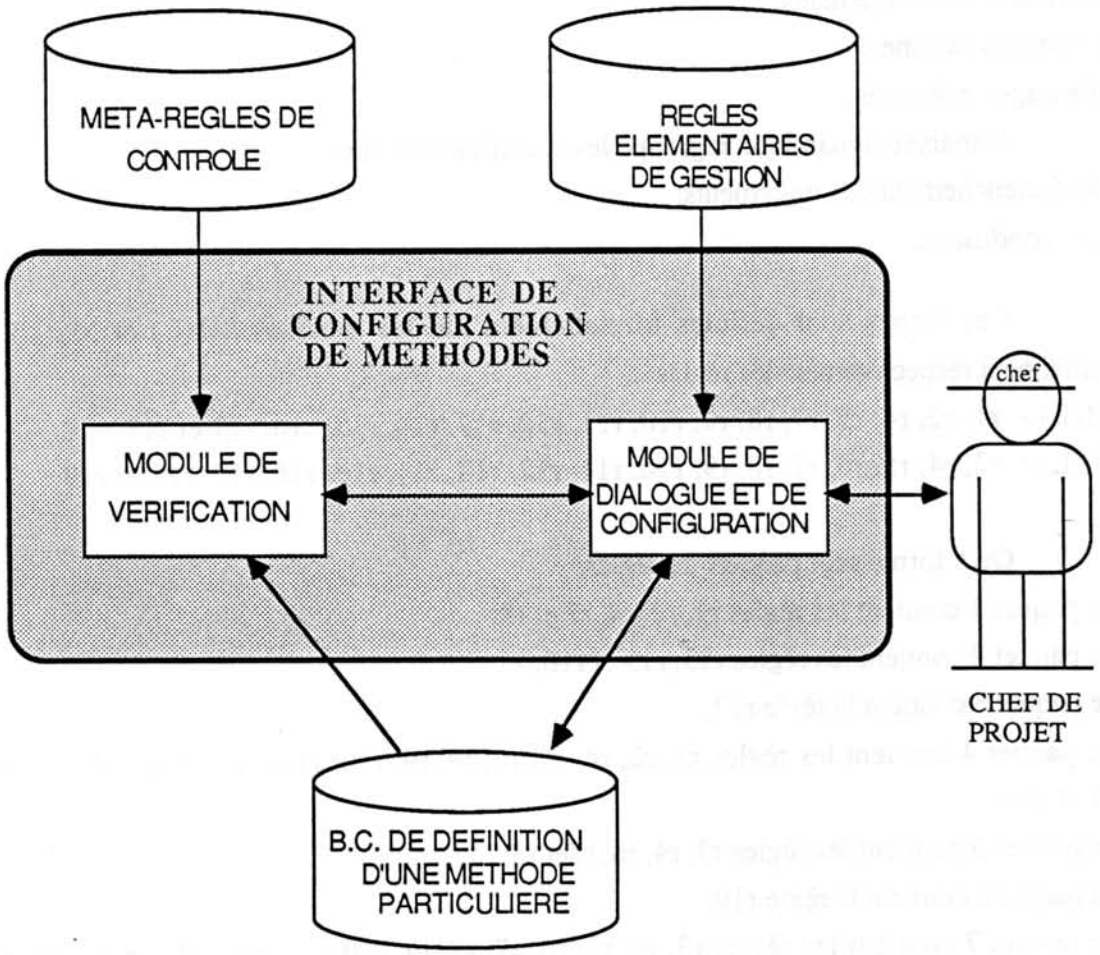


Figure IV.8: Structure d'une interface de configuration de méthodes

IV.4.5.1 Module de dialogue et de configuration de méthodes

Pour décrire une méthode on doit définir les étapes de la démarche et leur enchaînement. Pour chaque étape on doit spécifier d'une part, les règles élémentaires de gestion à utiliser pour vérifier la complétude et la cohérence des spécifications et d'autre part, les conditions d'activation de ces règles.

Exemple: On doit faire la spécification d'une application A avec le sous-ensemble de DSL/IDA défini en III.6 et que nous appelerons langage L. Pour ce langage L nous avons formulé dix-neuf règles élémentaires de gestion codées r1, r2,..., r19 (cf. III.6). Dans le cadre de la méthode choisie, la démarche comporte deux étapes appelées, analyse statique et

analyse dynamique.

L'analyse statique comporte trois types de spécifications:

- hiérarchie des traitements;
- interfaces externes;
- messages échangés.

L'analyse dynamique regroupe les spécifications sur:

- le déclenchement des traitements;
- les conditions.

Ces étapes sont définies formellement par les modèles M1(L) et M2(L) qui contiennent respectivement les règles:

M1(L) - r1, r2, r4, r5, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17 et r18.

M2(L) - r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18 et r19.

On a formé sept paquets de règles:

- le paquet 1 contient les règles r1, r2, r4, r7 et r8;
- le paquet 2 contient les règles r13, r15 et r16;
- le paquet 3 contient la règle r17;
- le paquet 4 contient les règles r1, r2, r4, r5, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17 et r18;
- le paquet 5 contient les règles r3, r4, r6, r7et r8;
- le paquet 6 contient la règle r19;
- le paquet 7 contient les règles r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18 et r19.

Le tableau ci-dessous complète la description de la méthode:

MODELE (ETAPE)	DECLENCHEMENT	
	PAQUET	EVENEMENT DECLENCHEUR
analyse statique	paquet1	fin_spécif_processus
	paquet2	fin_spécif_message
	paquet3	fin_spécif_interface
	paquet4	fin_spécif_modèle
analyse dynamique	paquet5	fin_spécif_processus
	paquet2	fin_spécif_message
	paquet3	fin_spécif_interface
	paquet6	fin_spécif_condition
	paquet7	fin_spécif_modèle

Par exemple, les règles r13,r15 et r16 regroupées dans le paquet 2 sont activées à chaque fin de définition d'un message pour contrôler que:

- r13: ce message n'est pas conjointement reçu et généré par des interfaces;
- r15: ce message est reçu par au moins un processus ou une interface;
- r16: ce message est généré par au moins un processus ou une interface;

Notons que:

- une même règle peut être attachée à plusieurs modèles;
- un paquet de règles peut avoir plusieurs événements déclencheurs;
- un même événement peut déclencher différents paquets de règles à différentes étapes de la démarche.

Dans notre implémentation nous utilisons deux catégories d'événements:

- fermeture d'un écran de saisie et de mise à jour de spécifications;
- choix dans un menu d'une option de vérification a posteriori.

Le module de dialogue et de configuration génère les faits Prolog suivants à partir des informations fournies par le chef de projet:

- pour spécifier le nom d'une méthode particulière:
  méthode(code_méthode).
  Exemple: méthode(exemple).

- pour spécifier le nom d'un modèle:
  modèle(code_modèle).
  Exemple: modèle(analyse statique).

- pour spécifier le nom d'un paquet de règles:
  paquet(code_paquet).
  Exemple: paquet(paquet1).

- pour associer un paquet de règles à un modèle:
  comprend(code_modèle,code_paquet).
  Exemple: comprend(analyse statique,paquet1).

- pour associer une règle élémentaire de gestion à un modèle ou à un paquet de règles:

  contient(code_modèle_ou_code_paquet,code_règle).

  Exemple: contient(analyse statique,r1).

- pour définir le premier modèle de la démarche:

  modèle_initial(code_modèle).

  Exemple: modèle_initial(analyse statique).

- pour spécifier que le modèle B suit le modèle A dans la démarche choisie:

  suivant(code_modèle_A,code_modèle_B)

  Exemple: suivant(analyse statique, analyse dynamique).

- pour spécifier le nom d'un événement:

  événement(code_événement).

  Exemple: événement(fin_spécif_processus).

- pour spécifier qu'un événement déclenche un paquet de règles dans un modèle:

  déclenche(code_événement,code_modèle,code_paquet).

  Exemple: déclenche(fin_spécif_processus,analyse statique,paquet1).

IV.4.5.2 Module de vérification de méthodes

Ce module assure la vérification des méthodes en cours de définition. Cette vérification est réalisée par l'exécution des méta-règles de contrôle (cf. §IV.4.3). Ce module est basé sur la clause Prolog suivante:

```
vérification_de_méthode  :-
        méta_règle(Code_méta_règle),          % (1)
        vérif_méta_règle(Code_méta_règle),    % (2)
        fail.                                 % (3)
vérification_de_méthode  :-  true.            % (4)
```

où (1) sélectionne une méta_règle de contrôle

(2) exécute une méta_règle

(3) force le "backtracking"

(4) fin des vérifications

IV.4.6 Base de connaissances de définition d'une méthode particulière

Cette base contient les clauses Prolog générées par les différents modules décrits ci-dessus et qui définissent précisément une méthode de modélisation. Ce sont des instances des clauses suivantes:

```
contient(code_modèle_ou_code_paquet,code_règle)
déclenche(code_événement,code_modèle,code_paquet)
diagnostic(code_règle,arguments)
événement(code_événement)
expression_formelle(code_règle,arguments)
expression_informelle(code_règle,expr_informelle)
méthode(code_méthode)
modèle(code_modèle)
modèle_initial(code_modèle)
paquet(cod_paquet)
recommandation(code_règle,arguments)
règle(code_règle)
suivant(code_modèle_A,code_modèle_B)
type_règle(code_règle,code_type_règle)
vérification_règle(code_règle)
```

L'annexe B présente la base de connaissances de définition de la méthode spécifiée dans l'exemple de la section IV.4.5.1.

IV.4.7 Sous-système de pilotage

Le sous-système de pilotage utilise les clauses caractérisées ci-dessus (cf. IV.4.6) pour contrôler et diriger une activité de modélisation. Il contient une base de faits et une interface de saisie et de contrôle de spécifications.

IV.4.7.1 Base de faits

Cette base, qui rassemble les spécifications d'une application particulière, contient des faits sous la forme générale

nom_ensemble(nom_entité).

nom_fonction(arguments).

Exemple: Dans le cadre d'une base de faits associée à une spécification utilisant le sous-ensemble du langage DSL/IDA énoncé en Z

PROCESS $\langle\!\langle\!-\!/\!-\!\overset{\text{Receives}}{\underset{\text{Received\_by}}{\text{-----------}}}\!/\!-\!\rangle\!\rangle$ MESSAGE x (SYS-PAR $\cup$ INTEGER)°

la base de faits contient des instances des faits Prolog

```
process(nom_processus).
message(nom_message).
sys_par(nom_sys_par).
receives(nom_processus,nom_message,nombre_messages).
```

comme par exemple:

```
process(inscription_électeur).
process(expédition).
message(dossier).
message(carte_électeur).
receives(inscription_électeur,dossier,1).
```

IV.4.7.2 Interface de saisie et de contrôle de spécifications

Cette interface assure plusieurs fonctions classiques (cf.IV.3.3.2) liées à la convivialité de l'utilisation du système: aide pour contrôler le déroulement d'une méthode, saisie et mise à jour de spécifications, ...

Nous nous limitons ici à la présentation du mécanisme de vérification des spécifications. Ce mécanisme de vérification est contrôlé par la clause **activation(Evénement)** décrite ci-dessous. Cette clause est exécutée chaque fois qu'un événement déclencheur de règles de vérification est instancié. Par exemple, le choix d'une option dans un menu, l'introduction

d'une spécification par un analyste-concepteur ou la fermeture d'une fenêtre à l'écran peuvent conduire à l'exécution de cette clause.

```
activation(E) :-
        modèle_courant(M),                      % (1)
        déclenche(E,M,Code_paquet),             % (2)
        contient(Code_paquet,Code_règle),       % (3)
        vérification_règle(Code_règle),         % (4)
        fail.                                   % (5)
activation(E) :- true.                          % (6)
```

où (1) sélectionne le modèle courant

(2) sélectionne un paquet de règles déclenché par l'événement E dans le cadre du modèle M

(3) sélectionne une règle du paquet

(4) exécute cette règle élémentaire de gestion

(5) force le "backtracking"

(6) condition d'arrêt

IV.5 Conclusion

Le prototype développé (cf. §IV.4) selon l'architecture fonctionnelle proposée (cf. §IV.3) permet de valider les choix initiaux établis en termes de:

- représentation formelle d'une méthode (cf. §II);
- description rigoureuse des contraintes de cohérence et de complétude de spécifications (cf.§III);
- personnalisation du mode de pilotage d'une modélisation (cf. §I.3).

Le chapitre suivant analyse les limites de cette expérimentation et de ces propositions d'architecture fonctionnelle.

CHAPITRE V

BILAN ET PERSPECTIVES

BILAN ET PERSPECTIVES

V.1 Formalisation de notions fondamentales

Après une étude des problèmes associés au processus de modélisation des SI et une présentation brève de certaines méthodes représentatives de ce domaine, nous avons énoncé les avantages d'une approche système expert pour le pilotage de la modélisation.

Considérant ensuite, que l'absence de représentation formelle des méthodes constitue une grande difficulté pour étudier le pilotage, nous avons présenté et utilisé un formalisme précis pour définir rigoureusement et complétement une méthode de modélisation des SI. Nous avons en particulier défini les notions de cohérence et de complétude des spécifications d'un SI exprimées dans ce formalisme.

Ces notions sont matérialisées par des formules logiques que nous appelons règles élémentaires de gestion. Ces règles ont été caractérisées selon leur utilité et leur portée dans le cadre d'une typologie où chaque type de règle est décrit systématiquement par cinq composantes: sous-schéma, expression formelle, expression informelle, diagnostic et recommandation. L'utilisation de cette typologie a été illustrée dans le cadre d'une méthode particulière employant comme langage de base un sous-ensemble de DSL/IDA. Nous nous sommes rendus compte que cette typologie est aussi très utile pour poser des questions pertinentes pour affiner une démarche.

Nous avons volontairement minimisé le nombre d'exemples de règles dans ce document pour ne pas en alourdir la présentation. Cependant, nous avions exprimé préalablement de très nombreuses règles sur l'ensemble du langage de spécification DSL/IDA. Le tableau ci-dessous présente une synthèse quantitative de ces règles pour chaque modèle IDA. Notons que le modèle de structuration des traitements a été le seul modèle pour lequel nous avons étudié aussi des règles associées à une démarche.

modèle IDA	nombre d'ensembles d'entités	nombre d'associations	nombre de règles
structuration des informations	10	17	51
structuration des traitements	2	2	16
dynamique des traitements	6	12	16
statique des traitements	6	7	30
ressources	5	12	13

Nous avons décrit le problème de dépendances entre règles; mais les caractéristiques particulières des groupes de règles manipulées rendent toute preuve mathématique délicate. Aussi avons-nous choisi une approche informelle de validation des ensembles de règles par des experts, combinée avec un contrôle par des méta-règles.

V.2 Un système de pilotage d'une modélisation

Finalement, nous sommes revenus à notre objectif principal, le pilotage d'une modélisation de SI. Notre système doit diriger la constitution d'une "bonne" spécification par rapport à la méthode choisie et non l'obtention des bons modèles vis à vis de la réalité à modéliser.

Nous pouvons éclairer cette idée en faisant un parallèle avec la programmation: qu'est-ce qu'un "bon" programme?

Historiquement, on est passé par une première étape où l'on ne considérait un "bon" programme que par rapport à une "bonne" grammaire et par conséquence par rapport à un "bon" compilateur avant d'aborder une deuxième étape où l'on commence à savoir valider sémantiquement un "bon" programme selon d'autres critères comme par exemple sa terminaison, ou sa complexité.

Notre apport à la modélisation des SI est d'aider la transition entre ces deux étapes: passer d'une correction syntaxique à une correction sémantique; cette correction sémantique n'est liée qu'à la sémantique formelle du modèle défini et n'inclue pas la sémantique associée à la réalité à modéliser.

Une méthode de modélisation ne peut pas être unique, ni rigide: plusieurs auteurs ont abordé le problème de la diversité des types de systèmes et ont constaté qu'ils ont besoin de moyens très différents pour leur construction [LEH81, LAN86]. Ainsi, notre architecture fonctionnelle d'un système de pilotage intègre un sous-système de configuration pour permettre la définition précise d'une méthode personnalisée.

Cette architecture fonctionnelle permet notamment:

• une formalisation des conditions de cohérence et de complétude d'une spécification;

• une formalisation d'une démarche personnalisée de modélisation;

• l'utilisation des mêmes expressions formelles pour définir une méthode et pour vérifier si une spécification (une base de faits) est conforme à cette méthode (cette définition). La cohérence entre la définition et la vérification est ainsi assurée.

Pour montrer la faisabilité de nos propositions nous avons développé en Prolog un prototype d'un tel système de pilotage . Il permet la définition et le pilotage d'une méthode personnalisée de modélisation basée sur un sous-ensemble du langage DSL/IDA. Le sous-système de configuration permet en particulier:

• le choix du nombre d'étapes de la démarche;

• la définition de plusieurs démarches en fonction des règles de vérification sélectionnées pour chaque étape;

• la définition de plusieurs modes de pilotage en fonction du choix de la forme d'activation des règles.

V.3 Conclusion

Arrivé au terme de cette thèse, on ne se trouve ni en présence d'une théorie de la modélisation des SI, ni en présence d'une réalisation d'un système opérationnel. Cette remarque indique dans quelles directions orienter la suite de ce travail.

Nous pouvons projeter ce travail vers quelques prolongements.

• **Intégration d'une interface de gestion du langage et des règles** - Il est en particulier nécessaire d'intégrer l'interface de saisie et de mise à jour du langage et des règles (cf. §IV.4.4) et d'utiliser la base du langage général de spécification pour faciliter l'expression des règles et en assurer un contrôle.

• **Aide à l'utilisateur** - A chaque règle introduite dans la base des règles et à chaque étape de la démarche choisie l'expert a joint un texte d'explication en langage naturel. Ce choix est basé sur l'idée que seul, celui qui définit une règle ou une étape, connaît réellement sa signification et sait pourquoi il l'introduit. A tout moment le système de pilotage "sait" ce que l'utilisateur fait et l'écran d'aide ou d'explication qui apparaît à la demande doit se rapporter à l'action en cours.

• **Coopération entre analystes** - La spécification d'une application doit pouvoir être réalisée par plusieurs analystes simultanément ce qui implique le contrôle de concurrence d'accès à la base de faits des spécifications. Le traitement et le contrôle de versions multiples de spécifications d'une application devraient être pris en compte.

• **Utilisation du prototype** - Il est nécessaire de faire utiliser le prototype actuel par des chefs de projet et des analystes-concepteurs sur plusieurs applications réelles pour en faire une évaluation de ses capacités, de son temps de réponse, de sa facilité d'utilisation et de la qualité des spécifications obtenues .

• **Ouverture vers d'autres méthodes** - Il faudrait, également, expérimenter l'architecture proposée sur d'autres méthodes de modélisation comme Merise, Remora ou SSA.

• **Evaluation de techniques de développement** - Il est indispensable d'étudier et de comparer d'autres techniques de développement comme des générateurs de systèmes experts ou des environnements basés sur des langages orientés objets.

C'est seulement après ces applications et ces extensions que l'on pourra véritablement apprécier la validité des propositions faites et faire évoluer et améliorer ces premières idées et cette expérimentation dans d'autres directions comme par exemple, en adaptant cette architecture fonctionnelle pour prendre en compte les connaissances du domaine spécifique de l'application et les connaissances générales d'analyse et conception.

Néanmoins, nous pensons avoir défini un cadre de travail pour maintenant mieux approfondir les problèmes liés au pilotage d'une modélisation des SI.

Sans avoir la prétention que ce rapport couvre et traite convenablement tous les problèmes, nous pensons avoir montré qu'il existait une approche différente pour offrir de nouveaux outils dans ce domaine.

Cette approche simple et rigoureuse offre déjà de réels services. Toutefois, il ne faudra pas oublier la nature des problèmes auxquels on s'intéresse, qui font appel conjointement à l'expérience variée et complexe des individus et à leur formation à des méthodes spécialisées. Autant dire qu'un souci pédagogique constant devra prédominer dans la définition de nouveaux produits de cette classe si l'on veut diffuser l'expertise acquise.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[ABR74] ABRIAL J.R.: Data Semantics.
Data Base Management , Klimbie J.W. et al eds, North Holland, Amsterdam, 1974.

[ABR78] ABRIAL J.R.: *Manuel du langage Z.*
Notes Z1 à Z15 non publiées, EDF, Paris, 1978.

[AHO86] AHO A.V., SETHI R., ULMAN J.D.: *Compilers: principles, techniques and tools.*
Addison-Wesley, Reading,1986.

[ALV82] ALVARES L.O.: Especificação de sistemas: um software de apoio.
IX° Conferencia Latino-Americana de Informática , Lima, 1982.

[ALV88] ALVARES L.O., BODART F.: Une architecture fonctionnelle pour le pilotage de la modélisation des systèmes d'information.
Congrès INFORSID 88 , La Rochelle, 7-10 juin 1988.

[AYE87] AYEL M.: *Détection d'incohérences dans les bases de connaissances: SACCO.*
Thèse d'Etat, Université de Savoie, Chambéry, 1987.

[BER86] BERNIER J., BOISSONNET V., CORNILY J., ESCULIER C., JOUVE M., KOULOUMDJIAN J., KOSCHELEFF C., LEPAPE B., MOULINE A., OGONOWSKI A.: Convergence des bases de données et des systèmes experts.
Modèles et Bases de Données, AFCET Informatique, n° 5, décembre 1986.

[BOA84] BOAR H.: *Application prototyping - a requirements definition strategy for the 80s.*
John Wiley & Sons Inc, New York, 1984.

[BOD86] BODART F.: IDA, une méthode de conception assistée des systèmes d'information.
Génie Logiciel, Agence de l'Informatique, n° 4, avril 1986.

[BOD88] BODART F., PIGNEUR Y.: *Conception assistée des applications informatiques 1- étude d'opportunité et analyse conceptuelle.*
Masson, Paris, 1988.

[BOE81] BOEHM B. W.: *Software engineering economics.*
Prentice-Hall, Englewood Clifs, 1981.

[BOU83] BOUZEGHOUB M., GARDARIN G.: An expert system for database design.
Int. Workshop on New Applications of Databases, Cambridge, UK, Gardarin & Gelenbe Ed., Septembre 1983.

[BOU86a] BOUZEGHOUB M.: *SECSI: un système expert en conception de systèmes d'information - modélisation conceptuelle de schémas de bases de données.*
Thèse de Doctorat, Université Paris VI, mars 1986.

[BOU86b] BOURGUIGNON J-P.: La structure d'accueil Emeraude.
Génie Logiciel, Agence de l'Informatique, n° 6, novembre 1986.

[BRA76] BRACCHI G., PAOLINI P., PELAGATTI G.: Binary logical association in data modelling.
Modelling in Database Management Systems , IFIP/TC2 Conf., Freudenstadt, G.M. Nijssen Ed., North Holland, 1976.

[BRI85] BRIAND H., HABRIAS H., HUE J-F., SIMON Y.: Expert system for translating an E_R diagram into databases.
4ème International Conference on Entity-Relationship Approach, 1985.

[BUD84] BUDDE R., KUHLENKAMP K., MATHIASSEN L., ZULLIGHOVEN H.: *Approaches to prototyping.*
Springer-Verlag, Berlin, 1984.

[CAU88] CAUVET C., PROIX C., ROLLAND C.: Information systems design: an expert system approach.
International Conference on Extending Database Technologie, Venice, mars 1988, *Lecture Notes in Computer Science*, Springer-Verlag, n° 303, 1988.

[CEC87] CECIMA: Message spécification, maquette, projet.
Journées Pratique des Méthodes et Outils Logiciels d'aide à la Conception de Systèmes d'Information, Nantes, 23-24 septembre 1987.

[CGI87] CGI-Informatique: La station de travail PACBASE.
Journées Pratique des Méthodes et Outils Logiciels d'aide à la Conception de Systèmes d'Information, Nantes, 23-24 septembre 1987.

[CHE76] CHEN P.P.: The entity relationship model - toward a unified view of data.
ACM Transactions on Database Systems, mars 1976.

[CHE80] CHEN P.P. (ed.): *Conf. Entity-Relationship Approach to Systems Analysis and Design.*
North Holland, Los Angeles, 1980.

[COU88] COUTAZ J.: De l'ergonomie à l'informaticien: pour une méthode de conception et réalisation des interfaces homme-machine.
ERGO-IA'88, Ergonomie et Intelligence Artificielle , Biarritz, 4-6 octobre 1988.

[DAV77] DAVIS R., BUCHANAN B.: Meta level knowledge and applications.
IJCAI, 1977.

[DEL82] DELOBEL C., ADIBA M.: *Bases de données et systèmes relationnels.*
Dunod Informatique, Paris, 1982.

[DEM78] DE MARCO T.: *Structured Analysis and System Specification.*
Yourdon Inc, New York, 1978.

[FAR85] FARRENY H.: *Les systèmes experts - principes et exemples.*
Editions Cepadues, Toulouse, 1985.

[FIS85] FISCHER D., LANGLEY P.: Approaches to conceptual clustering.
Proceedings of the 9th International Joint Conference of Artificial Intelligence, 1985.

[FRA86] FRASSON C., LUSTMAN F., PIZEM S., TSIANG D.: Système expert pour la conduite de projets informatiques.
Information & Communication 86, AFCET, Paris, 3-5 juin 1986.

[GAL84] GALACSI: *Les systèmes d'information - analyse et conception.*
Dunod Informatique, Paris, 1984.

[GAL86a] GALLO F.: *The PCTE initiative: toward an european approach to software engineering.*
Ada, Software Factories in Europe Conference, Capri, avril 1986.

[GAL86b] GALLO F., MINOT R., THOMAS I.: The object management system of PCTE as a software engineering database management system.
ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Henderson P. (ed), Palo Alto, 9-11 decembre 1986, *SIGPLAN Notices,* 22(1), janvier 1987.

[GAL87] GALINIER M.: STP, Software Through Pictures: multiples points de vue dans la conception d'un système d'information.
Journées Pratique des Méthodes et Outils Logiciels d'aide à la Conception de systèmes d'Information, Nantes, 23-24 septembre 1987.

[GAT86] GATELLIER M-H, BERTINCHAMPS J-P.: *Contribution au développement d'un système de pilotage pour la conception d'un schéma conceptuel d'informations.*
Mémoire de maîtrise en Informatique de l'Université de Namur, Namur, 1986

[GEN86] Informatique de gestion: les méthodes.
Génie Logiciel, Agence de l'Informatique, n° 4, avril 1986.

[GIA85] GIAMBIASI N., RAULT J-C., SABONNADIERE J-C.: *Introduction à la conception assistée par ordinateur.*
Hermes Publishing, Paris, 1985.

[GID84] GIDDINGS R.V.: Accommodating uncertainty in software design.
Communications of the ACM, 27(5), mai 1984.

[GIG86] GIGCH J. P., PIPINO L. L.: In search of a paradigm for the discipline of information systems.
Future Computing Systems , Oxford University Press, 1 (1), 1986.

[GIR85] GIRAUDIN J-P, DELOBEL C., DARDAILLER P.: Eléments de construction d'un système expert pour la modélisation progressive d'une base de données.
Journées Base de Données Avancées, AFCET, Saint-Pierre de Chartreuse, 6-8 mars 1985.

[HAR86] HARANDI M.T., SCHANG T., COHEN S.: Rule base management using meta-knowledge.
SIGMOD'86, International Conference on Management of Data, 15(2), juin 1986.

[IDA86] *IDA software support system for information system development. Reference Manual: DSL-SPEC. Release 2.0.*
Facultés Universitaires N.D. de la Paix, Namur, 1986.

[IND85] INDEX TECHNOLOGY CORP.: *Excelerator.*
1985.

[INF73] Informatique et Gestion: Les méthodes d'analyse.
Informatique et Gestion, nro. 45, mars 1973.

[LAN86] LAND F., SOMOGYI E.: Software engineering: the relationship between a formal system and its environment.
Journal of Information Technology, Kogan Page Ltd, Londres, février 1986.

[LEH81] LEHMAN M.: The environment of program development and maintenance.
Software Development Environments, Wasserman A.J. (ed), Computer Society Press, New York, 1981.

[LEM77] LEMOIGNE J. L.: *La théorie du système général.*
Presses Universitaires de France, Paris, 1977.

[LIE79] LIEN Y.E.: Multivalued dependencies with null values in relational databases.
VLDB , Rio de Janeiro, 1979.

[LIN88] LINGAT J-Y.: *RUBIS: un système pour la spécification et le prototypage d'applications bases de données.*
Thèse de Doctorat, Université Paris 6, Paris, 1988.

[LIS86] LISSANDRE M.: La méthode SADT.
Génie Logiciel, Agence de l'Informatique, n° 4, avril 1986.

[MAR86] MARTIN J.: Dossier: Développer des systèmes informatiques "pertinents".
Le Monde Informatique , 19 mai 1986.

[MAR87] MARCA D.A., MC GOWAN C.L.: *SADT - Structured Analysis and Design Technique.*
McGraw Hill Inc., New York, 1987.

[MAR88] MARY F., TRICAS F.: *Dossier de synthèse sur le projet de traduction des formules Z en clauses Prolog.*
Rapport DESS-IDC, Université Grenoble II, Grenoble, 1988.

[MEL79] MELESE J.: *Approches systémiques des organisations.* Editions Hommes et Techniques, Suresnes, 1979.

[MEY78] MEYER B., BAUDOIN C.: *Méthodes de programmation.* Editions Eyrolles, Paris, 1978.

[MIC83] MICHALSKI R.S.: A theory and a methodology of inductive learning. *Machine Learning*, Michalski, Carbonell et Mitchell (eds), Tioga, 1983.

[MIT82] MITCHELL T.M.: Generalisation as search. *Artificial Intelligence*, n° 18, 1982

[MOR86] MOREJON J., SANGUIGNOL R., FLORY A.: Un système d'aide à la conception associé au SGBD Scrabble. *Congrès INFORSID 86,* Fontevraud, 27-30 mai 1986.

[MUL79] MULLERY G.P.: CORE - A method for controled requirements specification. *4th International Conference on Software Engineering* , IEEE, Munich, Septembre 1979.

[NIX87] NIXON B., CHUNG K.L., LAUZON D., BORGIDA A., MYLOPOULOS J, STANLEY M.: *Technical note CSRI-44.* Departement of Computer Sciences, University of Toronto, 1987.

[OLL82] OLLE T.W., SOL H.G., VERRIJN-STUART A.A. (ed.): *Information Systems Design Methodologies: a comparative review.* North Holland, Amsterdam, 1982.

[PEL86] PELLAUMAIL P.: *La méthode AXIAL: conception d'un système d'information.* Les Editions d'Organisation, Paris, 1986.

[PIP87] PIPARD E.: *INDE: un système de détection d'inconsistances et d'incomplétudes dans les bases de connaissances.* Thèse de 3ème cycle, Université de Paris-Sud, Paris, 1987.

[PRO86] PROIX C.: Système expert pour la conception de systèmes d'information. *Information & Communication,* AFCET, Paris, 3-5 juin 1986.

[RIG86] RIGAL J.: SPECIF: outil d'aide à la spécification.
Génie Logiciel, Agence de l'Informatique, n° 5, mai 1986.

[ROL86] ROLLAND C.: Introduction à la conception des systèmes d'information et panorama des méthodes disponibles.
Génie Logiciel, Agence de l'Informatique, n° 4, avril 1986.

[ROL88] ROLLAND C., FOUCAUT O., BENCI G.: *Conception des Systèmes d'information: la méthode REMORA.*
Editions Eyrolles, Paris, 1988.

[ROS77] ROSS D.T., SCHOMAN JR K.E.: Structured analysis for requirements definition.
IEEE Transactions on Software Engineering, 3(1), janvier 1977.

[ROU87a] ROUSSEL A.: MEGA Progiciels.
Journées Pratique des Méthodes et Outils Logiciels d'aide à la Conception de systèmes d'Information, Nantes, 23-24 septembre 1987.

[ROU87b] ROUSSET M.C.: Sur la validité des bases de connaissances: le système COVADIS.
Les Systèmes Experts et leurs Applications, Avignon, mai1987.

[SAC68] SACKMAN H., ERIKSON W.J., GRANT E.E: Exploratory experimental studies comparing online and offline programmer performance.
Communications of the ACM, 11(1), janvier 1968.

[STE85] STEPHENS M., WHITEHEAD K.: The analyst: a workstation for analysis and design.
IEEE 8th International Conference on Software Engineering,1985.

[TAB86] TABOURIER Y.: *De l'autre côté de MERISE.*
Les Editions d'Organisation, Paris, 1986.

[TAR83] TARDIEU H., ROCHFELD A., COLETTI R.: *La méthode MERISE - principes et outils.*
Les Editions d'Organisation, Paris, 1983.

[TAR85] TARDIEU H. et alli: *La méthode MERISE - démarche et pratiques.*
Les Editions d'Organisation, Paris, 1985.

[TEI77] TEICHROEW D., HERSEY III E.A.: PSL/PSA: a computer-aided technique for structured documentation and analysis of information processing systems.
IEEE Transactions on Software Engineering, 3(1), janvier 1977.

[VER82] VERMEIR D., NIJSSEN G.M.: A procedure to define the object type structure of a conceptual schema.
Information Systems, vol. 7 n° 4, 1982.

[VES87] VESPA R., SECHER D.: Conceptor.
Journées Pratique des Méthodes et Outils Logiciels d'aide à la Conception de systèmes d'Information, Nantes, 23-24 septembre 1987.

[WAS82] WASSERMAN A.I.: Automated tools in the information system development environment.
Automated tools for information systems design, Schneider H.J. et Wasserman A.I. (eds), North Holland, Amsterdam, 1982.

[WAS86] WASSERMAN A.I.: A graphical extensible integrated environment for software development.
ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Henderson P. (ed), Palo Alto, 9-11 decembre 1986, *SIGPLAN Notices,* 22(1), janvier 1987.

[WEN87] WENGER E.: *Artificial intelligence and tutoring systems - computational and cognitive approach to the communication of the knowledge.*
Morgan Kaufman Publishers, Los Altos, 1987.

[WIL65] WILSON I.G., WILSON M.E.: *Information, computers and system design.*
John Wiley & Sons Inc., New York, 1965.

[ZAN84] Zaniolo C.: Data Base relations with null values.
Journal of Computer and System Sciences, n° 28, 1984.

ANNEXE A

LES MODELES IDA

LES MODELES IDA

A.1 Introduction

Par IDA (Interactive Design Approach), on désigne une méthodologie de conception des systèmes d'information et un logiciel intégré d'aide à la conception des SI [BOD86, BOD88]. Le projet IDA est développé à l'Institut d'Informatique des Facultés Universitaires de Namur, Belgique, en coopération avec le projet ISDOS-PRISE [TEI77] de l'Université de Michigan, EUA.

La méthode IDA prend principalement en considération les SI qui servent de support au fonctionnement opérationnel de l'organisation et aux décisions structurées qui s'y rapportent. Cette méthode est particulièrement bien adaptée pour réaliser l'automatisation de tâches administratives et de gestion routinière.

La démarche méthodologique proposée [BOD88] s'appuie sur un environnement composé de **modèles** , d'**outils automatisés** et de **règles** pour construire des solutions du SI à mettre en place.

Dans cette annexe nous nous limitons à présenter chaque **modèle IDA** à l'aide de schémas Z et d'exemples.

Chaque modèle IDA se rapporte à un aspect particulier du modèle général représenté à la figure A.1.

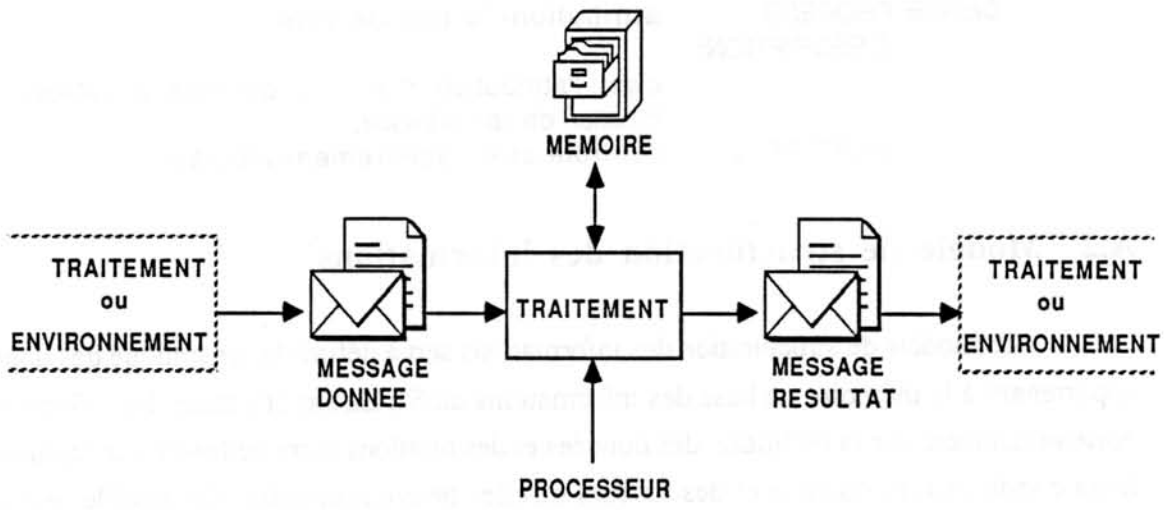


Figure A.1: Schéma du modèle général des SI

Ce schéma classique se lit ainsi: un message-donnée, en provenance d'un traitement ou de l'environnement du SI, est communiqué au SI qui le traite via un processeur, éventuellement à l'aide de sa mémoire, pour engendrer un message-résultat lequel est à son

tour transmis à un autre traitement ou à l'environnement du SI.

Le projet IDA utilise le langage DSL (Dynamic Specification Language) pour décrire formellement les images du SI selon les modèles retenus. Le langage DSL appartient à la famille des langages PSL (Problem Statement Language) défini par le projet ISDOS et présente les caractéristiques générales suivantes:

- c'est un langage de description des spécifications d'un SI;
- il est non procédural au sens où il permet la spécification du SI dans un ordre quelconque et de manière progressive;
- il est basé sur le modèle entité-association et il est construit à partir des notions: (i) de types d'**objet** dont le nombre est fixé dans le langage; (ii) de types de **relation** entre ces objets, dont le nombre est également limité; (iii) de **propriétés** associées à ces types d'objet ou de relation.

Une spécification DSL est formée de sections. Chaque section correspond à la définition d'un objet DSL. Une section commence par l'instruction DEFINE et se termine par une autre instruction DEFINE ou par la fin de la spécification.

Exemple:

```
DEFINE MESSAGE        carte-d-électeur;
       SYNONYMS ARE   carte-électeur,
                      c-e;
       CONSISTS OF    nom-électeur,
                      prénom-électeur,
                      circonscription-électeur,
                      date-émission;

DEFINE PROCESS        attribution-du-lieu-de-vote;
       DESCRIPTION;
                      c'est l'attribution d'un lieu de vote à l'électeur en
                      fonction de son adresse;
       PART OF        contrôle-et-enregistrement-électeur;
```

A.2 Modèle de structuration des informations

Ce modèle de structuration des informations sert à définir la sémantique des données appartenant à la mémoire ou base des informations du SI. La structuration des informations porte notamment sur la définition des données et des relations entre celles-ci, sur l'analyse de leurs conditions d'existence et des valeurs qu'elles peuvent prendre. Ce modèle est aussi utilisé pour définir les messages qui véhiculent les informations.

Le concept de mémoire ou base des informations couvre tous les types d'informations stockées dans le SI, aussi bien sur des supports magnétiques que sur des supports manuels.

Les informations sont structurées selon le modèle entité-association [CHE76, CHE80] qui permet de modéliser les informations du réel perçu à partir de 3 concepts élémentaires:

- **entité**- une entité est une chose concrète ou abstraite appartenant au réel perçu à propos de laquelle on veut enregistrer des informations, comme par exemple: ELECTEUR, LIEU DE VOTE, JUGE ELECTORAL.

- **association**- une association est définie par une correspondance entre deux ou plusieurs entités (non nécessairement distinctes) où chacune assume un rôle donné. Exemple: l'association VOTE entre les entités ELECTEUR et LIEU DE VOTE.

- **attribut**- caractéristique ou qualité d'une entité ou d'une association, comme par exemple: NOM DE L'électeur et ADRESSE DE L'électeur.

Sur ces éléments peuvent agir des *contraintes d'intégrité* telles que des contraintes d'existence, de connectivité, de format, de valeur, de dépendance fonctionnelle.

La figure A.2 présente un exemple de schéma du modèle de structuration des informations.

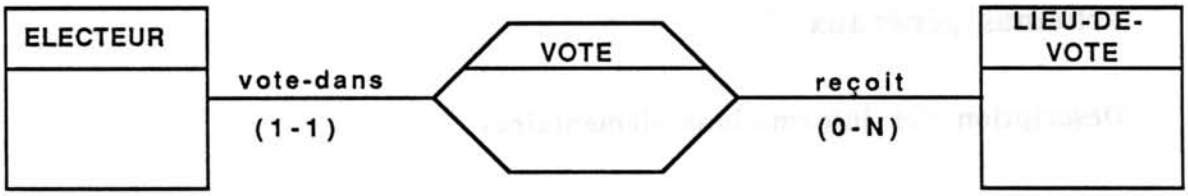


Figure A.2: Exemple de schéma d'un modèle de structuration des informations.

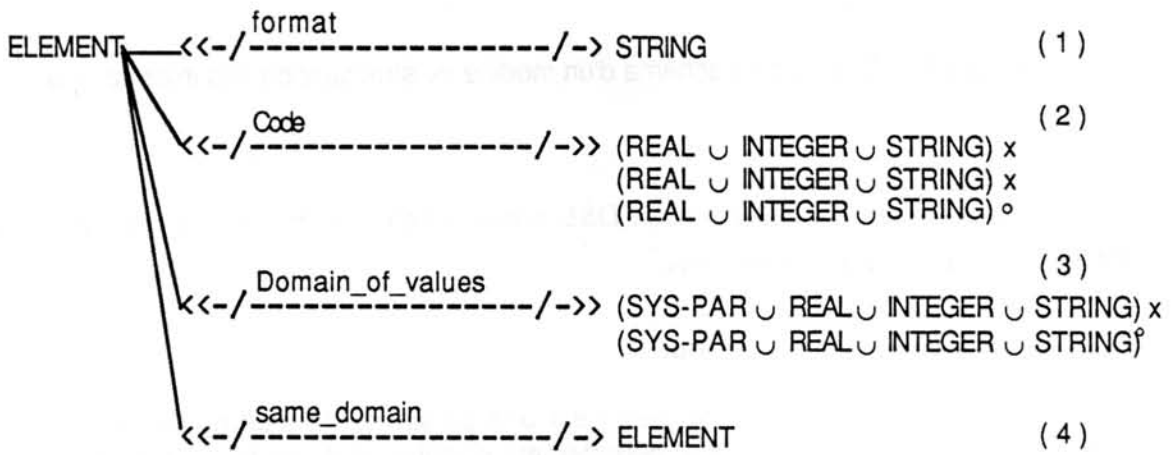
Un exemple de spécification DSL correspondant au modèle de structuration des informations est présenté ci-dessous:

```
DEFINE   ENTITY          électeur;
         DESCRIPTION;
                         un "électeur" est une personne de plus de 18 ans, qui habite
                         dans le département et qui a voulu se faire enregistrer;
         CONSISTS OF     nro-électeur,
                         ident-électeur,
                         adresse-électeur;
         IDENTIFIED BY   nro-électeur;
         IDENTIFIED BY   ident-électeur;

DEFINE   GROUP           ident-électeur;
         CONSISTS OF     nom-électeur,
                         prénom-électeur,
                         date-naissance-électeur;
```

```
DEFINE   ENTITY          lieu-de-vote;
         DESCRIPTION;
                         un "lieu de vote" est un lieu, déterminé par le service de
                         Justice Electorale, où il y a des urnes lors des élections;
         CONSISTS OF     nro-LV,
                         adresse-LV,
                         capacité-totale-LV,
                         nbr-votants-LV;

DEFINE   RELATION        vote;
         DESCRIPTION;
                         "vote" désigne l'attribution d'un lieu de vote à un électeur;
         RELATES         électeur AS vote-dans WITH CONNECTIVITY "1-1";
         RELATES         lieu-de-vote AS reçoit WITH CONNECTIVITY "0-N";

DEFINE   ELEMENT         date-naissance-électeur;
         FORMAT          "9(8)";
         VALUE ARE       01011850   THRU   31121980;
```

Schémas généraux Z

Description des informations élémentaires



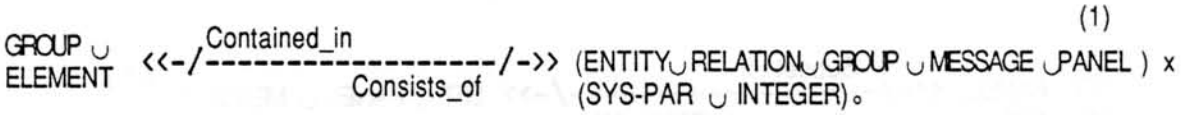
(1) format d'un élément

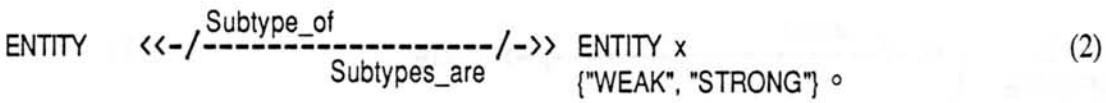
(2) caractérisation de la codification associée à un élément

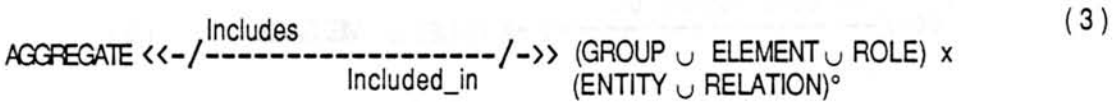
(3) attribution de valeurs ou d'intervalles de valeurs à un élément

(4) spécification d'un élément avec la même composition qu'un autre élément

Description des groupes d'information





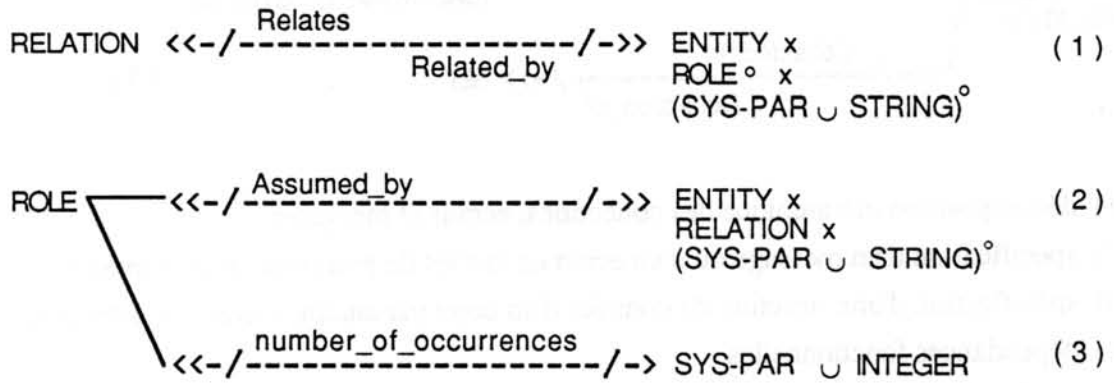


(1) composantes d'une structure de données et éventuellement nombre d'occurrences

(2) spécialisation d'entités

(3) composition d'un objet

Description des relations

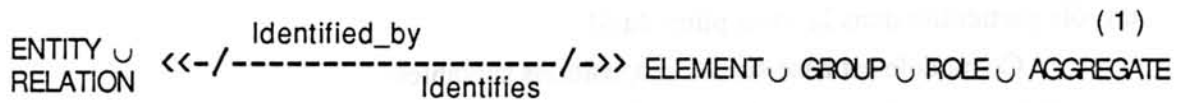


(1) liaisons d'une relation avec des entités et éventuellement les rôles et les connectivités de ces liaisons

(2) rôle assumé par une entité dans une relation et éventuellement sa connectivité

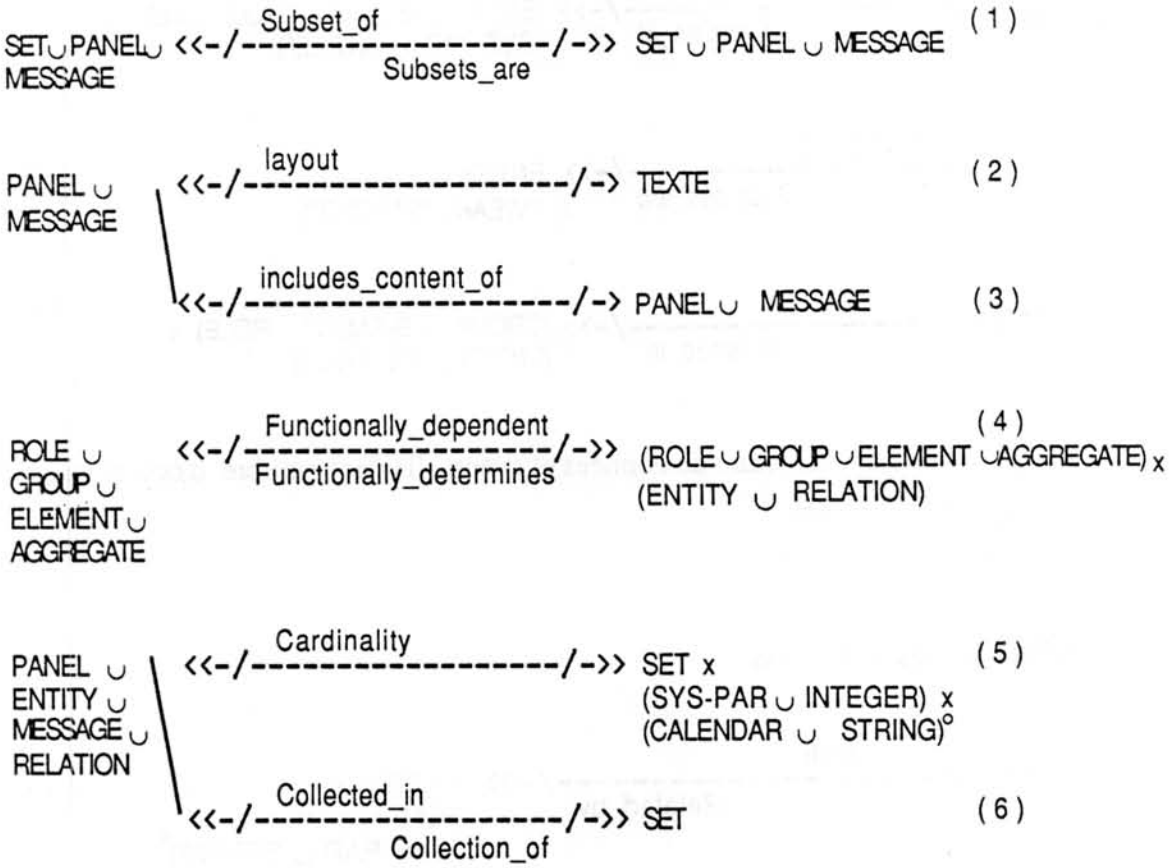
(3) nombre d'occurrences d'un rôle dans une relation

Identification



(1) définition d'un identifiant attaché à une entité ou à une relation

Descriptions complémentaires



(1) décomposition hiérarchique des collections, écrans et messages

(2) spécification d'un message ou d'un écran en termes de présentation du contenu

(3) spécification d'une structure de données d'un objet par analogie avec un autre objet

(4) dépendances fonctionnelles

(5) cardinalités d'un objet dans des collections et éventuellement périodes de validité

(6) objets contenus dans une collection

A.3 Modèle de structuration des traitements

Le modèle de structuration des traitements fournit aux concepteurs et aux analystes des critères leur permettant de décomposer un projet en traitements de plus en plus élémentaires. Plus précisément, ce modèle doit procurer des critères d'identification de ces traitements en fonction d'une nomenclature standardisée, chaque niveau de celle-ci assumant un rôle particulier dans la conception du SI.

Ce modèle est basé sur les propositions suivantes:

- tout traitement est décomposable sous forme arborescente: un traitement de niveau intermédiaire i (i >1) provient de la décomposition d'un seul traitement de niveau i-1 et se décompose en n ($n \geq 1$) traitements de niveau i+1;

- la décomposition en traitements de plus en plus élémentaires correspond à une démarche par raffinements successifs (conception descendante);
- à priori, tout traitement est décomposable en un nombre quelconque de niveaux. Cependant, on distinguera des niveaux ou repères privilégiés compte tenu de leur signification particulière dans la méthode. Ces repères, qui constituent une *nomenclature standard*, sont: (i) **projet**- partie du SI qui fait l'objet d'une analyse; (ii) **application**- traitement *quasi-autonome* par rapport aux autres applications d'un projet et représentant la dimension minimale d'un projet informatique; (iii) **phase**- traitement (manuel ou automatique) possédant une *unité spatio-temporelle d'exécution* ; (iv) **fonction**- correspondant au niveau élémentaire de la nomenclature des traitements; elle est associée à un *objectif* et à un *comportement* organisationnel considérés comme élémentaires par l'organisation. La figure A.3 montre les niveaux des traitements.

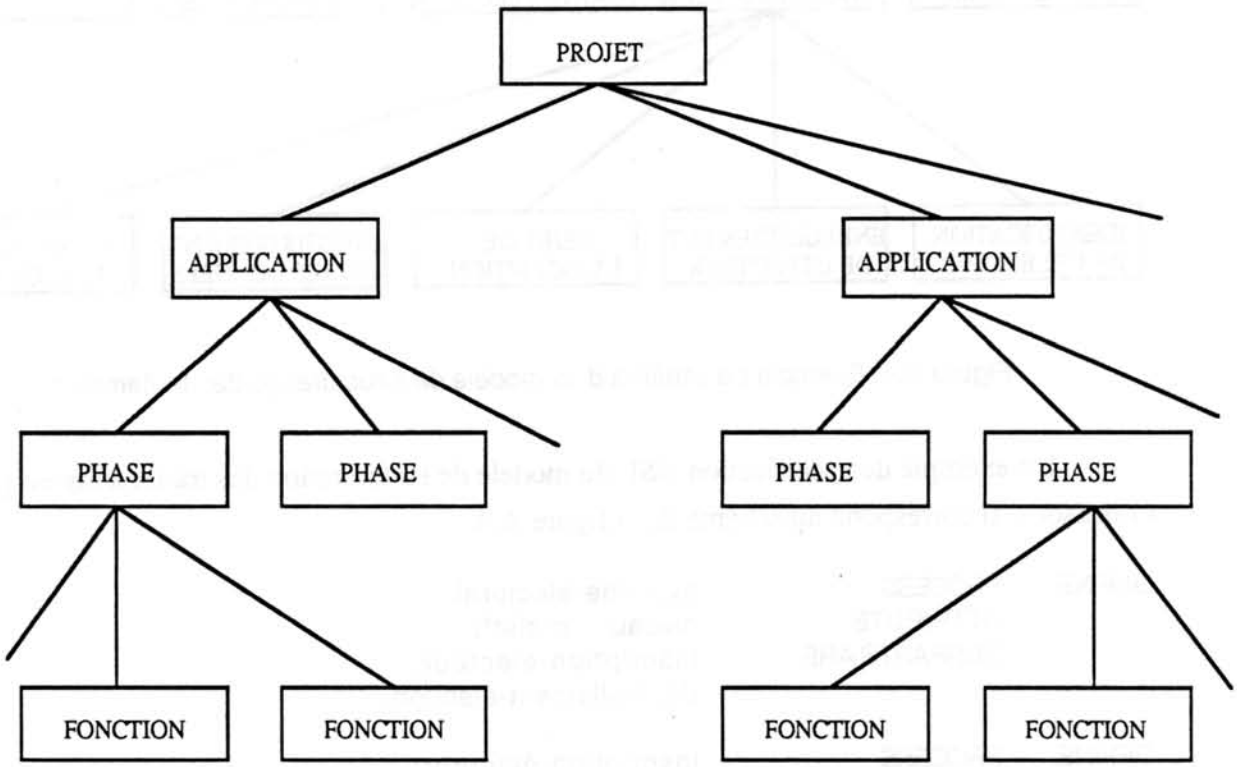


Figure A.3: Schéma de la nomenclature standard des niveaux des traitements.

La figure A.4 présente comme exemple de modèle de structuration des traitements une partie du schéma d'un système électoral simple où nous identifions deux traitements de niveau application: Inscription de l'Electeur et Dépouillement d'une Election. Le traitement Inscription de l'Electeur est décomposé en cinq traitements de niveau phase, respectivement: Vérification du Dossier, Contrôle et Enregistrement de l'Electeur, Préparation de la Carte de l'Electeur, Signature du Responsable et Expédition. Le traitement Contrôle et Enregistrement de l'Electeur, pour sa part, est décomposé en cinq traitements de niveau fonction: Identification de l'Electeur, Enregistrement de l'Electeur, Rejet de l'Inscription, s'il est déjà inscrit, Enregistrement d'Autres

Données, comme adresse de l'électeur et nom des parents et Attribution du Lieu de Vote.

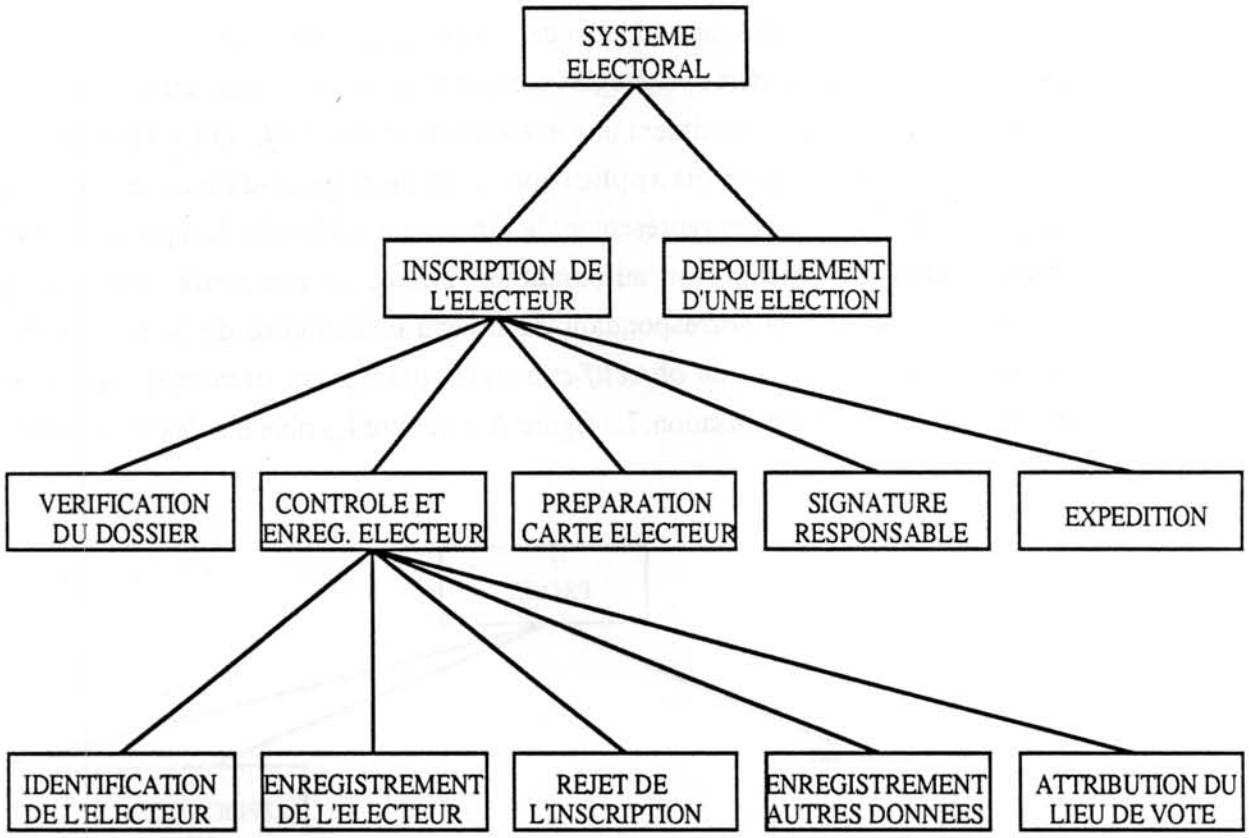


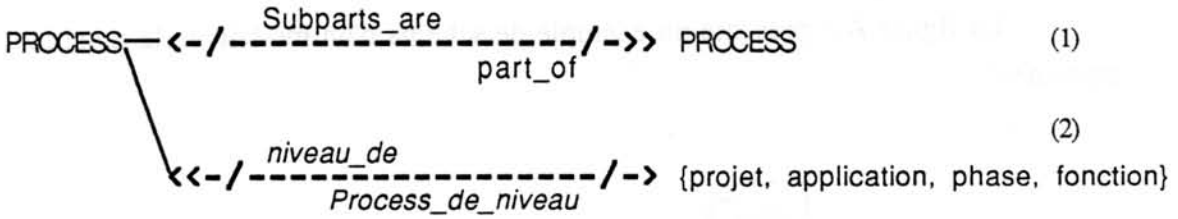
Figure A.4: Exemple de schéma d'un modèle de structuration des traitements.

Un exemple de spécification DSL du modèle de structuration des traitements est présenté ci-dessous. Il correspond au schéma de la figure A.4.

```
DEFINE    PROCESS          système-électoral;
          ATTRIBUTE        niveau "projet";
          SUBPARTS ARE     inscription-électeur,
                           dépouillement-élection;

DEFINE    PROCESS          inscription-électeur;
          ATTRIBUTE        niveau "application";
          SUBPARTS ARE     vérification-dossier,
                           contrôle-enregistrement-électeur,
                           préparation-carte-électeur,
                           signature-responsable,
                           expédition;

DEFINE    PROCESS          contrôle-enregistrement-électeur;
          DESCRIPTION;
                           si l'électeur n'est pas encore enregistrée,
                           l'enregistrer et lui fournir la carte d'électeur;
          SYNONYM IS       contr-enreg-électeur;
          ATTRIBUTE        niveau "phase";
```

SUBPARTS ARE identification-électeur,
enregistrement-électeur,
rejet-inscription,
enregistrement-autres-données,
attribution-lieu-vote;

Schéma général Z



(1) décomposition hiérarchique des processus

(2) caractérisation possible des niveaux hiérarchiques de la décomposition

A.4 Modèle de la dynamique des traitements

Ce modèle complète le modèle de structuration des traitements en fournissant à l'analyste des concepts et des mécanismes lui permettant de représenter les conditions de déclenchement, d'exécution et d'enchaînement des traitements en vue de caractériser le comportement du SI.

Le modèle de la dynamique repose sur deux concepts de base: le processus et l'événement.

Un **processus** est l'exécution d'une procédure de traitement de l'information dont la progression peut être représentée, à des points dans le temps, par son état. En se limitant aux aspects les plus souvents utilisés du modèle de la dynamique, sont retenus trois états caractéristiques du cycle de vie d'un processus:

- l'état **déclenché**- le processus est *créé* mais en *attente* d'exécution par manque de ressources;
- l'état **actif**- le processus est en cours d'exécution;
- l'état **terminé**- le processus cesse d'exister après avoir atteint son terme final.

Un **événement** correspond à un changement d'état du SI localisé dans le temps et dans l'espace. Un événement est dit **externe** s'il correspond au franchissement de la frontière du SI (changement d'état 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.

Un événement peut causer deux actions dynamiques: le déclenchement d'un processus et la contribution à un point de synchronisation.

La description de la dynamique d'un SI est réalisée à l'aide de six structures élémentaires d'enchaînement des traitements:

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

La figure A.5 présente un exemple de schéma d'un modèle de la dynamique des traitements.

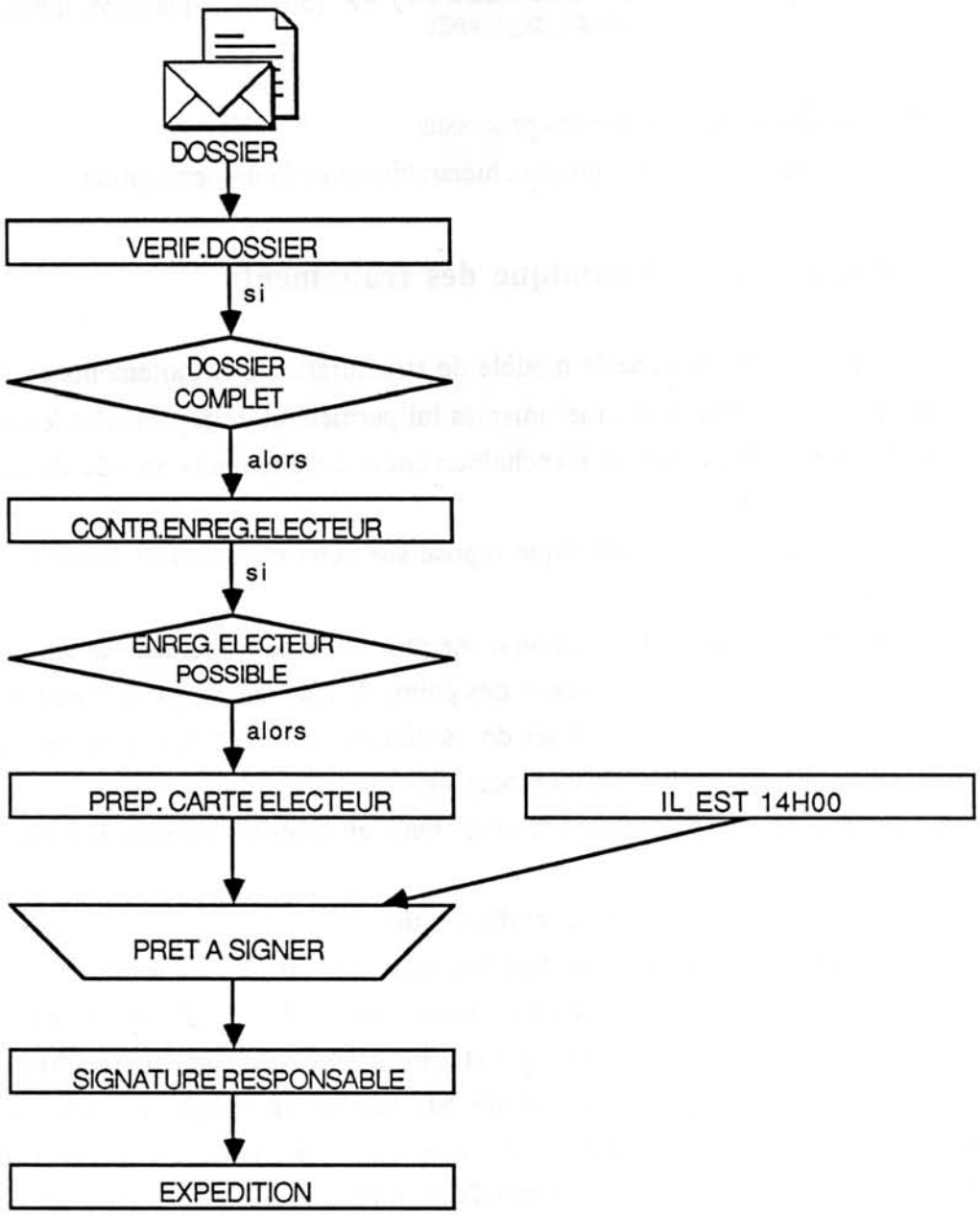


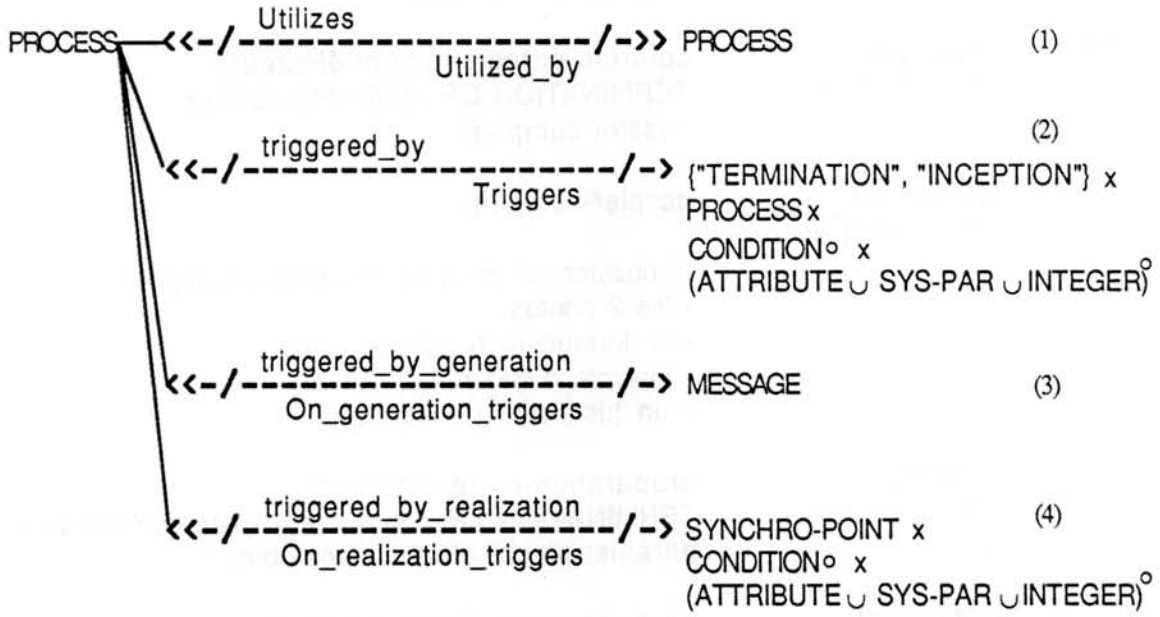
Figure A.5: Exemple de schéma d'un modèle de la dynamique des traitements.

Un exemple de spécification DSL du modèle de la dynamique des traitements, correspondant à la figure A.4, est présenté ci-dessous:

```
DEFINE   PROCESS                vérification-dossier;
         TRIGGERED BY           GENERATION OF dossier;

DEFINE   PROCESS                contrôle-enregistrement-électeur;
         TRIGGERD BY            TERMINATION OF vérification-dossier
         IF                     dossier-complet;

DEFINE   CONDITION              dossier-complet;
         TRUE-WHILE;

                                le dossier est complet, c-à-d qu'il comporte:
                                - les 2 photos
                                - le formulaire rempli et signé
                                - une pièce d'identité
                                - un justificatif de domicile;

DEFINE   PROCESS                préparation-carte-électeur;
         TRIGGERED BY           TERMINATION OF contrôle-enregistrement-électeur
         IF                     enregistrement-électeur-possible;

DEFINE   CONDITION              enregistrement-électeur-possible;
         TRUE-WHILE;

                                - la personne n'est pas encore répertoriée ET
                                - elle a plus de 18 ans ET
                                - elle habite dans le département;

DEFINE   PROCESS                signature-responsable;
         TRIGGERED BY           REALIZATION OF prêt-à-signer;

DEFINE   SYNCHRONIZATION        prêt-à-signer;
         REALIZED-WHEN;

                                il y a des cartes d'électeur à signer ET il est 14H;

DEFINE   PROCESS                expédition;
         TRIGGERED BY           REALISATION OF signature-responsable;
```

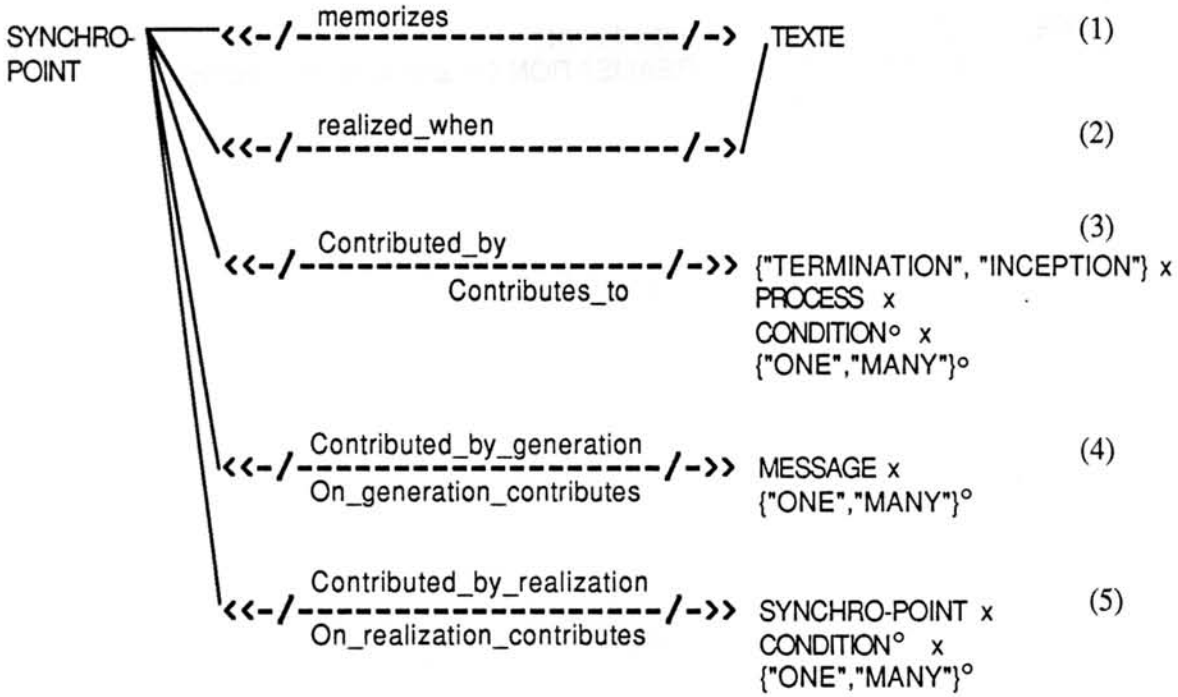
Schéma général Z

Description des processus



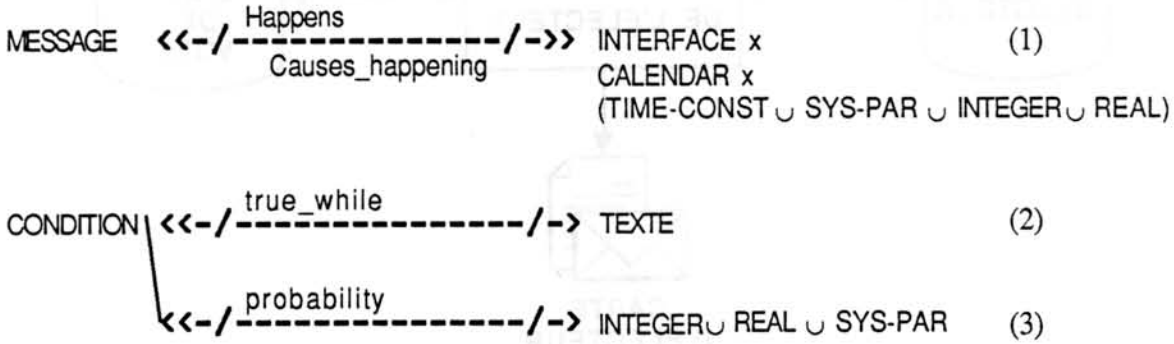
(1) utilisation d'un processus par d'autres processus
(2) déclenchement d'un processus par la terminaison ou l'initialisation d'un autre processus et éventuellement la condition de déclenchement et le nombre d'occurrences du processus déclenché
(3) déclenchement d'un processus par la génération d'un message
(4) déclenchement d'un processus par la réalisation d'un point de synchronisation et éventuellement la condition de déclenchement et le nombre d'occurrences du processus déclenché

Description des points de synchronisation



(1) durée de validité d'un point de synchronisation
(2) définition d'un point de synchronisation
(3) spécification de la contribution à un point de synchronisation par la terminaison ou par l'initialisation d'un processus et éventuellement le mode de contribution (condition et nombre d'occurrences)
(4) contribution d'un message à un point de synchronisation et éventuellement le nombre d'occurrences
(5) spécification de la contribution à un point de synchronisation par la réalisation d'un autre point de synchronisation et éventuellement le mode de contribution (condition et nombre d'occurrences)

Descriptions complémentaires



(1) spécification des messages générés par des interfaces en termes de calendrier et de nombre d'occurrences
(2) définition des conditions
(3) spécification de la probabilité de réalisation de la condition

A.5 Modèle de la statique des traitements

Le modèle de la statique des traitements est une application simple du modèle général d'un SI permettant d'*affiner l'idée de solution* qu'un analyste se crée pour la résolution d'un projet à résoudre, jusqu'à l'obtention d'une *solution* (construction d'un algorithme). Ce modèle a pour but, d'une part, de préciser pour chaque traitement les messages-données et la partie de la mémoire du SI nécessaires à l'obtention des messages-résultats (modèle de la boîte noire) et d'autre part, de spécifier, sous une forme adéquate, la procédure de traitement qui assure la transformation, à l'aide de la base de données du SI, des messages-données en messages-résultats.

Le modèle de la boîte noire permet de spécifier pour un traitement d'un niveau donné:

- les **objectifs** à réaliser;
- les **performances** souhaitées;
- les **informations** en entrée et en sortie;
- les **actions primitives** qui caractérisent les relations entre ces informations et le traitement.

Dans le cas des traitements interactifs, on complète la spécification des règles de traitement par la description du dialogue entre l'utilisateur et le terminal.

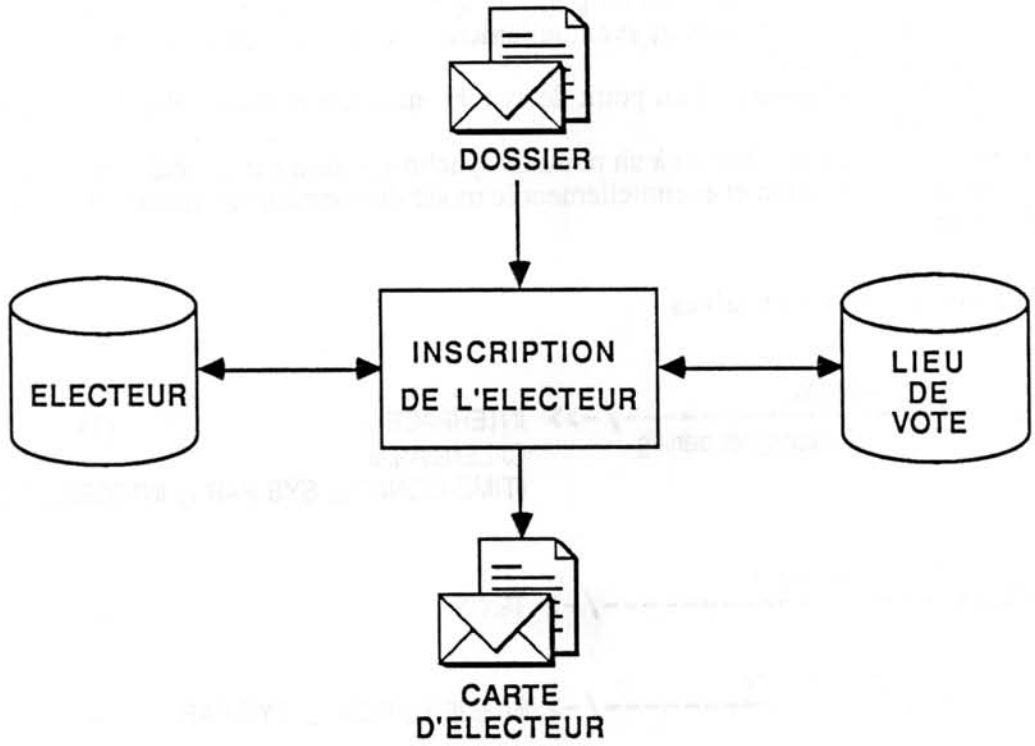


Figure A.6: Exemple de schéma d'un modèle de la statique des traitements.

Un exemple de spécification DSL correspondant au modèle de la statique des traitements de la figure A.6 est présenté ci-dessous:

```
DEFINE   PROCESS      inscription-électeur;
         RECEIVES     dossier;
         GENERATES    carte-électeur;
         USES         électeur,  lieu-de-vote;
```

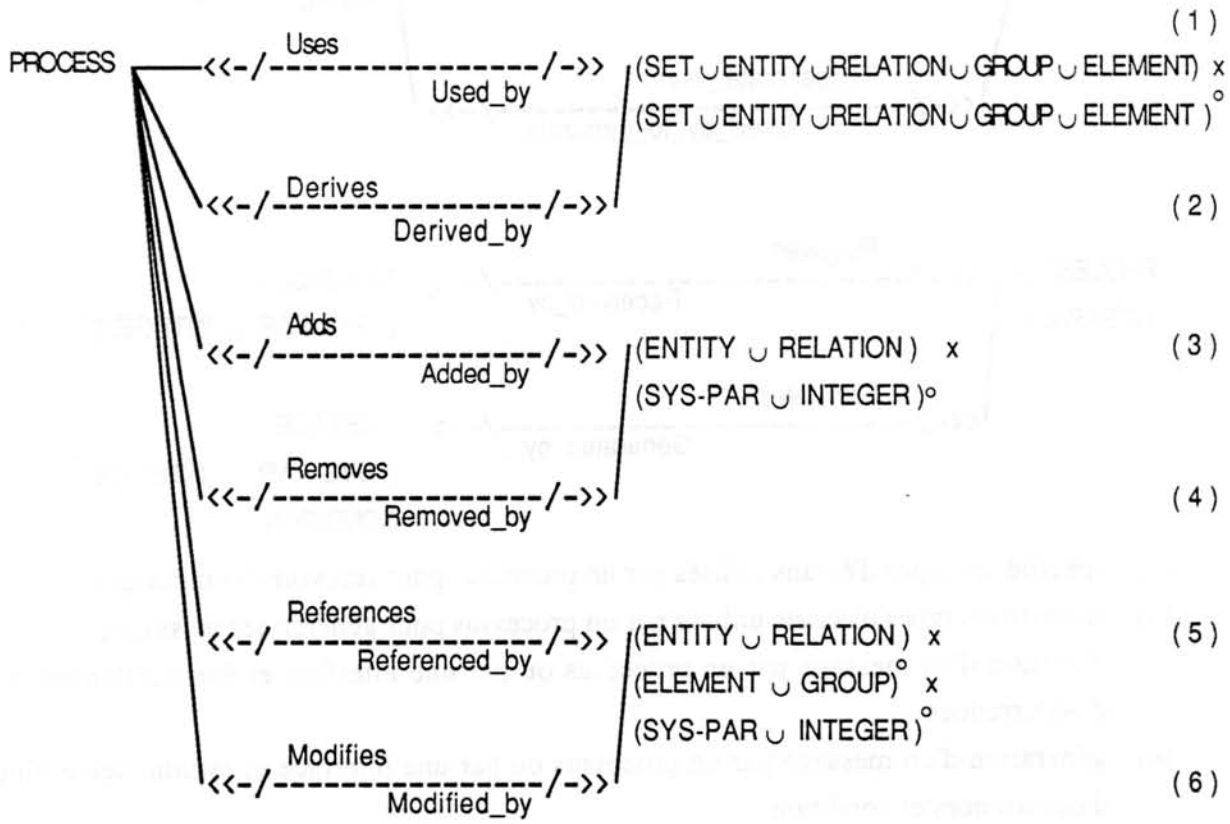
Schéma général

Spécification des traitements élémentaires

```
                    procedure
PROCESS  <<-/-----------------/->  TEXTE  (1)
```

(1) spécification des règles de transformation d'un processus

Utilisation de la mémoire du SI



(1) utilisation de la mémoire du SI par un processus et éventuellement la partie de la mémoire du SI dérivée par le processus

(2) partie de la mémoire du SI dérivée par un processus et éventuellement la partie utilisée par la dérivation

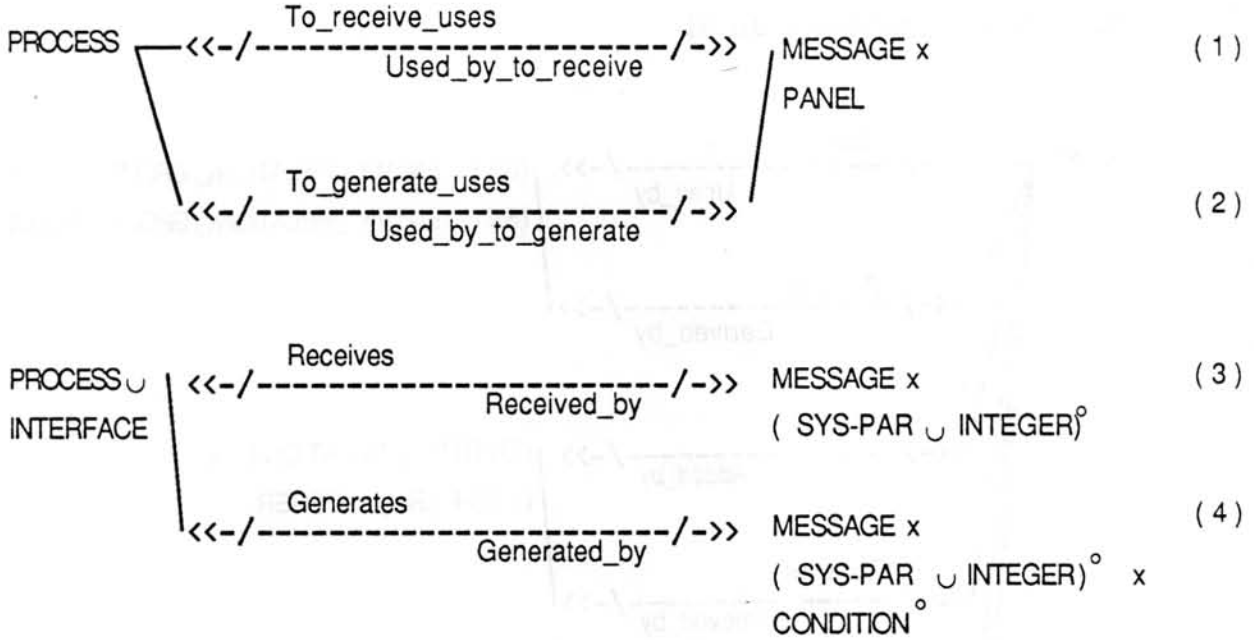
(3) ajout d'occurrences d'une entité ou d'une relation par un processus et éventuellement son nombre

(4) suppression d'occurrences d'une entité ou d'une relation par un processus et éventuellement son nombre

(5) accès en lecture à des occurrences d'une entité ou d'une relation par un processus et éventuellement l'élément ou le groupe référencé et nombre d'occurrences

(6) modification d'occurrences d'une entité ou d'une relation par un processus et éventuellement l'élément ou le groupe modifié et nombre d'occurrences

Description de l'échange de messages



(1) spécifie les types d'écrans utilisés par un processus pour recevoir des messages

(2) spécifie les types d'écrans utilisés par un processus pour générer des messages

(3) réception d'un message par un processus ou par une interface et éventuellement nombre d'occurrences

(4) génération d'un message par un processus ou par une interface et éventuellement nombre d'occurrences et condition

A.6 Modèle des ressources

Le modèle des ressources sert à caractériser le comportement des processeurs qui exécutent les procédures de traitement, afin d'évaluer le caractère réalisable d'une solution conceptuelle. Il importe, en effet, d'estimer si une solution conceptuelle est réalisable compte tenu de la disponibilité des ressources de l'organisation, telles que, par exemple, le personnel, les équipements, les moyens financiers, les horaires de travail ou la composition des équipes et des postes de travail.

Les concepts de base de ce modèle sont les processeurs ou ressources réutilisables et les ressources consommables.

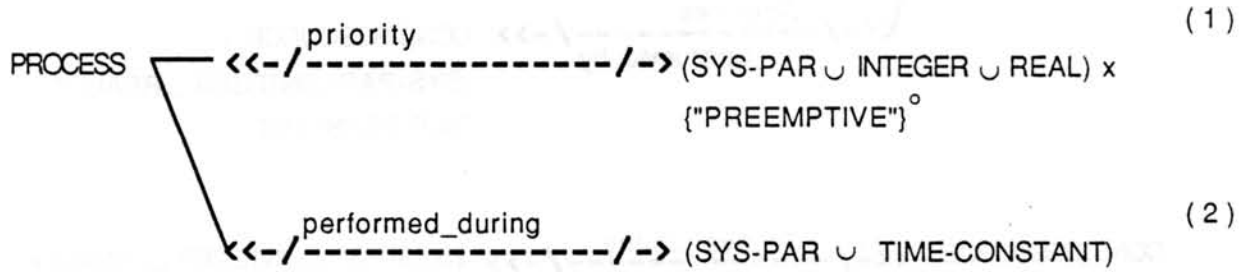
On appelle **processeur** une ressource réutilisable qui peut être requise lors de l'exécution d'un traitement; par réutilisable on entend toute ressource qui, dès qu'est terminé le processus pour lequel elle a été requise, est disponible pour un autre processus. Nous pouvons citer comme exemples une personne, une équipe, un ordinateur, etc...

Une **ressource consommable** est une ressource qui n'est pas réutilisable lorsqu'elle a été consommée lors de l'exécution d'un traitement. Par exemple, le papier d'impression, l'énergie, l'argent sont des ressources consommables.

Tous processeurs et ressources consommables peuvent être caractérisés par les propriétés suivantes: l'unité de mesure, le prix unitaire et la disponibilité.

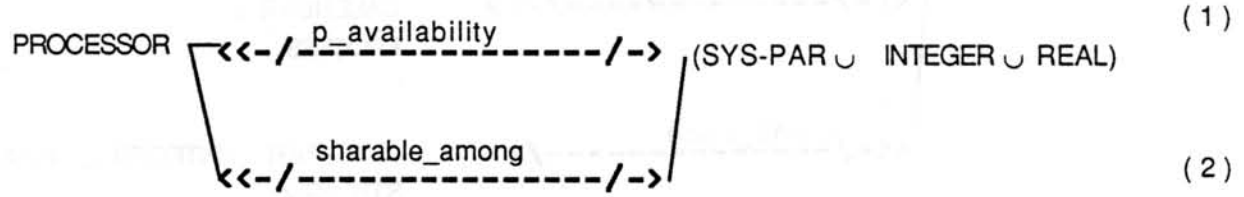
Schéma général Z

Caractéristiques des processus



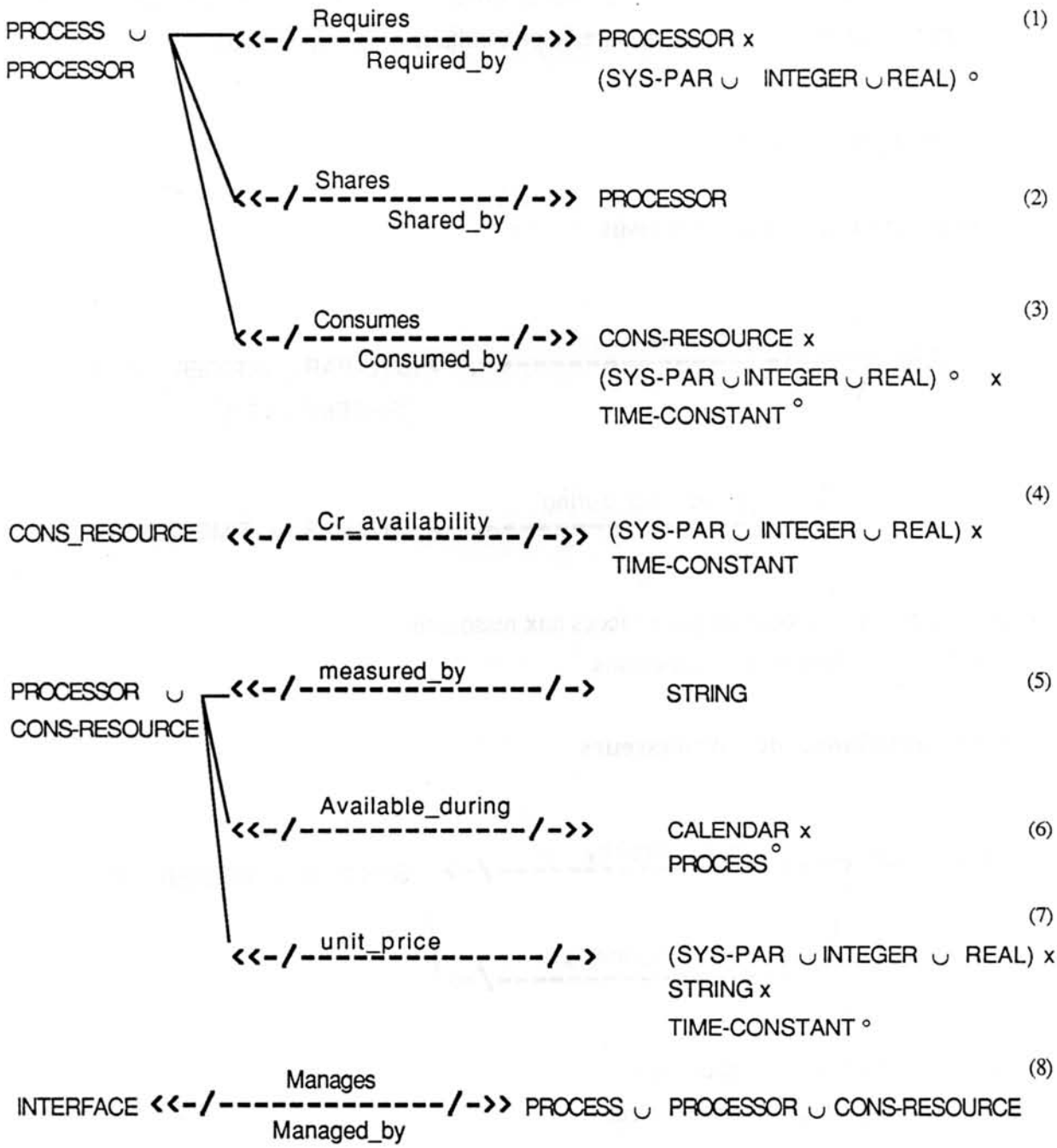
(1) priorité d'un processus pour l'accès aux ressources
(2) durée d'exécution d'un processus

Caractéristiques des processeurs



(1) capacité totale d'un processeur
(2) degré de partage d'un processeur

Utilisation des ressources



(1) réquisition d'un processus ou d'un processeur par un processeur

(2) partage d'un processeur par un processus ou par un processeur

(3) consommation d'une ressource consommable par un processus ou un processeur et éventuellement nombre d'unités par période de temps

(4) disponibilité en nombre et temps des ressources consommables

(5) unité de mesure d'un processeur ou d'une ressource consommable

(6) calendrier de disponibilité d'un processeur ou d'une ressource consommable

(7) prix par unité de mesure et durée

(8) unité organisationnelle qui contrôle un processus, un processeur ou une ressource consommable

---

ANNEXE B

LE PROTOTYPE DEVELOPPE

LE PROTOTYPE DEVELOPPE

Cette annexe B contient quelques parties des principaux modules, représentatives du prototype réalisé en Turbo-Prolog pour montrer la faisabilité de nos propositions.

Ce prototype, aujourd'hui opérationnel, permet de définir (cf. B.3) une nouvelle méthode de modélisation, en choisissant et en organisant des règles issues de la base des règles élémentaires de gestion (cf. B.1). Cette méthode est contrôlée par des méta-règles (cf. B.2) pour constituer une base de connaissances de définition d'une méthode particulière (cf. B.4) et piloter ainsi la saisie et le contrôle de spécifications d'une application (cf. B.5).

Cette annexe néanmoins ne contient que quelques pages des cinq composantes de base de notre prototype; nous les pensons suffisantes pour mettre en évidence les principes qui ont dominé ce développement (cf. §IV.4).

La base des règles élémentaires de gestion reprend les règles énoncées dans la section III.6.

La base de connaissances de définition d'une méthode particulière correspond à la méthode spécifiée dans l'exemple de la section IV.4.5.1.

L'interface de saisie et de contrôle de spécifications prend en compte le sous-ensemble de DSL/IDA défini dans la section III.6 comme langage de spécification des applications.

B.1 - Base des règles élémentaires de gestion

```
/*************************************************************************/
/*  SYSTEME DE PILOTAGE DE LA MODELISATION DES SYSTEMES D'INFORMATION     */
/*                                                                        */
/*  BASE DES REGLES ELEMENTAIRES DE GESTION                               */
/*                                                                        */
/*  auteur: Luis Alvares                                                  */
/*  date:   aout/88                                                       */
/*************************************************************************/

CLAUSES

regle(r1).
regle(r2).
regle(r3).
regle(r4).
regle(r5).
regle(r6).
regle(r7).
regle(r8).
```

• • •

```
/*************************************************************************
                    STRUCTURES DES REGLES DE VERIFICATION
*************************************************************************/

verification_regle(r1) :-
     expression_formelle2(r1,P),
     diagnostic2(r1,P),
     recommandation2(r1,P),
     asserta(erreur_detectee),
     fail.
verification_regle(r1).

verification_regle(r2) :-
     expression_formelle2(r2,P),
     diagnostic2(r2,P),
     recommandation2(r2,P),
     asserta(erreur_detectee),
     fail.
verification_regle(r2).

verification_regle(r3) :-
     expression_formelle2(r3,P),
     diagnostic2(r3,P),
     recommandation(r3),
     asserta(erreur_detectee),
     fail.
verification_regle(r3).
```

• • •

```
/*****************************************************************************
                        EXPRESSIONS INFORMELLES DES REGLES
*****************************************************************************/

expression_informelle(r1,
     "La specification de processus en termes de declenchement par la
      terminaison d'autre processus est interdite.").
expression_informelle(r2,
     "La specification de processus en termes de declenchement par la generation
      d'un message est interdite.").
expression_informelle(r3,
     "Le declenchement d'un processus doit etre unique.").
expression_informelle(r4,
     "La hierarchie des traitements ne doit pas etre definie circulairement.").
expression_informelle(r5,
     "Il doit y avoir au moins un processus defini.").
expression_informelle(r6,
     "Chaque processus doit avoir un declenchement.").
expression_informelle(r7,
     "Chaque processus doit avoir au moins une entree.").
```

• • •

```
/*****************************************************************************
                        EXPRESSIONS FORMELLES DES REGLES
*****************************************************************************/

expression_formelle(r5) :-
     not(process(_)).

expression_formelle(r14) :-
     not(verif_2_messages).

expression_formelle(r18) :-
     not(interface(_)).

expression_formelle2(r1,P) :-
     process(P),
     triggered_by_term(P,_,_).

expression_formelle2(r2,P) :-
     process(P),
     triggered_by_gen(P,_).

expression_formelle2(r3,P) :-
     process(P),
     triggered_by_term(P,_,_),
     triggered_by_gen(P,_).
```

• • •

```
/*****************************************************************************
                        DIAGNOSTICS DES REGLES
*****************************************************************************/


diagnostic(r5) :-
     makewindow(20,23,7,"ERREUR",19,0,6,80),
     write("Absence de specification de processus."),nl.

diagnostic(r14) :-
     makewindow(20,23,7,"ERREUR",19,0,6,80),
     modele(_),
     write("Il y a seulement un message specifie."),nl.
diagnostic(r14) :-
     write("Absence de specification de messages."),nl.

diagnostic2(r1,P) :-
     makewindow(20,23,7,"ERREUR",19,0,6,80),
     write("Le processus '",P,"' a ete specifie en termes de declenchement"),nl,
     write("par la terminaison d'autre processus."),nl.
```

• • •

```
/*****************************************************************************
                        RECOMMANDATIONS DES REGLES
*****************************************************************************/

recommandation(r3) :-
    write("Modifier les specifications pour obtenir un declenchement unique."),
    beep,
    readchar(_),
    removewindow.

recommandation(r4) :-
    write("Modifier les specifications de maniere a eliminer la definition"),nl,
    write("circulaire."),
    beep,
    readchar(_),
    removewindow.

recommandation(r5) :-
    write("Il faut utiliser ao moins une clause DEFINE PROCESS pour
caracteriser"),
    write("les traitements du systeme."),
    beep,
    readchar(_),
    removewindow.
```

• • •

```
/*****************************************************************************
                              TYPES DES REGLES
*****************************************************************************/

type_regle(r1,130).
type_regle(r2,130).
type_regle(r3,154).
type_regle(r4,162).
type_regle(r5,210).
type_regle(r6,220).
type_regle(r7,230).
type_regle(r8,230).
```

• • •

```
/*****************************************************************************
           ENSEMBLES UTILISES DANS CHAQUE REGLE
*****************************************************************************/

ensemble_utilise(r1,process).
ensemble_utilise(r2,process).
ensemble_utilise(r3,process).
ensemble_utilise(r4,process).
ensemble_utilise(r5,process).
ensemble_utilise(r6,process).
ensemble_utilise(r7,process).
ensemble_utilise(r8,process).
ensemble_utilise(r9,process).
ensemble_utilise(r9,message).
```

• • •

```
/*****************************************************************************
                        FONCTIONS UTILISEES PAR CHAQUE REGLE
*****************************************************************************/

fonction_utilisee(r1,triggered_by_termination).
fonction_utilisee(r2,triggered_by_generation).
fonction_utilisee(r3,triggered_by_termination).
fonction_utilisee(r3,triggered_by_generation).
fonction_utilisee(r4,subparts_are).
fonction_utilisee(r6,triggered_by_termination).
fonction_utilisee(r6,triggered_by_generation).
fonction_utilisee(r7,receives).
fonction_utilisee(r8,generates).
fonction_utilisee(r9,receives).
fonction_utilisee(r9,subparts_are).
```

• • •

B.2 - Base des méta-règles de contrôle de méthodes

```
/*************************************************************************/
/*   SYSTEME DE PILOTAGE DE LA MODELISATION DES SYSTEMES D'INFORMATION    */
/*                                                                        */
/*   BASE DES META-REGLES DE CONTROLE DE METHODES                         */
/*                                                                        */
/*   auteur: Luis Alvares                                                 */
/*   date: aout/88                                                        */
/*                                                                        */
/*   Cette base est utilisee par le module de verification de methodes    */
/*************************************************************************/

CLAUSES

meta_regle(mr1).
meta_regle(mr2).
meta_regle(mr3).
meta_regle(mr4).
meta_regle(mr5).
meta_regle(mr6).
meta_regle(mr7).

                              • • •

/*************************************************************************
                STRUCTURES DES META-REGLES DE CONTROLE
*************************************************************************/

verif_meta_regle(mr1) :-
     expression_formelle_mr(mr1),
     diagnostic_mr(mr1),
     recommandation_mr(mr1),
     fail.
verif_meta_regle(mr1).

verif_meta_regle(mr2) :-
     expression_formelle2_mr(mr2,M),
     diagnostic2_mr(mr2,M),
     recommandation_mr(mr2),
     fail.
verif_meta_regle(mr2).

verif_meta_regle(mr3) :-
     expression_formelle2_mr(mr3,P),
     diagnostic2_mr(mr3,P),
     recommandation_mr(mr3),
     fail.
verif_meta_regle(mr3).

verif_meta_regle(mr4) :-
     expression_formelle3_mr(mr4,R,M),
     diagnostic3_mr(mr4,R,M),
     recommandation_mr(mr4),
     fail.
verif_meta_regle(mr4).
```

• • •

```
/*****************************************************************************
                        EXPRESSIONS INFORMELLES DES REGLES
*****************************************************************************/

expression_informelle_mr(mr1,
      "Une methode doit avoir au moins un modele (etape).").
expression_informelle_mr(mr2,
      "Chaque modele doit comprendre au moins un paquet de regles.").
expression_informelle_mr(mr3,
      "Chaquet paquet de regles doit contenir au moins une regle.").
expression_informelle_mr(mr4,
      "Chaque regle d'un modele doit etre contenue dans un paquet de regles.").
expression_informelle_mr(mr5,
      "Chaque paquet de regles d'un modele doit etre declenchable dans ce
       modele.").
```

• • •

```
/*****************************************************************************
                        EXPRESSIONS FORMELLES DES REGLES
*****************************************************************************/

expression_formelle_mr(mr1) :-
     not(modele(_)).

expression_formelle2_mr(mr2,M) :-
     modele(M),
     not(comprend(M,_)).

expression_formelle2_mr(mr3,P) :-
     paquet(P),
     not(contient(P,_)).

expression_formelle2_mr(mr7,P) :-
     paquet(P),
     not(comprend(_,P)).

expression_formelle3_mr(mr4,R,M) :-
     modele(M),
     contient(M,R),
     not(verif_cond(mr4,M,R)).
```

• • •

```
/*****************************************************************************
                        DIAGNOSTICS DES REGLES
*****************************************************************************/

diagnostic_mr(mr1) :-
     write(" Absence de definition de modeles."),nl.

diagnostic2_mr(mr2,M) :-
     write(" Le modele ",M," ne comprend aucun paquet de regles."),nl.

diagnostic2_mr(mr3,P) :-
     write(" Le paquet ",P," ne contient aucune regle."),nl.

diagnostic2_mr(mr7,P) :-
     write(" Le paquet ",P," n'est compris dans aucun modele."),nl.

diagnostic3_mr(mr4,R,M) :-
     write(" La regle ",R," n'est contenue dans aucun paquet du modele
",M,"."),nl.

diagnostic3_mr(mr5,M,P) :-
     write(" Le paquet ",P," n'est pas declenchable dans le modele ",M,"."),nl.
```

• • •

```
/*****************************************************************************
                        RECOMMANDATIONS DES REGLES
*****************************************************************************/

recommandation_mr(mr1) :-
     write(" Une methode doit etre composee par au moins un modele (etape)"),
     nl,nl.

recommandation_mr(mr2) :-
     write(" Il faut definir le(s) paquet(s) du modele."),nl,nl.

recommandation_mr(mr3) :-
     write(" Il faut definir le(s) regle(s) du paquet."),nl,nl.

recommandation_mr(mr4) :-
     write(" Il faut definir le paquet qui contient cette regle."),nl,nl.

recommandation_mr(mr5) :-
     write(" Il faut definir l'evenement qui declenche ce paquet."),nl,nl.

recommandation_mr(mr6) :-
     write(" Il faut verifier si cette regle doit ou non etre "),
     write("declenchee dans ce modele."),nl,nl.
```

• • •

B.3 - Interface de configuration de méthodes

```
code=4000
/*****************************************************************************/
/*   SYSTEME DE PILOTAGE DE LA MODELISATION DE SYSTEMES D'INFORMATION         */
/*                                                                            */
/*   INTERFACE DE CONFIGURATION DE METHODES                                   */
/*       module de dialogue et de configuration de methodes                   */
/*       module de verification                                               */
/*                                                                            */
/*   programme: CONFIG.PRO                                                    */
/*   auteur:    Luis Alvares                                                  */
/*   date:      aout/88                                                       */
/*                                                                            */
/*   entree: - une base des regles elementaires de gestion                    */
/*           - une base des meta-regles de controle.                          */
/*   sortie: - une base de definition d'une methode particuliere              */
/*****************************************************************************/

/*****************************************************************************/
/*             DECLARATIONS NECESSAIRES AUX OUTILS DU TOOLBOX                */
/*****************************************************************************/

include "decltool.pro"

DATABASE
/*****************************************************************************
             CLAUSES DE DEFINITION DE LA METHODE A UTILISER
*****************************************************************************/

comprend(string,string)                  /* comprend(modele,paquet)            */
contient(string,string)                  /* contient(modele_ou_paquet,regle)   */
declenche(string,string,string)          /* declenche(evenement,modele,paquet) */
evenement(string)                        /* evenement(code_evenement)          */
methode(string)                          /* methode(code_methode)              */
modele(string)                           /* modele(code_modele)                */
modele_initial(string)                   /* modele_initial(code_modele)        */
paquet(string)                           /* paquet(code_paquet)                */
regle(string)                            /* regle(code_regle)                  */
suivant(string,string)                   /* suivant(Modele1,Modele2)           */


/*****************************************************************************
                CLAUSES DE LA BASE DE FAITS DES SPECIFICATIONS
*****************************************************************************/

condition(string)                        /* condition(code_condition)           */
generates(string,string,string)          /* generates(processus,message,condition) */
interface(string)                        /* interface(code_interface)          */
modele_courant(string)                   /* modele_courant(code_modele)        */
message(string)                          /* message(code_message)              */
part_of(string,string)                   /* part_of(processus_fils,processus_pere) */
proc_ou_inter(string)                    /* proc_ou_inter(processsus_ou_interface) */
process(string)                          /* process(code_processus)            */
receives(string,string)                  /* receives(processus,message)        */
triggered_by_term(string,string,string)  /*triggered_by_term(proc,proc,cond)    */
triggered_by_gen(string,string)          /* triggered_by_gen(processus,message) */
true_while(string,string)                /* true_while(condition,texte)        */
```

```
/*************************************************************************
                CLAUSES ASSOCIEES AUX BASES DE REGLES
*************************************************************************/

ensemble_utilise(string,string)      /* ensemble_utilise(regle,ensemble)       */
expression_informelle(string,string)/* expression_informelle(regle,expr_infor)*/
expression_informelle_mr(string,string)
type_regle(string,integer)           /* flag indiquant la detection d'erreur   */
erreur_detectee
```

• • •

- inclusion des outils du toolbox
- déclaration des prédicats
- inclusion de la base des règles
- inclusion de la base des méta-règles

```
/*************************************************************************
                          CORPS DU PROGRAMME
*************************************************************************/

GOAL
     def_methode,
     verif_methode.


CLAUSES

/************* MODULE DE DIALOGUE ET DE CONFIGURATION DE METHODES *************/

def_methode :-
     initialisation1,
     def_modeles,
     def_regles_modeles,
     conclusion1.

/************************** MODULE DE VERIFICATION ****************************/

verif_methode:-
     initialisation2,
     meta_regle(Mr),
     verif_meta_regle(Mr),
     fail.
verif_methode:-
     conclusion2.


/*************************************************************************/
/*                  INITIALISATION1                                      */
/*************************************************************************/

initialisation1 :-
     nettoyage,
     consult("ecran1.scr"),                  /* charge definition de l'ecran1 */
     consult("ecran2.scr"),                  /* charge definition de l'ecran2 */
     makewindow(1,7,7,"SOUS-SYSTEME DE CONFIGURATION",0,0,25,80),nl,
     lecture_nom_methode.
```

• • •

clauses: - nettoyage
- lecture_nom_méthode

```
/*****************************************************************************
                DEFINITION DES MODELES
*****************************************************************************/


def_modeles:-
     shiftscreen(ecran1),                          /* charge definition d'ecran1  */
     makewindow(10,47,15,"MODELES (ETAPES) DE LA METHODE",2,3,20,40),
     scrhnd(off,_),                                /* programme de saisie d'ecran */
     assert_modeles,                               /* genere clauses              */
     retract_value,                                /* nettoyage                   */
     removewindow.
```

• • •

- traitement des écrans de saisie des modèles

```
/************************ GENERE CLAUSES DE DEFINITION DES MODELES ************/

assert_modeles:-
     value(ecr1_modele1,M1),
     assertz(modele(M1)),
     assertz(modele_initial(M1)),
     value(ecr1_modele2,M2),
     assertz(modele(M2)),
     assertz(suivant(M1,M2)),
     value(ecr1_modele3,M3),
     assertz(modele(M3)),
     assertz(suivant(M2,M3)),
     value(ecr1_modele4,M4),
     assertz(modele(M4)),
     assertz(suivant(M3,M4)),
     value(ecr1_modele5,M5),
     assertz(modele(M5)),
     assertz(suivant(M4,M5)),
     value(ecr1_modele6,M6),
     assertz(modele(M6)),
     assertz(suivant(M5,M6)),
     value(ecr1_modele7,M7),
     assertz(modele(M7)),
     assertz(suivant(M6,M7)).
assert_modeles.
```

```
/*****************************************************************************
                DEFINITION DES REGLES ASSOCIEES AUX MODELES
*****************************************************************************/

def_regles_modeles:-
     shiftscreen(ecran2),                          /* charge definition d'ecran2  */
     modele(M),                                    /* selectionne un modele       */
     def_regles_un_modele(M),                      /* definition des regles d'un  */
     fail.                                         /* modele                      */
def_regles_modeles.


def_regles_un_modele(M):-
     upper_lower(MMaj,M),
     concat("MODELE: ",MMaj,Titre),
     saisie_ecran2(Titre,M,process),
     saisie_ecran2(Titre,M,message),
     saisie_ecran2(Titre,M,interface),
     saisie_ecran2(Titre,M,condition),
     saisie_ecran2(Titre,M,totale),
     !.

saisie_ecran2(Titre,M,process):-
     makewindow(10,47,15,Titre,2,0,23,80),
     retract(txtfield(1,_,_,_)),
     asserta(txtfield(1,0,77," Regles associees a la fin de la saisie d'un ecran
de definition de PROCESSUS")),
     scrhnd(off,_),
     assert_contient(M,fin_spec_process),
     retract_value,
     removewindow.
```

• • • - saisie des autres écrans

```
/************** GENERE CLAUSES RELATIVES AUX REGLES D'UN MODELE ***************/

assert_contient(M,Ev):-
     value(ecr2_regle1,R1),
     genere_contient(M,R1),
     concat(Ev,"_",Ev1),
     concat(Ev1,M,P),
     assertz(paquet(P)),
     assertz(contient(P,R1)),
     assertz(declenche(Ev,M,P)),
     assertz(comprend(M,P)),
     value(ecr2_regle2,R2),
     genere_contient(M,R2),
     assertz(contient(P,R2)),
     value(ecr2_regle3,R3),
     genere_contient(M,R3),
     assertz(contient(P,R3)),
     value(ecr2_regle4,R4),
     genere_contient(M,R4),
     assertz(contient(P,R4)),
     value(ecr2_regle5,R5),
```

```
     genere_contient(M,R5),
     assertz(contient(P,R5)),
     value(ecr2_regle6,R6),
     genere_contient(M,R6),
     assertz(contient(P,R6)),
     value(ecr2_regle7,R7),
     genere_contient(M,R7),
     assertz(contient(P,R7)),
     value(ecr2_regle8,R8),
     genere_contient(M,R8),
     assertz(contient(P,R8)).
assert_contient(_,_).

genere_contient(M,R):-
     not(contient(M,R)),
     assertz(contient(M,R)),!.
genere_contient(_,_):- !.


/*****************************************************************************
                              CONCLUSION1
*****************************************************************************/

conclusion1:-
     faits_derives,
     removewindow.

faits_derives :-
     creer_evenements,
     completer_modeles.
```

• • • - création de faits dérivés d'autres faits

```
/*****************************************************************************/
/*                      VERIFICATION DE LA METHODE DEFINIE                   */
/*****************************************************************************
                              INITIALISATION2
*****************************************************************************/

initialisation2:-
     methode(Met),
     upper_lower(MetMaj,Met),
     concat("VERIFICATION DE LA METHODE ",MetMaj,Titre),
     makewindow(20,7,15,Titre,0,0,25,80),
     nl,nl,nl,
     write(" DEBUT DES VERIFICATIONS"),nl,
     !.
```

```
/*****************************************************************************
                CONCLUSION2
*****************************************************************************/


conclusion2 :-
     nl,nl,
     write(" FIN DES VERIFICATIONS"),
     retract_champs_aux,
     cursor(Lin,_),
     Lin = 6,
     methode(M),
     concat(M,".met",Nom_fichier),
     nl,nl,write("Enregistrant le fichier ",Nom_fichier),
     save(Nom_fichier),
     retract_methode,
     readchar(_),
     removewindow,
     removewindow.
conclusion2:-
    nl,nl,
    write(" Methode non creee."),
    retract_methode,
    readchar(_),
    removewindow,
    removewindow.
```

B.4 - Base de connaissances de définition d'une méthode particulière

```
comprend("analyse statique","fin_spec_process_analyse statique")
comprend("analyse statique","fin_spec_message_analyse statique")
comprend("analyse statique","fin_spec_interface_analyse statique")
comprend("analyse statique","fin_spec_modele_analyse statique")
comprend("analyse dynamique","fin_spec_process_analyse dynamique")
comprend("analyse dynamique","fin_spec_message_analyse dynamique")
comprend("analyse dynamique","fin_spec_interface_analyse dynamique")
comprend("analyse dynamique","fin_spec_condition_analyse dynamique")
comprend("analyse dynamique","fin_spec_modele_analyse dynamique")
contient("analyse statique","r1")
contient("fin_spec_process_analyse statique","r1")
contient("analyse statique","r2")
contient("fin_spec_process_analyse statique","r2")
contient("analyse statique","r4")
contient("fin_spec_process_analyse statique","r4")
contient("analyse statique","r7")
contient("fin_spec_process_analyse statique","r7")
contient("analyse statique","r8")
contient("fin_spec_process_analyse statique","r8")
contient("analyse statique","r13")
contient("fin_spec_message_analyse statique","r13")
contient("analyse statique","r15")
contient("fin_spec_message_analyse statique","r15")
contient("analyse statique","r16")
contient("fin_spec_message_analyse statique","r16")
contient("analyse statique","r17")
contient("fin_spec_interface_analyse statique","r17")
contient("analyse statique","r5")
contient("fin_spec_modele_analyse statique","r5")
contient("analyse statique","r9")
contient("fin_spec_modele_analyse statique","r9")
contient("analyse statique","r10")
contient("fin_spec_modele_analyse statique","r10")
contient("analyse statique","r11")
contient("fin_spec_modele_analyse statique","r11")
contient("analyse statique","r12")
contient("fin_spec_modele_analyse statique","r12")
contient("analyse statique","r14")
contient("fin_spec_modele_analyse statique","r14")
contient("analyse statique","r18")
contient("fin_spec_modele_analyse statique","r18")
contient("analyse dynamique","r3")
contient("fin_spec_process_analyse dynamique","r3")
contient("analyse dynamique","r4")
contient("fin_spec_process_analyse dynamique","r4")
contient("analyse dynamique","r6")
contient("fin_spec_process_analyse dynamique","r6")
contient("analyse dynamique","r7")
contient("fin_spec_process_analyse dynamique","r7")
contient("analyse dynamique","r8")
contient("fin_spec_process_analyse dynamique","r8")
contient("analyse dynamique","r13")
contient("fin_spec_message_analyse dynamique","r13")
contient("analyse dynamique","r15")
contient("fin_spec_message_analyse dynamique","r15")
```

```
contient("analyse dynamique","r16")
contient("fin_spec_message_analyse dynamique","r16")
contient("analyse dynamique","r17")
contient("fin_spec_interface_analyse dynamique","r17")
contient("analyse dynamique","r19")
contient("fin_spec_condition_analyse dynamique","r19")
contient("analyse dynamique","r5")
contient("fin_spec_modele_analyse dynamique","r5")
contient("analyse dynamique","r9")
contient("fin_spec_modele_analyse dynamique","r9")
contient("analyse dynamique","r10")
contient("fin_spec_modele_analyse dynamique","r10")
contient("analyse dynamique","r11")
contient("fin_spec_modele_analyse dynamique","r11")
contient("analyse dynamique","r12")
contient("fin_spec_modele_analyse dynamique","r12")
contient("analyse dynamique","r14")
contient("fin_spec_modele_analyse dynamique","r14")
contient("analyse dynamique","r18")
contient("fin_spec_modele_analyse dynamique","r18")
contient("fin_spec_modele_analyse dynamique","r19")
contient("fin_spec_modele_analyse statique","r1")
contient("fin_spec_modele_analyse statique","r2")
contient("fin_spec_modele_analyse statique","r4")
contient("fin_spec_modele_analyse statique","r7")
contient("fin_spec_modele_analyse statique","r8")
contient("fin_spec_modele_analyse statique","r13")
contient("fin_spec_modele_analyse statique","r15")
contient("fin_spec_modele_analyse statique","r16")
contient("fin_spec_modele_analyse statique","r17")
contient("fin_spec_modele_analyse dynamique","r3")
contient("fin_spec_modele_analyse dynamique","r4")
contient("fin_spec_modele_analyse dynamique","r6")
contient("fin_spec_modele_analyse dynamique","r7")
contient("fin_spec_modele_analyse dynamique","r8")
contient("fin_spec_modele_analyse dynamique","r13")
contient("fin_spec_modele_analyse dynamique","r15")
contient("fin_spec_modele_analyse dynamique","r16")
contient("fin_spec_modele_analyse dynamique","r17")
declenche("fin_spec_process","analyse statique","fin_spec_process_analyse
          statique")
declenche("fin_spec_message","analyse statique","fin_spec_message_analyse
          statique")
declenche("fin_spec_interface","analyse statique","fin_spec_interface_analyse
          statique")
declenche("fin_spec_modele","analyse statique","fin_spec_modele_analyse
          statique")
declenche("fin_spec_process","analyse dynamique","fin_spec_process_analyse
          dynamique")
declenche("fin_spec_message","analyse dynamique","fin_spec_message_analyse
          dynamique")
declenche("fin_spec_interface","analyse dynamique","fin_spec_interface_analyse
          dynamique")
declenche("fin_spec_condition","analyse dynamique","fin_spec_condition_analyse
          dynamique")
declenche("fin_spec_modele","analyse dynamique","fin_spec_modele_analyse
          dynamique")
```

```
evenement("fin_spec_process")
evenement("fin_spec_message")
evenement("fin_spec_interface")
evenement("fin_spec_modele")
evenement("fin_spec_condition")
methode("demthese")
modele("analyse statique")
modele("analyse dynamique")
modele_initial("analyse statique")
paquet("fin_spec_process_analyse statique")
paquet("fin_spec_message_analyse statique")
paquet("fin_spec_interface_analyse statique")
paquet("fin_spec_modele_analyse statique")
paquet("fin_spec_process_analyse dynamique")
paquet("fin_spec_message_analyse dynamique")
paquet("fin_spec_interface_analyse dynamique")
paquet("fin_spec_condition_analyse dynamique")
paquet("fin_spec_modele_analyse dynamique")
suivant("analyse statique","analyse dynamique")
```

B.5 - Interface de saisie et de contrôle de spécifications

```
code=6000
/*************************************************************************/
/*  SYSTEME DE PILOTAGE DE LA MODELISATION DE SYSTEMES D'INFORMATION      */
/*                                                                        */
/*  INTERFACE DE SAISIE ET DE CONTROLE DE SPECIFICATIONS                  */
/*                                                                        */
/*  programme: PILOTAGE.PRO                                               */
/*  auteur:    Luis Alvares                                               */
/*  date:      aout/88                                                    */
/*                                                                        */
/*  entree: - une base de definition d'une methode, generee par le        */
/*            sous-systeme de configuration.                              */
/*          - une base initiale de faits des specifications d'une application */
/*  sortie: - une base de faits des specifications d'une application      */
/*************************************************************************/

/*************************************************************************/
/*             DECLARATIONS NECESSAIRES AUX OUTILS DU TOOLBOX            */
/*************************************************************************/

include "decltool.pro"

DATABASE
/*************************************************************************
                CLAUSES DE LA BASE DE FAITS DE SPECIFICATIONS
*************************************************************************/

condition(string)                       /* condition(code_condition)              */
generates(string,string,string)         /* generates(processus,message,condition) */
interface(string)                       /* interface(code_interface)              */
modele_courant(string)                  /* modele_courant(code_modele)            */
message(string)                         /* message(code_message)                  */
part_of(string,string)                  /* part_of(processus_fils,processus_pere) */
proc_ou_inter(string)                   /* proc_ou_inter(processsus_ou_interface) */
process(string)                         /* process(code_processus)                */
receives(string,string)                 /* receives(processus,message)            */
triggered_by_term(string,string,string) /*triggered_by_term(proc,proc,cond)       */
triggered_by_gen(string,string)         /* triggered_by_gen(processus,message)    */
true_while(string,string)               /* true_while(condition,texte)            */


/*************************************************************************
             CLAUSES DE DEFINITION DE LA METHODE A UTILISER
*************************************************************************/

comprend(string,string)                 /* comprend(modele,paquet)                */
contient(string,string)                 /* contient(modele_ou_paquet,regle)       */
declenche(string,string,string)         /* declenche(evenement,modele,paquet)     */
evenement(string)                       /* evenement(code_evenement)              */
methode(string)                         /* methode(code_methode)                  */
modele(string)                          /* modele(code_modele)                    */
modele_initial(string)                  /* modele_initial(code_modele)            */
paquet(string)                          /* paquet(code_paquet)                    */
regle(string)                           /* regle(code_regle)                      */
suivant(string,string)                  /* suivant(Modele1,Modele2)               */
```

• • • - clauses auxiliaires
- inclusion des outils du toolbox
- déclaration des prédicats
- inclusion de la base des règles

```
/*****************************************************************************
                    CORPS DU PROGRAMME
*****************************************************************************/

GOAL
     pilotage.


CLAUSES

pilotage :-
     initialisation,
     pulldown(7,                              /* programme de controle de menus */
                                  [ curtain(5, "Inf.Gen",[]),
                                    curtain(19,"Specification",
                                                ["Process",
                                                 "Message",
                                                 "Interface",
                                                 "Condition",
                                                 "Quit"] ),
               curtain(36,"Verification",
                               ["Mod.courant",
                                "Regle"]),
               curtain(52,"Documentation",[]),
               curtain(70,"Exit",[])
             ]
     ,_,_),
     conclusion.


/*****************************************************************************
                          INITIALISATION
*****************************************************************************/

initialisation :-
     nettoyage,
     makewindow(1,7,7,"SYSTEME DE PILOTAGE",0,0,25,80),
     nl,nl,
     lecture_nom_application,
     consult("defproc.scr"),
     consult("defmess.scr"),
     consult("definter.scr"),
     consult("defcond.scr"),
     clearwindow.
```

• • • clauses: - nettoyage
- lecture_nom_methode

```
/*****************************************************************************
              CHOIX DE L'OPTION DU MENU  (DANS PULLDOWN)
*****************************************************************************/

pdwaction(1,0) :-                       /* selection de INF.GEN. dans le menu */
     informations_generales.

pdwaction(2,1):-                        /* selection de PROCESS dans le menu  */

     asserta(ecran_courant(process)),
     specification(process).

pdwaction(2,2):-                        /* selection de MESSAGE dans le menu  */
     asserta(ecran_courant(message)),
     specification(message).

pdwaction(2,3):-                        /* selection d'INTERFACE dans le menu */
     asserta(ecran_courant(interface)),
     specification(interface).

pdwaction(2,4):-                        /* selection de CONDITION dans le menu*/
     asserta(ecran_courant(condition)),
     specification(condition).

pdwaction(3,1):-                        /* selection de MOD.COURANT           */
     verif_mod_courant.

pdwaction(3,2):-                        /* selection de REGLE dans le menu    */
     verif_regle.

pdwaction(4,0):-                        /* selection de DOCUMENTATION         */
     documentation.


/*****************************************************************************
                    SAISIE DES SPECIFICATIONS
*****************************************************************************/

specification(process) :-
     shiftscreen(defproc),              /* definition de l'ecran de processus */
     makewindow(12,47,7,"PROCESS",3,15,19,55),
     scrhnd(off,_),                     /* activation de l'ecran de processus */
     removewindow,
     !.
```

• • • - traitements associés aux écrans

```
/*****************************************************************************
  mise a jour de la base de faits d'une application apres la saisie d'un ecran
*****************************************************************************/

m_a_j_specif(message) :-
        value(message,M),
        not(message(M)),
        assertz(message(M)),
        assertz(temp2(message,M)),
        fail.
m_a_j_specif(message) :-
        value(received_by,PI),
        not(process(PI)),
        not(interface(PI)),
        not(proc_ou_inter(PI)),
        assertz(proc_ou_inter(PI)),
        assertz(temp2(proc_ou_inter,PI)),
        fail.
m_a_j_specif(message) :-
        value(received_by,P),
        value(message,M),
        not(receives(P,M)),
        assertz(receives(P,M)),
        assertz(temp3(receives,P,M)),
        fail.
m_a_j_specif(message) :-
        value(generated_by,PI),
        not(process(PI)),
        not(interface(PI)),
        not(proc_ou_inter(PI)),
        assertz(proc_ou_inter(PI)),
        assertz(temp2(proc_ou_inter,PI)),
        fail.
m_a_j_specif(message) :-
        value(generated_by,P),
        value(message,M),
        verif_exist_generates(P,M),
        assertz(generates(P,M,"")),
        assertz(temp4(generates,P,M,"")),
        fail.
m_a_j_specif(message) :-
        value(if_generated_by,C),
        not(condition(C)),
        assertz(condition(C)),
        assertz(temp2(condition,C)),
        fail.
m_a_j_specif(message) :-
        value(if_generated_by,C),
        value(generated_by,P),
        value(message,M),
        verif_exist_generates(P,M),
        assertz(generates(P,M,C)),
        assertz(temp4(generates,P,M,C)),
        fail.
m_a_j_specif(message) :-
        !.
```

• • • - mises à jour associées aux autres écrans

```
/*****************************************************************************
                        VERIFICATION A POSTERIORI
*****************************************************************************/

verif_mod_courant :-
     modele_courant(M),
     menu(4,50,7,7,["Totale","Coherence","Completude"],M,1,Choix),
     exec_choix(Choix),
     retract_erreur_detecte.

verif_regle :-
     makewindow(13,7,7,"",3,0,22,80),
     lecture_code_regle,
     verification_regle(R),
     removewindow.


exec_choix(0).

exec_choix(1) :-
     activation(fin_spec_modele).

exec_choix(2) :-
     modele_courant(M),                  /* selection du modele courant        */
     declenche(fin_spec_modele,M,P),     /* selection d'un paquet declenche    */
     contient(P,R),                      /* selection d'une regle du paquet    */
     type_regle(R,T),                    /* determination du type de la regle  */
     T < 200,                            /* si est une regle de coherence      */
     verification_regle(R),              /* execution de la regle              */
     fail.
exec_choix(2).

exec_choix(3) :-
     modele_courant(M),
     declenche(fin_spec_modele,M,P),
     contient(P,R),
     type_regle(R,T),
     T > 200,                            /* si est une regle de completude     */
     verification_regle(R),
     fail.
exec_choix(3).


lecture_code_regle :-
     write(" Code de la regle: "),
     readln(R),
     regle(R).

lecture_code_regle :-
     write(" REGLE INEXISTANTE"),nl,
     beep,
     lecture_code_regle,!.
```

```
/*****************************************************************************
              Activation des Verifications
*****************************************************************************/

activation(Evenement) :-
     modele_courant(M),          /* selection du modele courant                */
     declenche(Evenement,M,P),   /* selection du paquet declenche par Evenement */
     contient(P,R),              /* selection d'une regle du paquet            */
     verification_regle(R),      /* execution de cette regle                   */
     fail.
activation(_) :- !.              /* condition d'arret                          */


/*****************************************************************************
                            CONCLUSION
*****************************************************************************/

conclusion :-
     retract_champs_aux,         /* suppression des champs aux. avant le "save" */
     retract_champs_aux2,
     application(A),             /* obtient le nom de l'application            */
     concat(A,".apl",Fichier),   /* montage du nom du fichier                  */
     save(Fichier),              /* stockage sur disque des specifications     */
     removewindow.
```

• • • - clause retract_champs_aux