Une contribution au Génie Automatique : le prototypage des machines et systèmes automatisés de production

Hervé Panetto

Nancy-Université
Université Henri Poincaré

AVERTISSEMENT

LIENS

THESE

présentée à

L'UNIVERSITE DE NANCY I

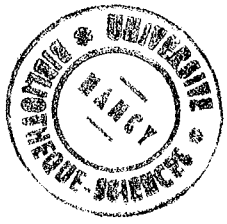
pour l'obtention du grade de

Docteur de l'Université de Nancy I

Option "Production Automatisée"

par

Hervé PANETTO



UNE CONTRIBUTION AU GENIE AUTOMATIQUE : LE PROTOTYPAGE DES MACHINES ET SYSTEMES AUTOMATISES DE PRODUCTION

Soutenue publiquement le 28 Janvier 1991 devant la Commision d'Examen composée de :

Président	R. HUSSON	Professeur à l'I.N.P.L. Directeur du C.R.A.N.
Rapporteurs	F. PRUNET	Professeur à l'U.S.T.L. Montpellier Directeur Dpt S.D.C.I.A. / L.A.M.M.
	J.F. AUBRY	Professeur à l'I.N.P.L.
Directeur de Thèse	M. VERON	Professeur à l'Université de Nancy I Directeur du L.A.C.N. / C.R.A.N.
Examinateurs	G. VILLERMAIN LECOLIER	Professeur à l'Université de Reims Directeur du L.A.M.
	F. CORBIER	Responsable Méthodes et Coordination des Moyens - SPIE AUTOMATION
	P. LHOSTE	Maître de Conférences à l'Université de Nancy I - L.A.C.N. / C.R.A.N.

RESUME : L'émergence du Génie Automatique est la conséquence de besoins en méthodes et outils adaptés aux métiers de l'Automatisation. La mise en oeuvre des principes largement éprouvés par le Génie Logiciel tels que structuration, modularité, réutilisation, prototypage des applications contribue ainsi à l'évolution du Génie Automatique et à l'amélioration de la qualité de la conception. SPEX, outil de conception pour le Génie Automatique, offre ces possibilités mais nécessite, pour assurer une cohérence globale du processus de développement, de pouvoir s'intégrer, ou tout du moins s'interconnecter, avec des outils assurant les phases amont (spécification) et aval (codage et tests hors site) de l'application. A terme, la définition d'un Atelier de Génie Automatique devrait centraliser l'ensemble des informations dans une base de données ou d'objets commune à l'ensemble des outils pour permettre une mémorisation du savoir-faire entreprise.

MOTS CLES : Méthode - Méthodologie - Modèles - SADT - Prototypage - Maquettage - Génie Automatique - Atelier Logiciel - Grafcet - Conception - Outil de Conception

ABSTRACT : Automation Engineering appearance results from needs of methods and tools for Automation "life cycle". Software Engineering concepts such as structuration, modularity, reutilisability, application prototype conception make a contribution to Automation Engineering evolution and to conception quality amelioration. SPEX tool implements these possibilities but requires, for global consistency of developpement process, to become integrated, or connected, with specification, code production and test tools. At short-dated, the definition of an Automation Engineering CASE would centralize information on common data or object base for know-how memorization.

KEYWORDS : Method, Models, SADT, Prototype conception, Automation Engineering, CASE tool, State charts, Conception, Conception tool

SOMMAIRE GENERAL

CHAPITRE II : SEMANTIQUE DES OBJETS D'AUTOMATISATION

CHAPITRE III : DE L'INTERCONNEXION D'OUTILS

CHAPITRE IV : ... A LA SPECIFICATION D'UN ATELIER DE GENIE AUTOMATIQUE PARTICULIER

Introduction - Problématique

INTRODUCTION - PROBLEMATIQUE

Cadre Général de l'étude

La complexité croissante des Machines et Systèmes Automatisés de Production (MSAP) ne permet plus leur conception par les méthodes traditionnelles telles que l'ont montré les travaux sur le Poste de Travail pour l'Automaticien [PTA 87]. Il est maintenant plus que nécessaire de mettre en oeuvre, dans le domaine de l'automatisation, des principes largement éprouvés par le **Génie Logiciel** tels que structuration, modularité, réutilisation, *prototypage* des applications, contribuant ainsi à l'évolution du **Génie Automatique.**

Face à cette évolution nécessaire, des besoins en modélisation se sont rapidement fait sentir et ont nécessité des **méthodes** permettant de mieux structurer la pensée des concepteurs. Des **méthodes** de spécification et de conception s'avèrent donc indispensables pour favoriser une meilleure compréhension des besoins d'automatisation et doivent, pour être praticables, être supportées par des outils informatisés ou **outils-méthodes.**

Le développement d'un système automatisé nécessite, au cours des différentes phases de modélisation, une collection d'outils aptes à supporter des concepts importés du **Génie Logiciel** et ayant des liens "homogènes" entre eux pour éviter toutes incohérences et erreurs d'interprétation.

Réutilisation, Qualité et Productivité

La création de véritables **composants logiciels réutilisables** (composants génériques) relève de différents niveaux **d'abstraction** par une combinaison des niveaux conceptuel, logique ou physique avec les modèles générique, partiel ou particulier. En effet, les objectifs définis pour de tels composants ne sont pas indépendants de l'"horizon" de généricité déterminé par les compétences propres des concepteurs.

Il est cependant certain que la création et l'enrichissement de tels composants tout au long du cycle de développement d'un Système Automatisé, depuis sa spécification jusqu'à son exploitation et sa maintenance, doit permettre une amélioration des gains en

productivité ainsi que de la **qualité** et de la **maîtrise** des Systèmes Automatisés de Production Manufacturière.

Maquettage et Prototypage

Contrairement aux démarches de validation courantes qui ne sont effectuées qu'en phase de recette, d'intégration et de test en plateforme (Figure 1a), alors qu'il est déjà trop tard, économiquement pour une remise en cause de l'analyse, il semble que les cycles de développement futurs (mais pas si lointain que cela) doivent prendre en compte la validation à chaque phase du cycle de vie d'un Système Automatisé [CHO 88], de la *spécification* ... à *l'exploitation* (Figure 1b).

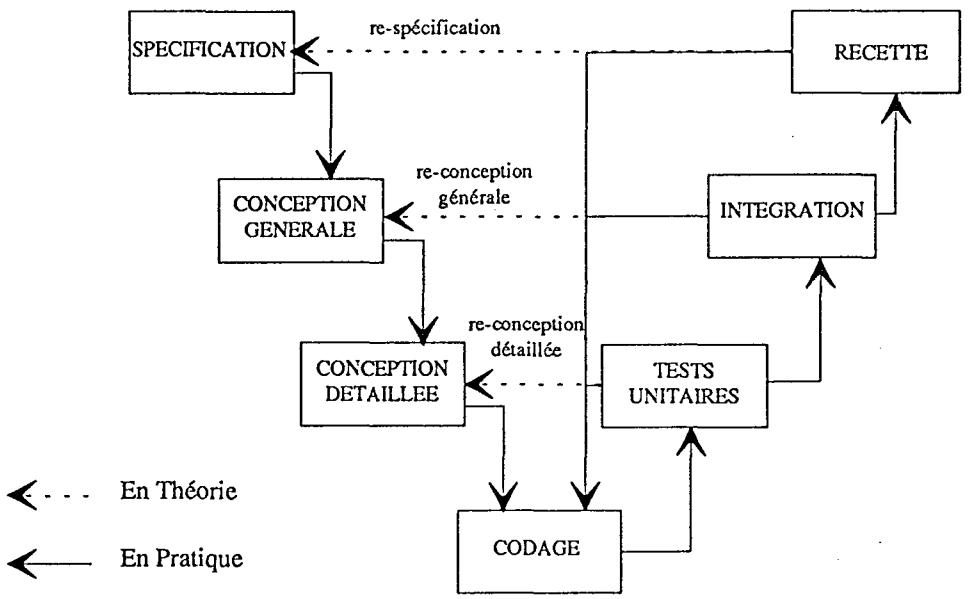


Figure 1.a : Place de la validation dans le cycle de développement pratique d'un automatisme

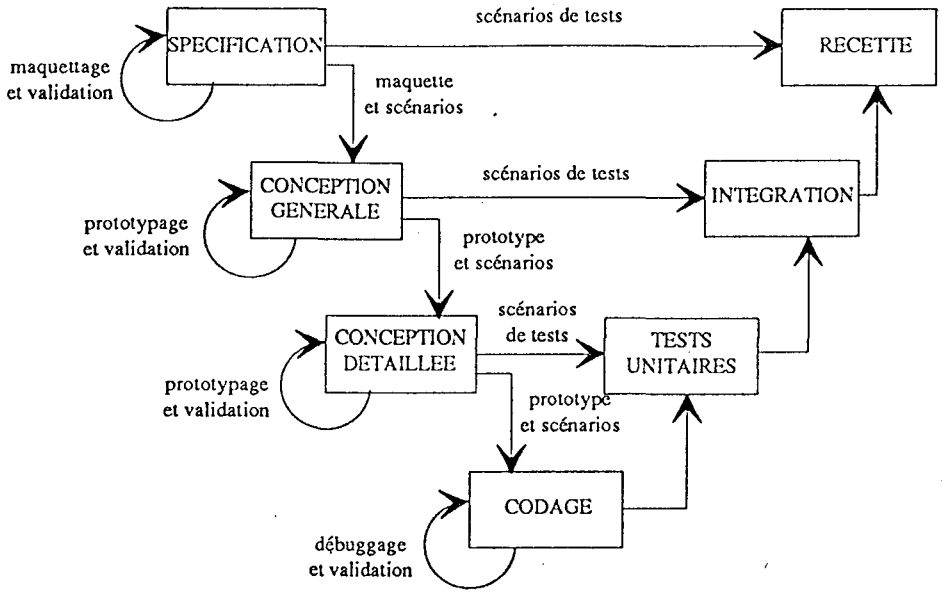


Figure 1.b : Place de la validation dans le cycle de développement actuel d'un automatisme

La nécessité de valider **au plus tôt** le résultat de l'analyse n'est envisageable, à chaque stade de la modélisation, que si elle est **exécutable**, ceci pour compléter efficacement non seulement la conception d'un système mais aussi et surtout sa spécification.

Une **spécification** devient **exécutable**, d'après [FOR 90], si elle est exprimée dans un langage qui possède une sémantique opérationnelle, c'est à dire proche du langage d'implémentation.

Il doit donc être possible de décrire une "vue externe" du système à l'aide d'une **maquette**, représentant la structure de l'ensemble des composants ainsi que leurs relations sans en exprimer le comportement détaillé. La maquette cherchera à répondre à la question :

Comment être sûr que l'application que nous allons réaliser répond bien aux attentes de ses futurs utilisateurs ?
Elle mesure ainsi l'adéquation entre les besoins et les premiers résultats d'analyse.

A partir d'une maquette validée, peut être réalisée la "vue interne" du système pour l'élaboration d'un **prototype** qui décrit les comportements internes de chaque entité dans un langage proche de l'implémentation [BOU 90] mais indépendamment des moyens utilisés dans l'application réelle.

Ce prototype peut être de deux types en fonction de la finalité envisagée [CHO 88] :

- prototype **incrémental** s'il est réutilisé partiellement et complété pour constituer l'application finale, option prise dans le cas de l'outil SPEX (présenté dans ce mémoire au chapître I),
- prototype **jetable** s'il n'est destiné qu'à mettre à l'épreuve rapidement certaines hypothèses indépendamment de choix réalisationnels ultérieurs.

Pour la validation de la spécification, et de la conception, un outil logiciel doit permettre:

- la construction d'une **maquette** par la définition de **l'interface** de chaque entité participant à la modélisation du système sans en décrire, à ce niveau de spécification, le comportement détaillé. **L'exécution d'une telle spécification** permet de focaliser l'attention sur la structure d'ensemble des entités et leurs relations mutuelles sans en connaître leur comportement interne;

- l'élaboration d'un **prototype** de l'application par un enrichissement des composants de la maquette. **L'exécution de cette conception** fournit des résultats sur la validité des comportements internes des entités et des contraintes de synchronisation qu'elles induisent.

Pour résumer, la maquette est une spécification exécutable de l'application validant la structure, le prototype décrit une conception exécutable de l'installation vérifiant l'adéquation de la solution proposée par rapport aux spécifications initiales du cahier des charges. En référence à la modélisation en diachronie du cycle de vie proposé par [VOG 88], nous pouvons situer le maquettage et le prototypage sur la Figure 2 :

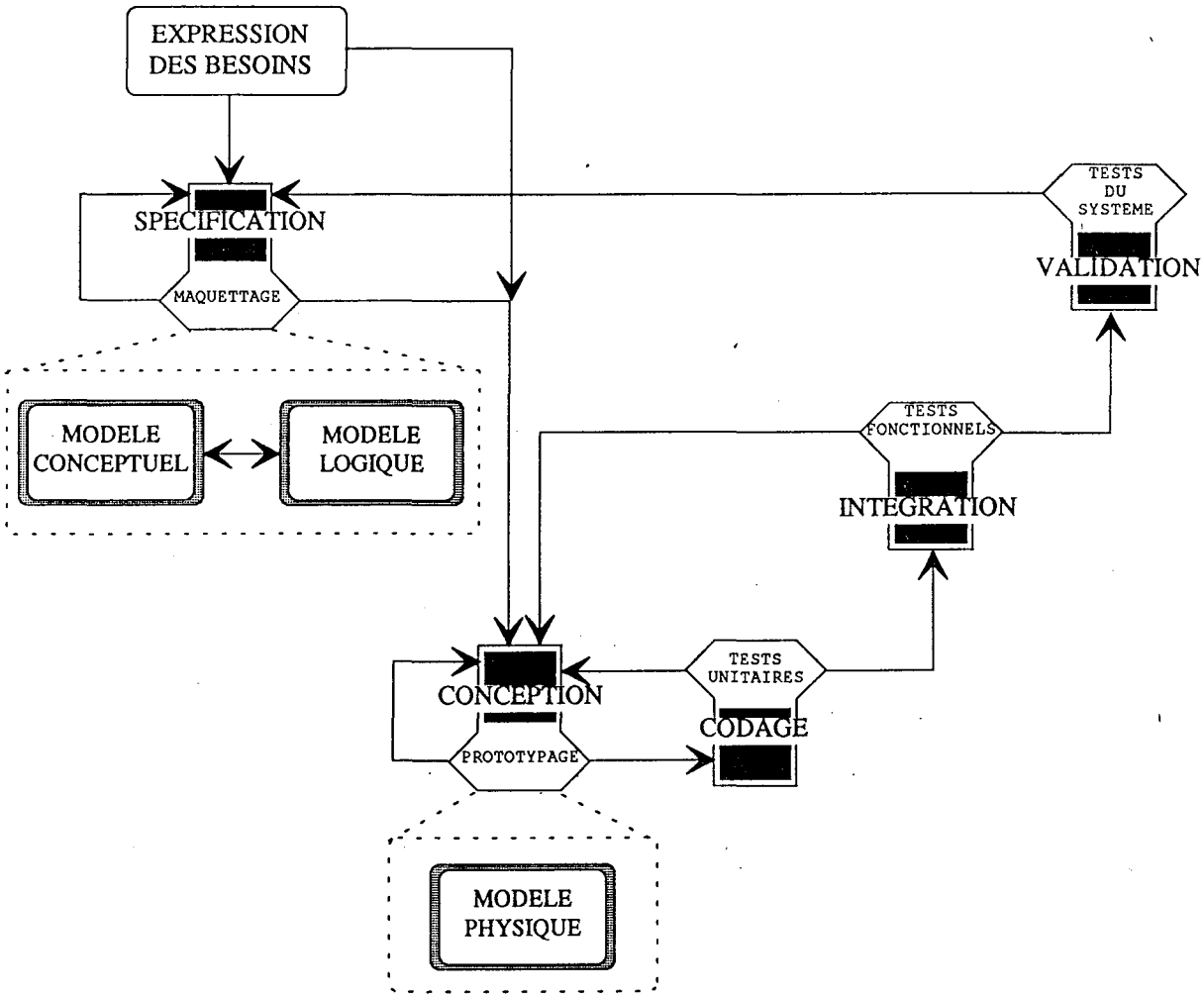


Figure 2. : Place de la maquette et du prototype dans le cycle de développement d'une entité

Plan du mémoire :

Notre contribution au Génie Automatique est présentée, dans ce mémoire, en cinq Chapitres principaux :

- le Chapitre I présente l'outil SPEX et son assistance à la structuration, à la réutilisation, au maquettage et prototypage par l'utilisation de concepts inspirés du Génie Logiciel ;

- le Chapitre II montre que, avant de réutiliser, il est nécessaire de définir ce qui est réutilisable. Les approches en Génie Logiciel n'offrant pas de solutions pour la détermination précise des objets, nous présentons quelques approches liées au Génie Automatique pour des classes d'objets particuliers ;

- le Chapitre III expose notre approche pour l'interconnexion possible de SPEX avec des outils supportant les phases amont et aval de la conception de Systèmes Automatisés ;

- après un bilan des Ateliers de Génie Logiciel et une présentation des recherches en cours pour la définition d'un Atelier de Génie Automatique, le Chapitre IV présente nos perspectives pour la définition d'un Atelier de Génie Automatique particulier ;

- enfin, nous concluons par les perspectives d'évolution de l'outil SPEX pour son intégration dans un Atelier de Génie Automatique.

L'annexe présente un extrait d'un exemple d'utilisation de la chaîne d'outils (ORCHIS-GRILLE-SPEX) pour la conception de la partie commande d'une Machine Transfert.

SOMMAIRE
Chapitre I

I. SPEX : UN OUTIL, DEUX METIERS

1. Définition du Génie Automatique

Un M.S.A.P. (Machine et Système Automatisé de Production) est défini dans [FRA 87] par :

> **"Un MSAP est un ensemble de dispositifs divers, de constituants, catalogués et le plus souvent standards, agencés, organisés, programmés en vue d'assurer, avec une intervention humaine réduite, des fonctions de production industrielles prédéfinies"**

L'identification des divers constituants standards d'un MSAP ainsi que leur modélisation d'un point de vue commande nécessitent des méthodes, langages et outils favorisant le développement de modèles génériques. Leur utilisation massive implique une qualité intrinsèque de leur modélisation indépendamment du domaine d'application qui les utilise. C'est en ce sens que le Génie Automatique doit apporter des solutions comme cela fut le cas, il y a quelques années, pour le Génie Logiciel vis à vis du développement logiciel.

[FRA 87] propose :

> **"Le Génie Automatique est l'ensemble des questions scientifiques et techniques posées par l'application concrète des théories de l'Automatique en milieu industriel"**

mais nous préférons, dans une optique d'intégration des aspects méthodes, langages et outils de conception, une définition dérivée de celle du Génie Logiciel [MEY 88] donnée dans [MOR 88] :

> **"On appelle Génie Automatique l'application de méthodes scientifiques au développement de théories, méthodes, techniques, langages et outils favorisant la production de systèmes automatisés de qualité"**

en remarquant que le terme de production recouvre des aspects aussi bien technologiques que logiciels et méthodologiques sans oublier la maintenance. La notion de qualité est présente tant au point de vue réalisationnel que conceptuel.

La ressemblance avec le Génie Logiciel n'est que partielle puique l'automaticien doit aussi tenir compte d'une partie matérielle étroitement liée à la partie logicielle et induisant un grand nombre de contraintes.

2. Les besoins en Génie Automatique

Depuis plusieurs années, la modernisation de l'appareil industriel, motivée par le besoin d'amélioration de la qualité de la production, a provoqué une demande croissante en Machines et Systèmes Automatisés de Production .

A la crise du logiciel de commande, identique à celle identifiée en Informatique, et aux développements "anarchiques" d'outils de développement pour l'Automatisation, les industriels français, en tant que clients, ont réagi à cette situation dès 1984 par la création du projet **PTA** [ALA 84], regroupant la RNUR, PSA, MICHELIN, l'ADEPA ainsi que SGN avec pour objectif la spécification d'un Poste de Travail pour l'Automaticien, afin d'assister la conception, la réalisation et l'exploitation des Systèmes Automatisés.

Les résultats les plus importants, présentés dans le rapport final du 08 janvier 1987 [PTA 87], concernaient plus particulièrement : la documentation, la représentation des données, la qualité, et la (ou les) méthodes à appliquer lors de la conception des MSAP.

Suite à ces travaux et dans le cadre de l'action du MIPTT/SERICS, ayant comme objectifs, l'étude de la base application du PTA (projet BASE-PTA [BAS 88]), la pré-étude de faisabilité d'un **Atelier Logiciel pour le Génie Automatique** [3IP 88], et le développement des premiers outils "informatisés" de cet atelier, notre travail dans le projet **SPEX** [TIX 89] s'est intéressé aux aspects de spécification et conception des logiciels de Partie Commande des MSAP dans la continuité des travaux du C.R.A.N./L.A.C.N.

Il convient de distinguer deux aspects à notre contribution :

- à cout terme : l'objectif de développement d'un outil industrialisable dans le cadre de la réponse à l'appel d'offres MIPTT/SERICS où notre contribution correspond à la définition, en collaboration avec la société SPIE TRINDEL (Intégrateur de Systèmes automatisés), des spécifications à intégrer dans un tel outil et la vérification, après prototypage par la société TNI (société de services en informatique), du respect de ces spécifications (erreurs de prototypage, "bugs", incomplétude de la spécification, ...) ;

- à moyen terme : la définition et le prototypage de nouvelles spécifications pour fournir des mécanismes complémentaires à l'outil lui permettant de supporter les résultats de travaux de recherche menés par l'équipe Automatisation Intégrée de Production du L.A.C.N., tant sur les **concepts** et **modèles** du Génie Automatique que sur les **méthodes** inhérentes.

Ces deux horizons se traduisent donc techniquement par deux versions de l'outil :

- une version en cours de développement par la société TNI sur la base de nos spécifications, ayant pour objectif de fournir rapidement un produit praticable par les automatiseurs sur des supports informatiques de type micro-ordinateur PC (sous l'environnement Windows) et de type station de travail (sous l'environnement X-Windows) ;

- un prototype intégrant des fonctionnalités différentes et complémentaires à celui développé par TNI montrant l'application de nos idées pour l'ouverture de SPEX aux travaux conceptuels et méthodologiques de Génie Automatique.

Notre contribution, que nous présentons dans ce chapitre, prend comme point de départ les travaux initiaux exposés en [TIX 89].

L'offre des constructeurs d'environnement de conception, qu'ils soient fournisseurs d'Automates Programmables Industriels (ORPHEE, XTEL, ...) ou qu'ils soient spécialisés dans le domaine logiciel (ASA, DESIGN-IDEF, GRAL) restent limitées à des mécanismes de décomposition permettant une amélioration certaine des applications quant à leur **structuration** mais où la **réutilisation** reste confinée à la manipulation de ces "objets" par copie différentielle.

Les apports du **Génie Logiciel** et en particulier les "**approches objet**" permettent, moyennant une certaine adaptation à notre domaine d'application, de "*marier*" "**structuration**" et "**réutilisation**" et ainsi offrir un environnement de spécification exécutable dédié aux Machines et Systèmes Automatisés de Production, par l'association d'une analyse **descendante** et d'une synthèse **ascendante** à partir de *composants réutilisables* décrits dans des langages familiers aux automaticiens.

L'identification et la création de composants génériques réutilisables nécessitent un travail intellectuel au moins égal à celui de la création d'une application sur la base de ces composants. Il existe donc deux métiers distincts ayant des objectifs différents, le **Génie Automatiseur**, homme "savoir-faire" de l'entreprise, apte à s'abstraire de l'applicatif pour la définition des **invariants** (fonctionnels [ALD 90] ou technologiques [ISM 83] [ADE 84]) de l'entreprise (génériques ou "standards" entreprise) et/ou des domaines d'application (partiel entreprise ou "standards" application) et l'**Automatiseur**, homme métier de l'automatisation, utilisant les méthodes, outils et concepts mis à sa disposition par le Génie Automatiseur, dans le monde de l' "application particulière".

En complément du "cycle de vie de l'automatisation" présenté dans [MOR 88], le **Génie Automatiseur** fournit des ressources d'automatisation (méthodes, outils, modèles génériques, ...) à l'**Automatiseur** en particularisant les productions du **Génie Automatique** (indépendantes du cycle de vie entreprise) au "savoir-faire entreprise et/ou domaine d'application". (Figure I.1). Le **Génie Automaticien** fournit les mécanismes généraux d'assistance à l'automatisation alors que le **Génie Automatiseur** "particularise" ces derniers pour les besoins de son entreprise et/ou de domaines d'application particuliers.

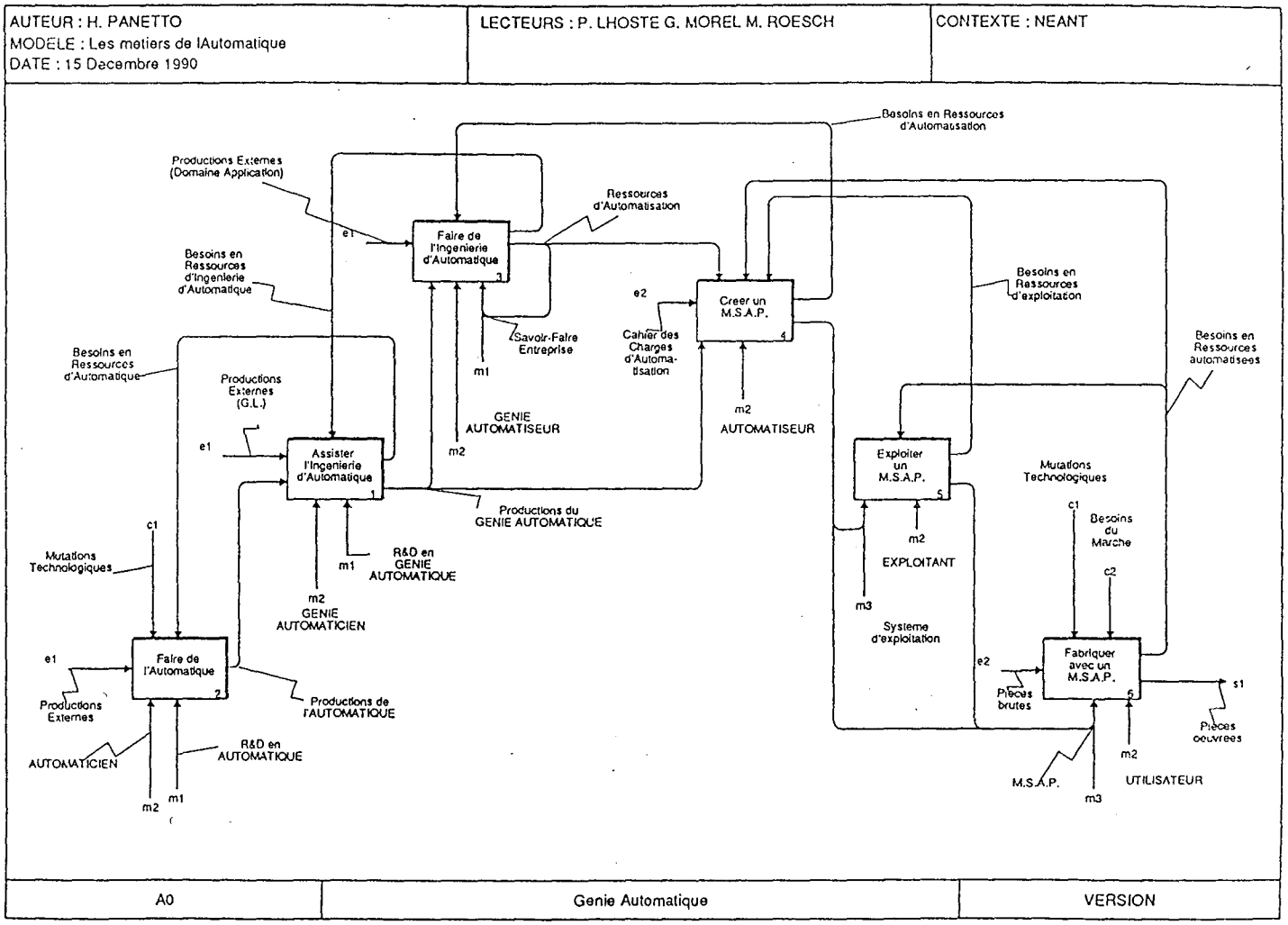


Figure I.1. : Les "métiers" de l'Automatique

SPEX est un outil issu du **Génie Automatique** en ce sens où les mécanismes qu'il fournit sont indépendants des domaines d'application et assistent le **Génie Automatiseur** pour la création d'une **base de savoir-faire entreprise** réutilisable pour la conception de M.S.A.P.

3. Principes importés du Génie Logiciel

3.1. La réutilisation

Thomas Standish [STA 84] observe que "*puisque l'analyse économique montre que le coût du logiciel est fonction exponentielle de la taille du logiciel, diviser par deux le logiciel à construire divise beaucoup plus que par deux le coût de sa construction. La réutilisation devient alors la clé de voûte dans bien des efforts actuels pour l'amélioration de la productivité*". La construction de petits composants de comportement bien maitrisés permet la définition de **classes fonctionnelles** de composants [ING 87] réutilisables à **forte cohérence** interne et **faible couplage** externe présentant trois avantages principaux [BOO 88] :

- réduction du coût de développement ;
- amélioration de la qualité du logiciel : plus un composant est réutilisé, plus il subit les épreuves de l'utilisation, ce qui incite à son amélioration constante et aboutit ainsi à une meilleure qualité intrinsèque;
- accélération de la production du logiciel.

Au delà des aspects qualité induits par de tels composants nous devons garder à l'esprit que, quelles que soient les méthodes de conception que nous employons, les capacités humaines à administrer la complexité sont limitées.

Quelle que soit l'approche choisie, nous devons être conscient que nous ne pouvons pas espérer avoir une connaissance complète et parfaite du domaine constituant l'application. Il est donc difficile de "trouver" les *bons composants* répondant au mieux à nos besoins et à la perspective d'une évolution. La façon dont notre compréhension progresse est plutôt un processus perpétuellement itératif [ALA 88a]. L'apparition de nouveaux aspects du problème au cours de notre conception n'est pas chose impossible, au contraire. Cependant, nous pouvons généralement limiter l'étendue de cette évolution aux seuls composants incriminés. Ces composants reflètent bien, chacun, une partie de la solution, indépendamment du reste de l'application.

3.2. Les qualités nécessaires à la réutilisation

[BOO 88], [ABB 86], [MEY 88] et beaucoup d'autres auteurs citent les qualités principales d'un **Composant Logiciel Réutilisable** :

- **adéquation** : Le logiciel répond à l'objectif exprimé par le demandeur. Cette qualité dépend de la bonne communication entre le demandeur et le spécifieur puis du spécifieur au concepteur. **Le produit doit être conforme au cahier des charges et à ses spécifications.**

- **efficacité** : Le logiciel doit fonctionner de façon optimale avec ses ressources disponibles. Une polarisation trop précoce des efforts pour une meilleure efficacité va au détriment de l'efficacité du système global. *"Une bonne perspicacité reflétant une compréhension plus unifiée d'un problème a beaucoup plus d'impact sur l'efficacité que n'importe quels tripotages de bits dans une structure déficiente"* [ROS 75]

- **fiabilité** *:* Le système doit fonctionner quelles que soient les circonstances et permettre de traiter les défauts et pannes éventuels par un fonctionnement en mode **"dégradé"**

- **maintenabilité** : Le développement du logiciel doit permettre la correction d'erreurs et son évolution en limitant les coûts que cela induit, ces derniers représentant jusqu'à 70% du coût du produit

Deux principes essentiels doivent être pris en compte pour assurer ces qualités :

- **abstraction** : l'abstraction consiste à porter son effort sur une représentation simplifiée suffisante du problème. Il est donc nécessaire de décomposer l'application en **niveaux** suffisament simples pour une meilleure compréhension du système et en extraire les détails essentiels au niveau concerné. A chaque niveau de notre décomposition, il est aussi possible d'en extraire des "invariants" permettant une **généralisation** de composants élémentaires réutilisables.

- **encapsulation** : encore appelée *dissimulation d'information* consiste à regrouper au sein d'entités toutes les décisions de conception de plus bas niveau et les données propres à la réalisation de la fonction pour ne donner accès à l'utilisateur qu'à une vue simplifiée du modèle. Il y a donc séparation complète entre **comportement** et **réalisation**, entre le **QUOI** et le **COMMENT**.

3.3. La notion d'objet

3.3.1. Définition

L'**objet** est une abstraction d'une partie de la solution qui permet de répondre aux 2 principes précédemment cités et ainsi satisfaire aux critères de qualité souhaités pour le logiciel [BOO 88] [MEY 88]. Un objet est une entité qui a un état (ses variables d'instances ou attributs) et dont le comportement est défini par les actions qu'il peut subir ou les services qu'il peut demander à d'autres objets. Ces actions décrivent les **services** que peut rendre l'objet à un demandeur. C'est l'entité **logique** de base manipulée à travers une **interface publique**, seule partie connue de l'utilisateur (Figure I.2).

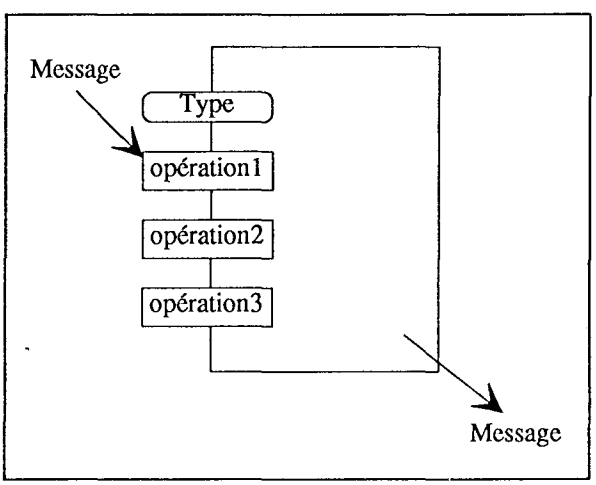


Figure I.2. : Un objet selon [BOO88]

3.3.2. La notion de messages

Un objet ne communique avec un autre objet que par l'envoi d'un **message**. La réception d'un message active un comportement particulier (ou méthode) qui modifie éventuellement l'état de l'objet et lui permet d'envoyer des messages à d'autres objets.

3.3.3. La notion de classe

Une **classe** est une famille d'objets. Elle est elle-même éventuellement un objet, qui décrit toujours toutes les caractéristiques (messages) et propriétés d'une famille. Tout objet manipulé est un élément d'une famille ou encore instance d'une classe et réagit en fonction du comportement défini dans sa classe.

3.3.4. L'instanciation

Un objet est créé par un processus d'**instanciation** d'une classe. Il possède alors un identificateur et une valuation particulière de ses variables d'instances. L'instanciation **duplique** les variables d'instances et **réutilise** le comportement défini dans les méthodes de sa classe.

3.3.5. L'héritage

Une classe peut être définie par rapport à une autre classe. Elle **hérite** alors de l'ensemble des propriétés et méthodes de sa classe mère. Elle peut, éventuellement définir ses propres attributs et ses propres méthodes. Elle peut redéfinir des méthodes héritées de sa classe mère pour implémenter son propre comportement, c'est ce que l'on appelle "surcharge".

3.4. SPEX et le Génie Logiciel

L'outil SPEX permet la définition de **classes** d'entités, appelées "**Boîtes Fonctionnelles**" (BF), identifiables aux *classes* des concepts objets. Ces composants possèdent une interface publique, un ensemble de paramètres (variables d'instances) et définissent un comportement.

Un processus d'instanciation de BF permet de créer un exemplaire particulier (ou instance) ayant ses propres paramètres (variables d'instances) et réutilisant le comportement défini dans sa "classe".

La culture des automatiseurs étant très imprégnée par les Automates Programmables Industriels et leurs lots de cartes d'E/S, l'ensemble des objets ne communiquent pas, dans l'outil SPEX, par des messages mais par un positionnement de variables d'entrées et/ou de sorties.

Une "Boîte Fonctionnelle" (BF), "objet" élémentaire manipulé par l'outil, représente un comportement d'automatisation, qui calcule des valeurs de sorties en fonction de ses valeurs d'entrées et de ses paramètres éventuels. La notion **d'héritage**, au sens "classes d'objets" ne peut pas raisonnablement s'appliquer à un tel comportement mais pourrait être utilisée pour permettre à une BF d'hériter de l'interface d'une autre BF. Ce type d'utilisation de l'héritage nous paraît difficile à mettre en oeuvre par l'"Automatiseur" et même pour le "Génie Automatiseur" car il nécessite la définition préalable d'un ensemble de "**classes abstraites**" représentatives de l'ensembles des interfaces possibles (et utilisées) pour tout comportement. Si ce type d'entités "abstraites" semble réalisable (et identifiable) pour la définition de comportements d'équipements d'automatismes (catalogues constructeurs) [COR 89] [MAX 91], il n'est pas encore certain que cela soit le cas pour des composants décrivant des comportements **fonctionnels**, c'est à dire réalisant une **fonction typée contrôle-commande** d'un processus plus complexe. A titre d'exemple d'utilisation possible de l'héritage, la réalisation d'un modèle générique applicable à la conception de la commande d'un montage d'usinage (Figure I.3) fait apparaître deux **classes fonctionnelles** [ING 87] nommées "Positionner" et "Maintenir" avec une décomposition de "Positionner" en trois classes "Poser", "Plaquer", "Caler". "Positionner" peut être décrit dans SPEX par un **Diagramme Fonctionnel Générique Vide**, les autres par des **Boîtes Fonctionnelles Génériques Vides** ayant des interfaces et une liste de paramètres bien définis. L'instanciation d'un tel modèle et l'association, dans une

application, d'une technologie particulière (et donc d'un comportement particulier) à chaque BFGV pose un problème lorsqu'un même mécanisme supporte plusieurs de ces fonctions. En effet, dans le cas particulier du montage d'usinage présenté en [LHO 89], les fonctions "Maintenir" et "Caler" sont mises en oeuvre par un même mécanisme (Vérin), les fonctions "Poser" et "Plaquer" sont supportées par une même mécanique (Rondelles). Il est donc ici nécessaire de redéfinir les comportements "Poser-Plaquer" et "Caler-Maintenir" en héritant des interfaces et paramètres des fonctions "Poser" et "Plaquer" pour le premier, et des fonctions "Caler" et "Plaquer" pour le second.

La fonction d'héritage de l'interface et des paramètres devraient donc, à terme, être intégrée à l'outil SPEX pour son utilisation possible dans de tels cas de figure.

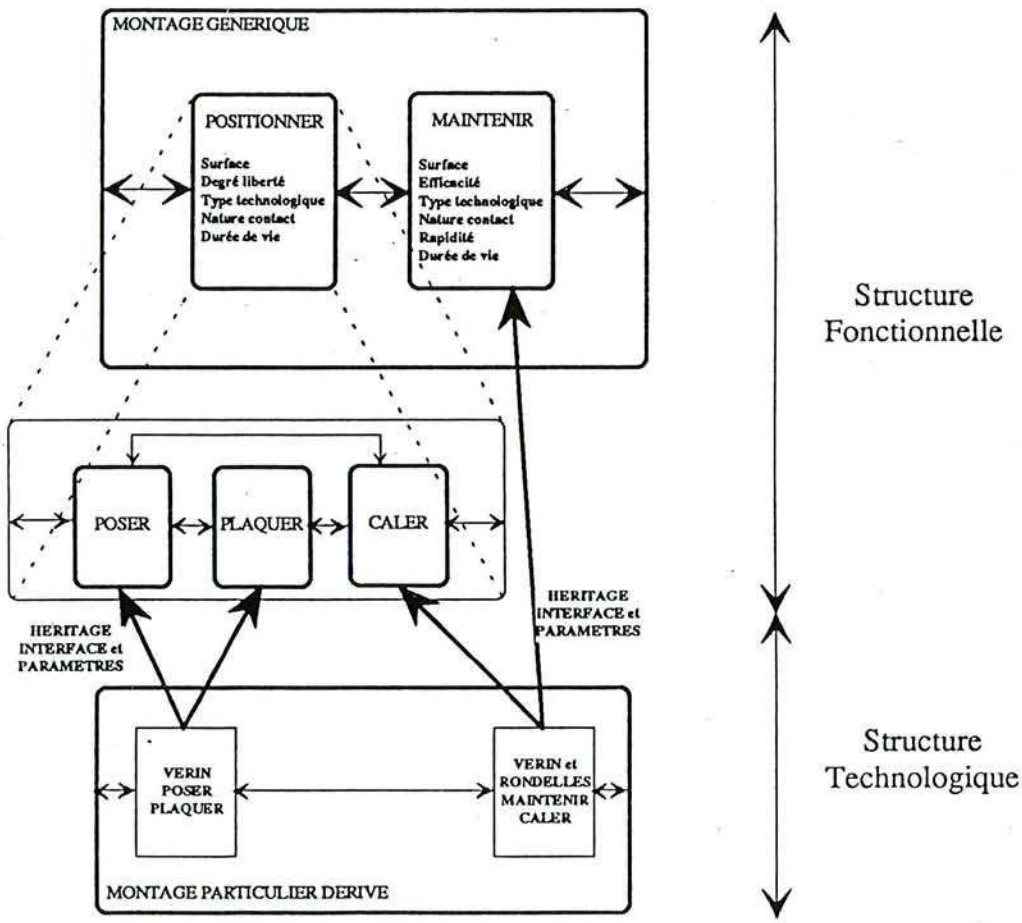


Figure I.3. : Utilisation de l'héritage dans SPEX

Nous avons eu comme objectif d'appliquer ces quelques concepts importants du Génie Logiciel aux outils de l'automatiseur, tout en préservant l'essentiel de sa culture, de son savoir-faire, ...

La mise en oeuvre dans SPEX de la réutilisation, de la simulation, du maquettage (spécification exécutable), du prototypage (conception exécutable) en sont les points principaux.

Les objectifs principaux de SPEX peuvent être résumés de la manière suivante :

- offrir un moyen aux concepteurs ("automatiseurs" et "génie automatiseurs") de construire des comportements réutilisables et de conserver ce savoir-faire,
- permettre la création d'application à partir de ces composants,
- fournir une documentation complète, claire et précise pour les composants et les applications,
- autoriser l'utilisation de l'outil, lors de phases d'analyse, pour valider une organisation fonctionnelle (issue d'une analyse structurée) après avoir associé à chaque fonction un comportement (une "dynamique") de manière à vérifier progressivement la qualité de la spécification (notion de spécification exécutable),
- qualifier la prestation du concepteur par la mise en place d'indicateurs traduisant la complexité de la modélisation.
- ouvrir l'outil aux générateurs de code (utilisation de la description des comportements de commande) et à des outils de validation du code (utilisation de la description des comportements des éléments "opératifs" environnants),

II. FONCTIONNALITES DE L'OUTIL SPEX

1. SPEX : démarches pour la description de comportements

La spécification et le développement d'un outil comme SPEX amène nécessairement à se poser des questions sur son utilisation future pour permettre à un automatiseur quelconque d'analyser et/ou concevoir un automatisme.

Un des objectifs principaux de SPEX est de permettre à un automatiseur de décrire des comportements de fonctions dans des langages qui lui sont familiers et de vérifier la

conformité de ces comportements par rapport à la spécification initiale. L'objet de cette affectation d'un comportement à une fonction, nous ne pouvons pas (et ne devons pas !) en préjuger si nous voulons garantir son indépendance vis-à-vis des méthodes avec lesquelles il sera utilisé. Cependant il est nécessaire de permettre l'utilisation de SPEX à partir de différentes formes de description des fonctions pour lesquelles nous distinguons:

- les fonctions exprimées indépendamment de toute structure,
- les fonctions décrites dans une structure (relations entre fonctions - c'est par exemple le cas pour une fonction décrite dans un modèle SADT/IDEF-0).

Le premier cas n'impose rien pour SPEX si ce n'est qu'il respecte ce pour quoi il est conçu, c'est à dire, offrir les moyens de décrire un comportement pour chaque fonction.
Le second cas impose que SPEX permette aussi d'exploiter les relations entre fonctions, et donc entre comportements associés à chaque fonction, telles que décrites dans la structure fonctionnelle, dans un formalisme éventuellement spécifique (DFD [DEM 78], SADT [ROSS 75].

Pour décrire des comportements associables à des fonctions, plusieurs démarches, non forcément exclusives, sont à distinguer :

- *démarche descriptive* : cette démarche, encore trop courante, consiste à décrire, sans recherche de modularité ni de réutilisabitité, un comportement spécifique ;

- *démarche descendante* : c'est la démarche la plus courante qui consiste à décomposer progressivement les "fonctions" en "sous-fonctions" jusqu'à obtenir des fonctions "élémentaires", ou autrement dit, des comportements que l'on sait aisément décrire ;

- *démarche ascendante* : ce type de démarche cherche à favoriser la réutilisation d'éléments (de comportements) pré-existants ainsi que la construction de nouveaux comportements par combinaison de ces éléments (comportements "génériques", c'est à dire, indépendants de toute application). Elle est mise en oeuvre par les approches "objets" définissant des classes de composants et généralisant la notion de réutilisabilité.

La "meilleure" démarche réside sans aucun doute dans une démarche "mixte" combinaison de "descendant", d'"ascendant", d'analyse et de synthèse et permettant d'appliquer à chaque nouveau problème une étude systématique tout en favorisant au maximum la réutilisation d'un savoir-faire acquis précédemment. Cette approche permet ainsi une **conception incrémentale** de l'application par va-et-vient entre analyse et synthèse jusqu'à l'obtention de la solution désirée ou, au moins, la plus proche des besoins réels.

SPEX fournit, pour supporter ces différentes démarches, une aide à la structuration et à la hiérarchisation par la définition de composants réutilisables : La **Boîte Fonctionnelle**, composant élémentaire dont le corps est décrit dans un formalisme "métier" et le **Diagramme Fonctionnel**, composant complexe défini par un ensemble de boîtes fonctionnelles élémentaires liées les unes aux autres.

Nous aboutissons finalement pour SPEX à une architecture très modulaire qui fait apparaître :

- des éditeurs , supportant des formalismes typiques de l'automatiseur (Grafcet, Schéma à Contacts, Logigrammes, Langage C, Pupitre), permettant la création de Boîtes Fonctionnelles,
- un éditeur d'Applications ou de Diagrammes Fonctionnels,
- un outil général de simulation.

2. SPEX : Du Génie Automatiseur ... à l'Automatiseur

La réutilisation d'un savoir-faire et des compétences de tout à chacun nécessite, de la part du concepteur, une définition exhaustive et sûre de composants génériques réutilisables. Il est donc nécessaire d'établir une **séparation nette** entre le concepteur de composants génériques et l'utilisateur de tels composants pour la réalisation d'une application particulière.

Le **Génie Automatique**, discipline qui doit permettre une automatisation - ou tout du moins une assistance - de l'automatisation, tant d'un point de vue méthodologique que qualitatif [MOR 88], trouve ici son **application** en la personne du **Génie Automatiseur.** Ce dernier doit posséder une connaissance des moyens mis en oeuvre pour l'automatisation d'installations et être capable d'en extraire la **composante générique** par une "superposition" des applications de l'entreprise et une généralisation des comportements spécifiques, seule partie réutilisable pour un ensemble d'applications d'un domaine particulier. Ces composants doivent être de qualité (au sens réaction à des évènements imprévus) et venir ainsi enrichir la base de savoir-faire de l'entreprise ou d'un domaine d'application. Il doit, de plus, définir et mettre en oeuvre des **méthodes et outils supports** pour mettre à la disposition des automatiseurs des mécanismes de manipulation et d'assemblages de ces composants.

L'**automatiseur**, utilisateur de composants génériques, doit les particulariser (ou les adapter) à son application ou éventuellement, se "fabriquer" ses propres composants réutilisables pour la création de son installation. Ces derniers ne pourront être généralisés à l'entreprise ou un un domaine d'application particulier qu'après vérification et validation par le Génie Automatiseur, seul responsable de la généricité inter-applications.

SPEX apporte sa contribution à ces deux métiers par une séparation des mécanismes en fonction du niveau d'abstraction choisi :

- il offre la possibilité au Génie Automatiseur de manipuler des composants "génériques" pour la définition d'autres composants "génériques" venant enrichir la base entreprise ou domaine d'application,
- l'Automatiseur manipule des "instances" de composants génériques qu'il particularise pour son application et peut se définir des composants réutilisables dans son application particulière mais n'étant pas réputés comme génériques pour l'entreprise ou dans le domaine d'application concerné.

Nous pouvons résumer ces différents types d'objets manipulés en fonction du niveau d'abstraction dans la figure I.4 :

	UTILISATION			CREATION	
Génie Automatiseur	Générique Entreprise	Partiel Entreprise Domaine application	Partiel Application	Générique Entreprise	Partiel Entreprise Domaine application
Automatiseur	Générique Entreprise	Partiel Entreprise Domaine application	Partiel Application	Partiel Application	Particulier Application

Figure I.4. : Génie Automatiseur et Automatiseur : 2 métiers

L'objectif que nous nous sommes fixés dans SPEX offre

au **Génie Automatiseur,** pour la définition d'entités génériques :

- une démarche **descendante** en permettant :

. la description progressive d'un "**Diagramme Fonctionnel Générique**" (Figure I.5) par niveaux hiérarchiques structurés composés, de manière non exclusive de :

* "**Diagrammes Fonctionnels Génériques Vides**" (DFGV) et/ou de "**Boîtes Fonctionnelles Génériques Vides**" (BFGV) décrivant une *structure d'accueil* dont seule l'interface avec l'environnement a été définie (interface générique). Cette approche est à relier à la définition d'une classe au sens "approche objet" pour laquelle il est nécessaire de définir l'interface avant de décrire le comportement [MEY 88].

* "**Diagrammes Fonctionnels Génériques**" (DFG) et/ou de "**Boîtes Fonctionnelles Génériques**" (BFG) décrivant des *comportements élémentaires* archivés en bibliothèque ;

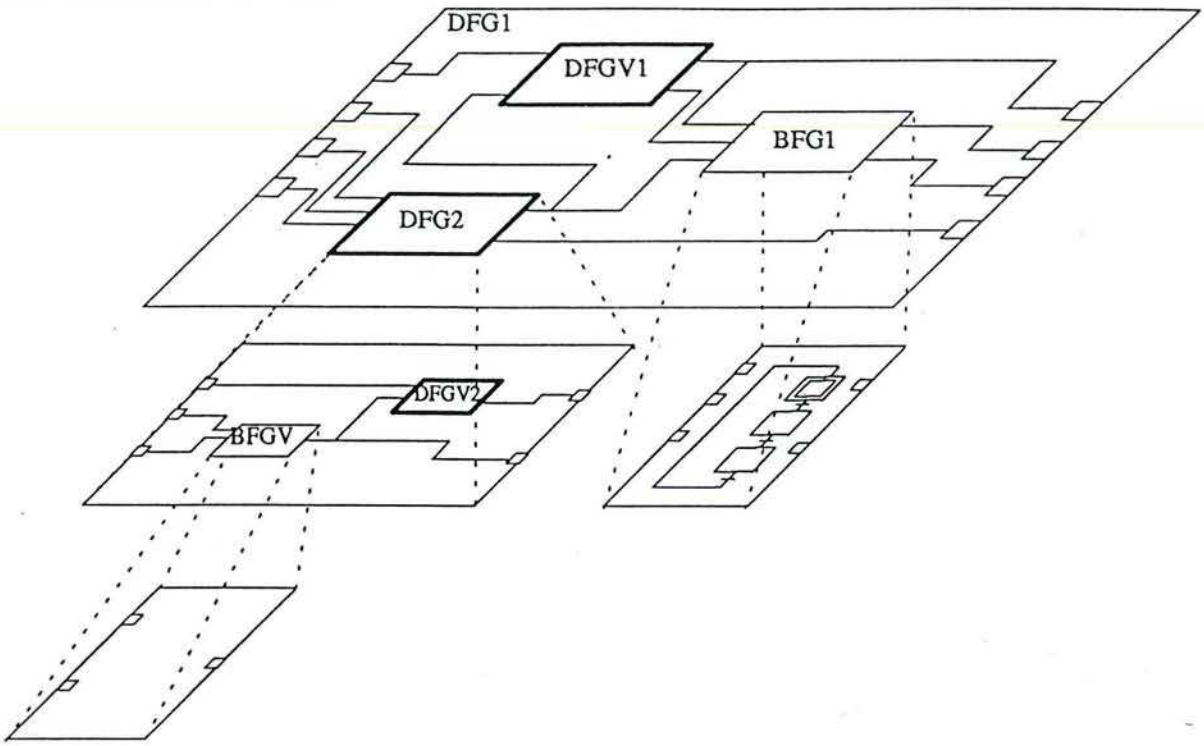


Figure I.5. : Décomposition hiérachique d'un Diagramme Fonctionnel Générique

- une démarche **ascendante** en permettant :

  . La description de "**Boîtes Fonctionnelles Génériques**" (BFG) élémentaires, indépendantes de l'application qui les utilisera,

  . la définition de "**Diagrammmes Fonctionnels Génériques**" par assemblage de BFG et/ou DFG représentant un comportement réutilisable (Figure I.6)

- une démarche **mixte** pour la description d'un Diagramme Fonctionnel Générique par l'utilisation conjointe des deux approches "*descendante*" et "*ascendante*" manipulant des composants **génériques** (Figure I.6);

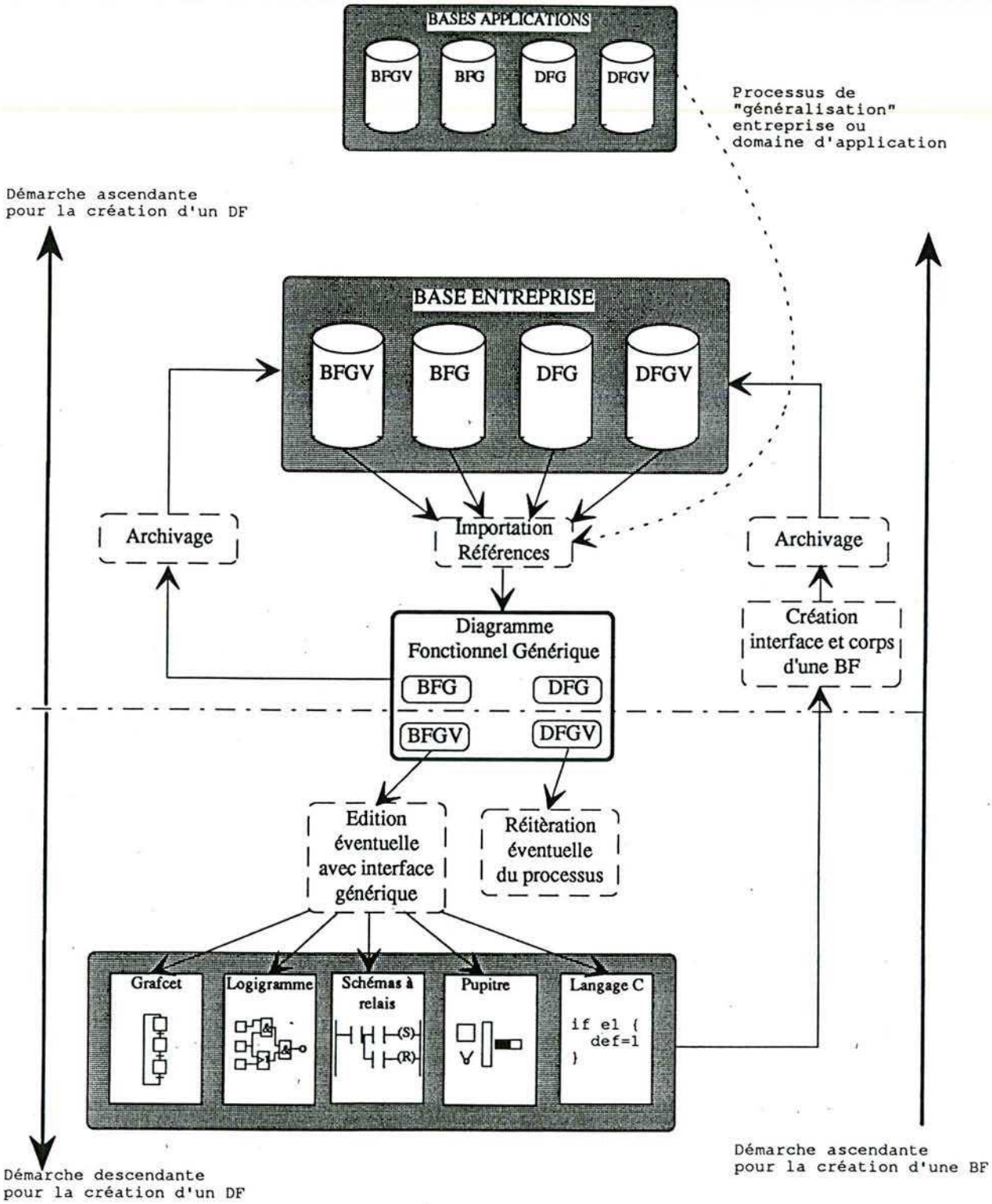


Figure I.6. : Processus de création d'un DFG par le Génie Automatiseur

à l'**Automatiseur**, pour la définition d'applications particulières:

- une démarche **descendante** en permettant :

  . la description progressive d'une Application par niveaux hiérarchiques structurés composés d'Exemplaires (ou instances) de "**Diagrammes Fonctionnels Génériques Vides**" (DFVE) ou de "**Boîtes Fonctionnelles Génériques Vides**" (BFVE) décrivant des comportements élémentaires dont seule l'interface avec l'environnement a été définie (interface particulière à l'application étudiée),

  . la définition de "**Boîtes Fonctionnelles**" typées application à partir de l'interface particulière des BFVE définies dans l'application, la description d'un comportement élémentaire puis l'application d'un processus de "généralisation", dans le monde de l'application représentant un **composant réutilisable** pour l'application;

- une démarche **ascendante** en permettant :

  . La description de "**Boîtes Fonctionnelles Génériques**" (BFG), réutilisables dans le cadre de l'application mais ne constituant pas, telles quelles, des éléments du patrimoine entreprise(Figure I.7),

  . la définition de "**Diagrammmes Fonctionnels**" typés application par assemblage de BFE et/ou DFE et l'application d'un processus de "généralisation" , cet assemblage "généralisé" représentant une structure d'accueil réutilisable **mais non générique** au sens de l'entreprise (Figure I.7);

- une démarche mixte pour la définition de **Diagrammes Fonctionnels** typés application par l'utilisation conjointe des deux approches "descendante" et "ascendante" manipulant des composants exemplaires (Figure I.7).

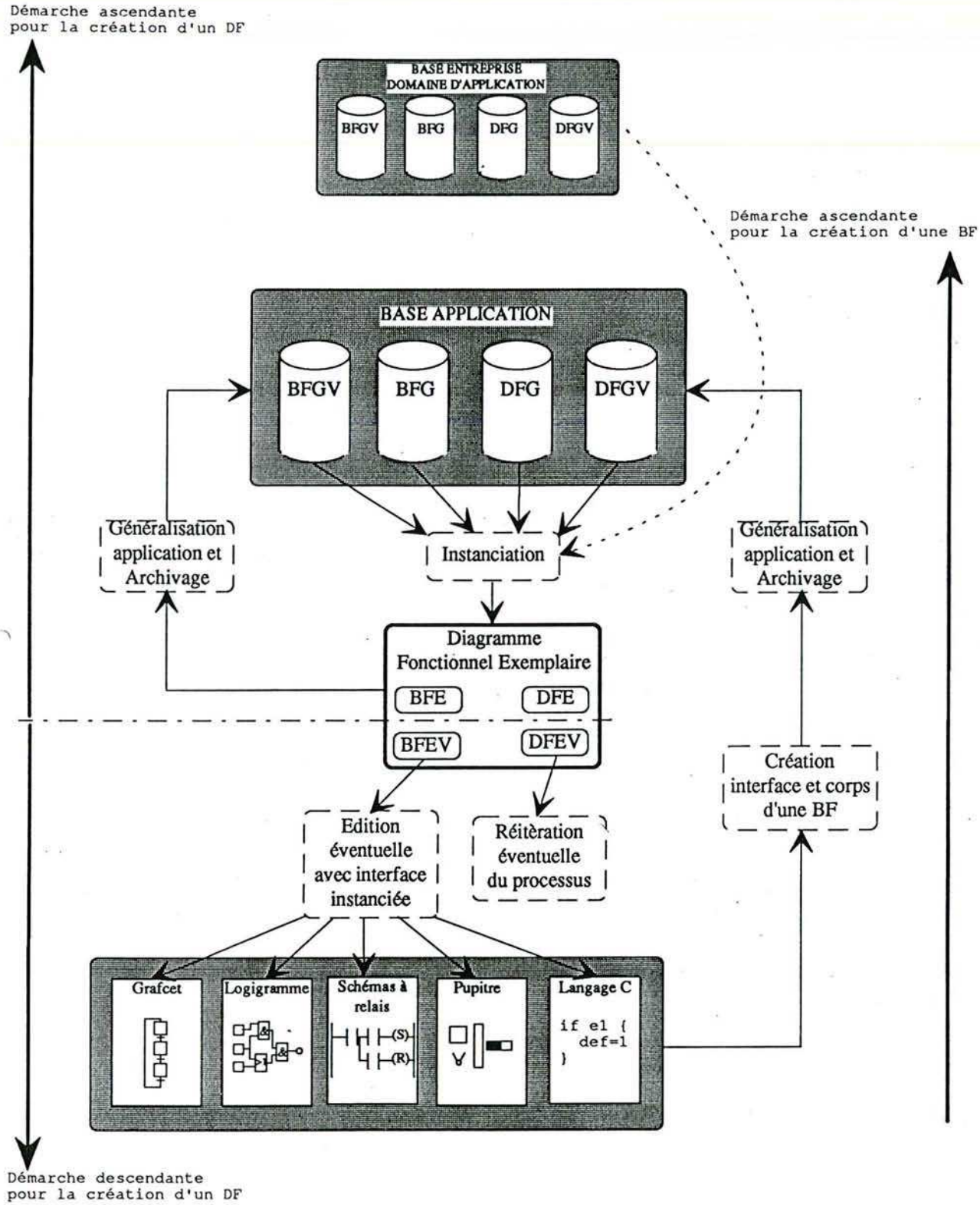


Figure I.7. : Processus de création d'un DFG par l'Automatiseur

Pour résumer, SPEX permet la manipulation d'objets **particularisés** ou **génériques**, **"vides"** ou **complètement définis** en fonction de l'utilisateur et de la démarche méthodologique choisie (Figure I.8)

	Ascendante	**Descendante**
Génie Automatiseur	DFG et/ou BFG DFE et/ou BFE	DFGV et/ou BFGV DFG et/ou BFG DFE et/ou BFE
Automatiseur	DFG et/ou BFG typés application DFE et/ou BFE	DFVE et/ou BFVE DFG et/ou BFG typés application

Figure I.8. : Les objets manipulés

Nous n'avons cependant pas contraint l'utilisateur à respecter telle ou telle démarche, et donc, la démarche descriptive peut être utilisée sur SPEX (pour les automatiseurs qui désireront continuer à mal travailler !) en considérant, par exemple et à l'extrême, qu'une application est décrite par un seul **Diagramme Fonctionnel** constitué d'une seule **Boîte Fonctionnelle.**

3. Les Boîtes Fonctionnelles

3.1. Définition

La **Boîte Fonctionnelle** (BF) est la notion qui correspond le mieux, en Automatique, à la notion de **Composant Logiciel Réutilisable** [COX 86] ou de macro-constituant [FRA 87]. Il s'agit d'une unité de "comportement", dont l'objectif est d'élaborer une ou plusieurs valeurs de sorties en fonction des valeurs de ses entrées, variables locales et/ou paramètres.

3.2. Le processus de création

3.2.1. Les éditeurs typiques de l'automatiseur

Le **corps** d'une Boîte Fonctionnelle peut être décrit à l'aide d'éditeurs utilisant les formalismes propres au domaine de l'automatisation tels que Grafcet selon la norme AFNOR NFC-03-190 [NFC 82], les propositions d'extensions définies par l'AFCET [AFC 87] et bientôt la norme internationale [CEI 88], Schémas à relais, Logigrammes, "Pupitres" ainsi que d'un éditeur permettant la description du comportement en Langage C. Ces éditeurs ont tous la même interface homme/machine pour permettre une utilisation simplifiée et uniforme. Ils sont capables d'intégrer des exemplaires de BF et/ou DF dont le corps aurait été décrit précédemment dans n'importe quel formalisme. Il est à noter que les éditeurs Grafcet et Langage C, du fait de leur formalisme, ne permettent pas une intégration explicite (expression des liens) d'un exemplaire de BF/DF tel quel mais permettent son appel par l'intermédiaire d'une action associée à une de ses étapes pour le Grafcet (Figure I.9), ou d'une fonction C pour le Langage C selon la syntaxe :

NomBFDF (liste variables ou valeurs entrées, liste variables sorties, liste variable ou valeurs paramètres).

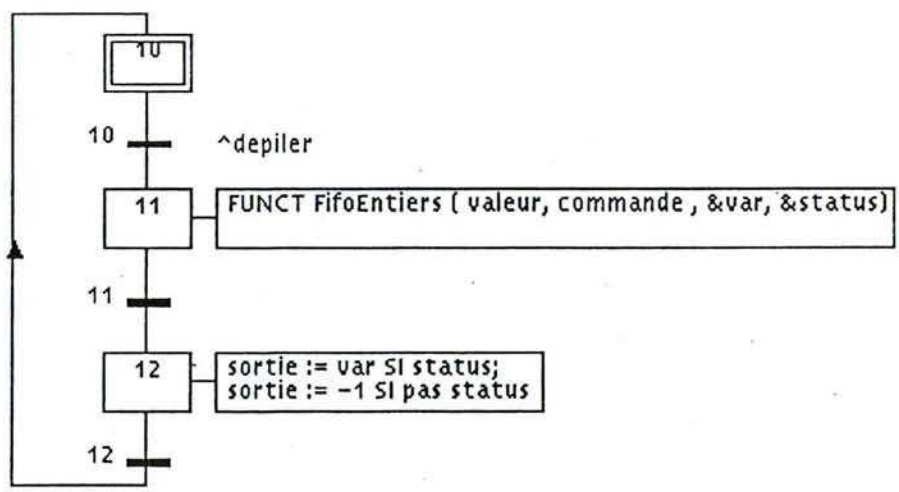


Figure I.9.a : Appel d'exécution d'une BFE dans un Grafcet

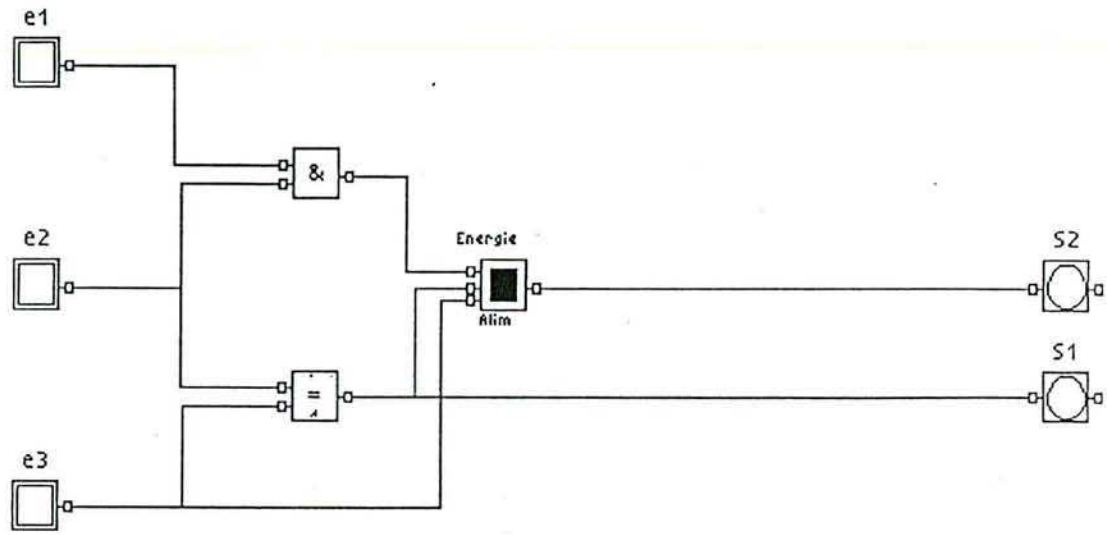


Figure I.9b : Intégration d'une BFE dans un Logigramme

3.2.2. L'éditeur Grafcet

Il s'agit de créer un **prototype** de modèle de comportement dont le corps (exécutable) est décrit en utilisant la notation Grafcet [NFC 82]. Ce corps est considéré dans SPEX comme étant un **Graphe Partiel**, composé d'un ou plusieurs **Graphes Connexes**, de deux listes de combinatoires d'entrée et de sortie, conformément aux recommandations de l'AFCET [AFC 87].

Les éléments manipulés par l'éditeur sont :

- l'étape, qui peut être normale, initiale ou macro, source ou puits
- la transition, normale, source ou puits
- les arcs, munis éventuellement de renvois.

Notons que toutes les caractéristiques de la norme Grafcet sont respectées dans SPEX avec, en plus, quelques extensions qu'il nous semblait nécessaire d'ajouter pour garantir la praticabilité de l'outil par des puristes, mais aussi, par des non-puristes (Forçages entre graphes connexes, action sur macro-étapes, ...)

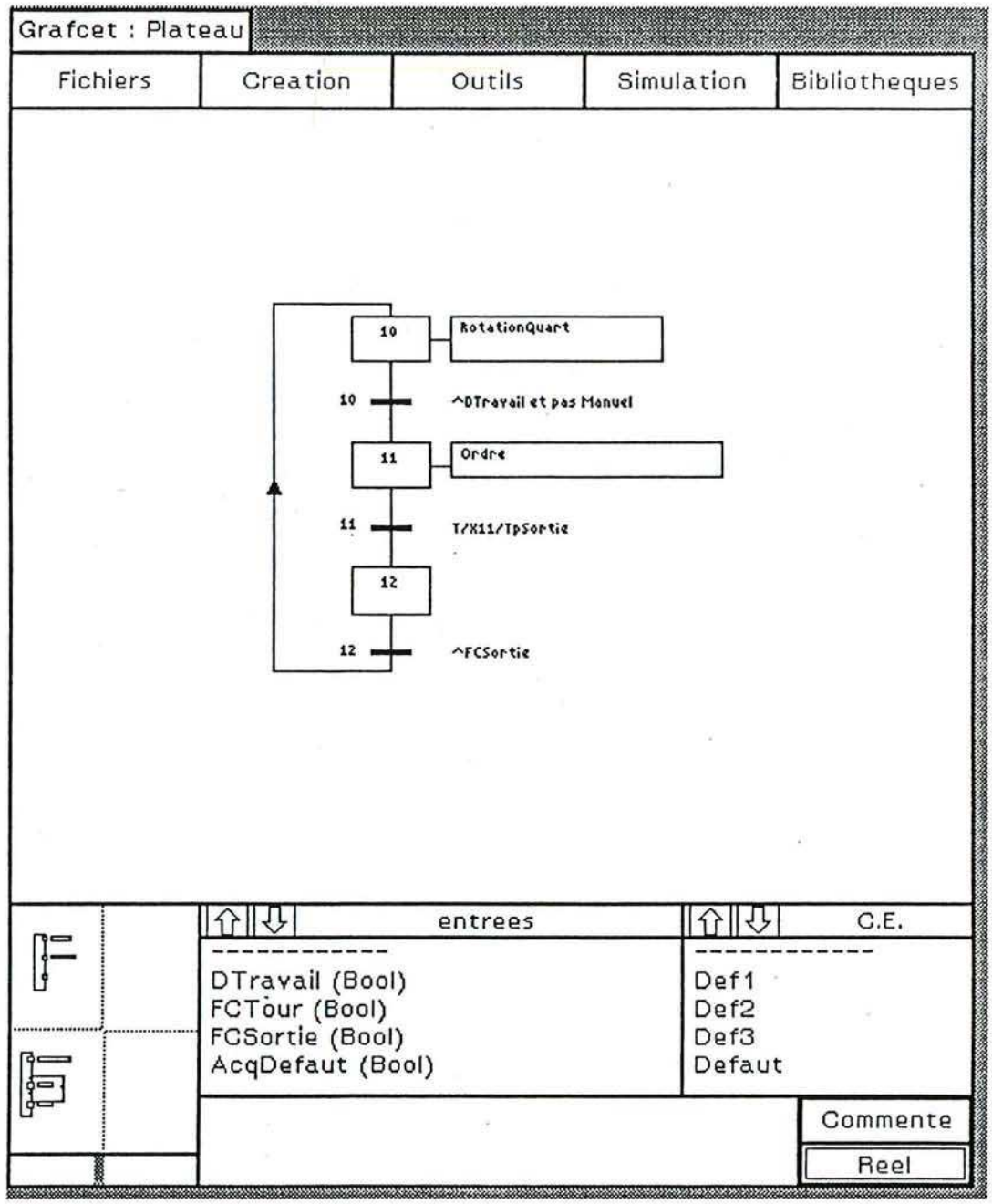


Figure I.10. : L'éditeur Grafcet

3.2.3. Les éditeurs Schémas à relais et Logigrammes

Les éditeurs de schémas à relais et logigrammes permettent de décrire des comportements combinatoires en utilisant les langages graphiques correspondants. L'utilisation de ces éditeurs est comparable à l'éditeur grafcet et met à la disposition du concepteur des symboles propres à l'édition de ces schémas.

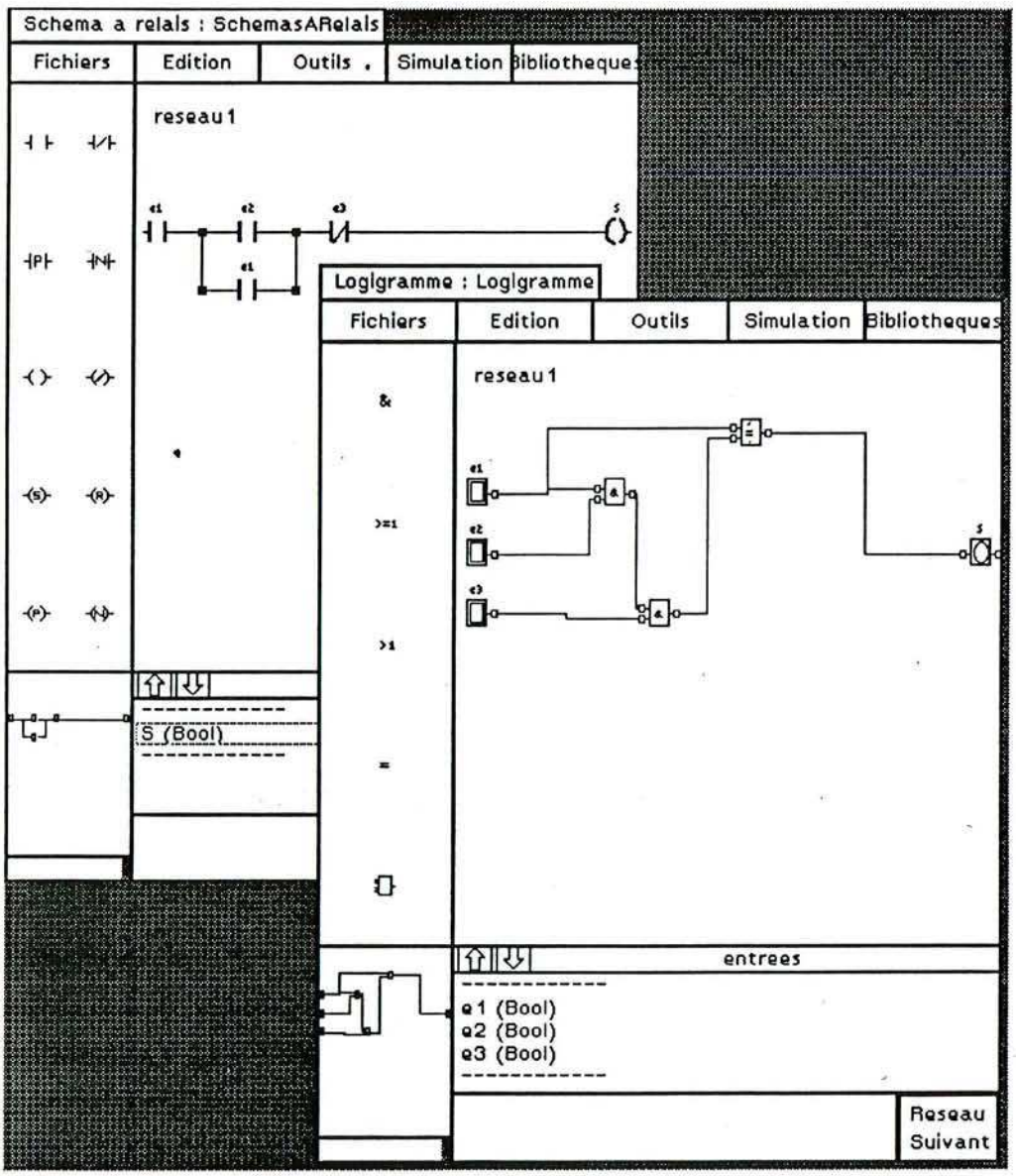


Figure I.11. : Les éditeurs Schémas à relais et Logigrammes

Il est à noter que ces éditeurs fournissent une aide à la vérification des schémas par une **reconstitution textuelle** des équations logiques associées à chaque élément en fonction du réseau combinatoire y aboutissant (Figure I.12).

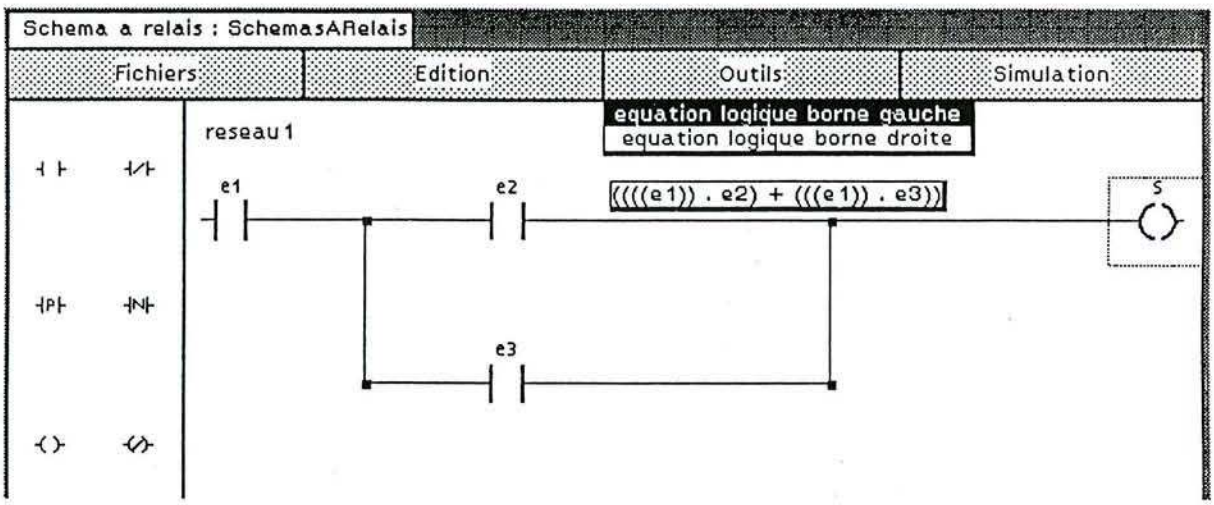


Figure I.12. : Reconstitution textuelle d'une équation logique

3.2.4. L'éditeur Langage C

La description d'un modèle de comportement en Langage C permet de modéliser des fonctions de calcul ou algorithmiques difficilement exprimables avec les autres notations. L'utilisateur doit ici décrire une fonction C spécifiant le mode de calcul des sorties en fonctions des entrées, variables locales et paramètres du modèle. Comme pour les autres éditeurs, il doit être au préalable renseigné sur les noms, types et éventuellement valeurs initiales des différentes variables puis il permet la description du corps en respectant la syntaxe C avec la possibilité d'utiliser toutes fonctions C standards. Il est tout de même conseillé de ne pas

utiliser des fonctions bloquantes dans l'environnement (entrées/sorties standards du C, ...), ni d'utiliser les différents types de boucles qui risquent d'allonger le temps de cycle si ce n'est bloquer la simulation de l'ensemble de l'application (simulation totalement synchrone). Ces restrictions correspondent à celles préconisées dans la chaîne de conception assistée PIASTRE [CAZ 83].

Une vérification syntaxique de code est réalisée à la demande (fonction "vérifier") ou systématiquement avant sauvegarde ou simulation.

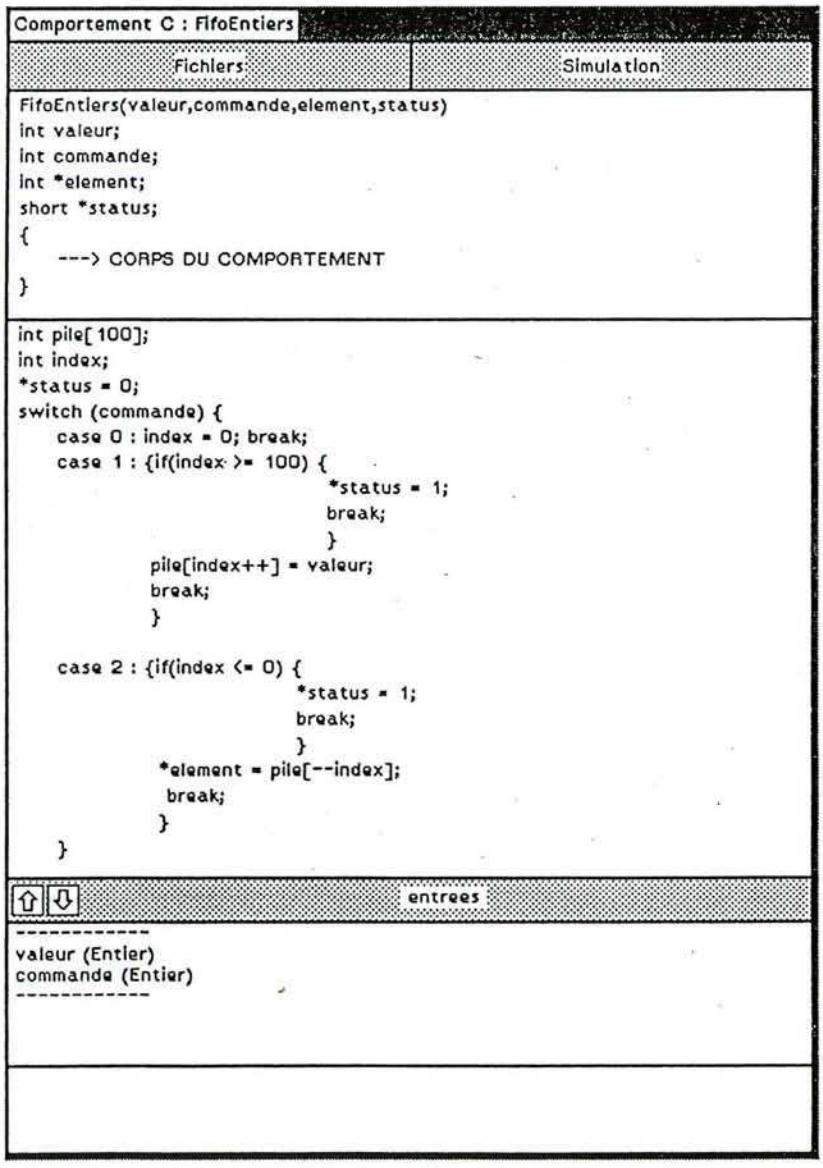


Figure I.13. : L'éditeur Langage C

3.2.5. L'éditeur de Pupitres de commande

L'opérateur, dans un automatisme, peut être considéré soit comme composante décisionnelle de la Partie Commande soit, comme composante, à comportement indéterministe, de l'environnement.

Le choix, dans l'outil SPEX, étant de ne pas préjuger du typage des fonctions, il est nécessaire d'offrir les moyens de décrire sous forme de BF des "comportements" d'**opérateurs de conduite.**

Cette spécification pour l'outil SPEX est justifiée, en particulier, au Chapitre IV, § 1.3.1.1.2 (Grilles de Contrôle-Commande) où le pupitre de commande de l'installation est, nécessairement, composé de "pupitres élémentaires" définissant, par exemple, les modes de marche et commandes opérateur, locaux à un sous-ensemble de l'installation. Il est donc nécessaire de pouvoir modéliser, au sein de l'application, l'ensemble des pupitres élémentaires dépendant du niveau de décomposition en cours.

Ce pupitre est décrit par un ensemble d'afficheurs/générateurs permettant la description des commandes et visualisations de la sous-installation. Ces informations sont définies par les entrées et sorties de façon identiques aux autres éditeurs. Le comportement d'un pupitre peut être, soit laissé à l'initiative de l'opérateur en cours de simulation, soit décrit par un scénario (cf § 2.6.2) "simulant" le comportement opérateur de manière éventuellement non déterministe.

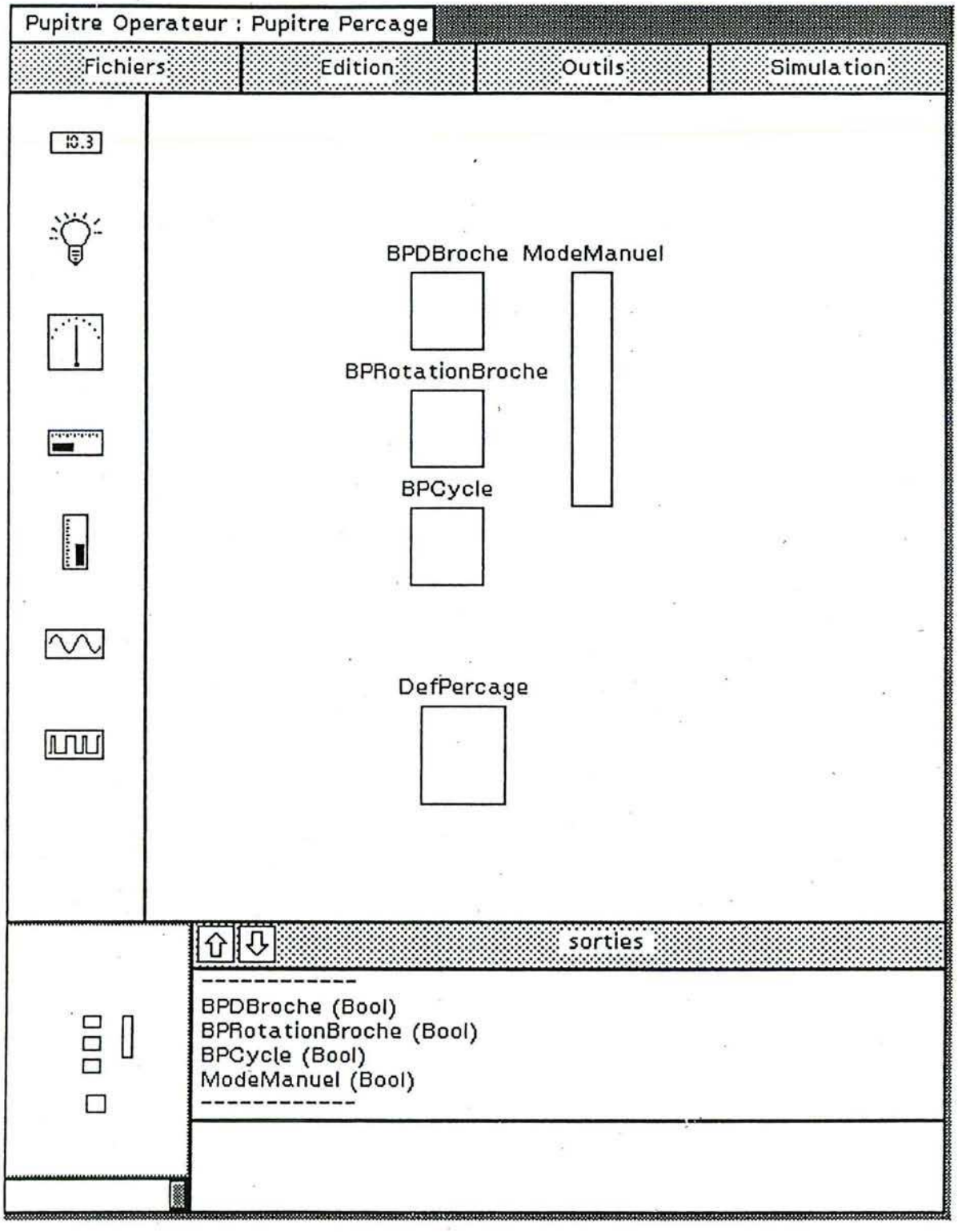


Figure I.14. : L'éditeur Pupitre de Commande

3.3. Le processus d'instanciation de Boîtes Fonctionnelles

Plusieurs **instances ou exemplaires** (BFE) peuvent être créés à partir d'une même Boîte Fonctionnelle Générique, chacun possédant alors un identificateur (son nom) ainsi que son propre jeu de variables (Entrées, Sorties, Variables Locales, Paramètres) (Figure I.15). Seules les variables d'entrée et de sortie constituent l'interface du modèle de comportement, pour mieux masquer les informations inutiles et aboutir ainsi à une meilleure réutilisabilité (par un couplage plus faible). Les paramètres du modèle (ou "variables d'instance") permettent de particulariser (configurer) des exemplaires d'un même modèle et ainsi leur apporter une forte cohérence interne par l'utilisation de leur propre structure de donnée.

Le processus d'instanciation d'une BF correspond exactement à celui défini dans le Génie Logiciel et mis en oeuvre par les approches "objets". Chaque instance duplique les variables de sa classe et réutilise son comportement.

La création d'un exemplaire de Boîte Fonctionnelle Vide est un processus éventuellement utilisable lors de la définition d'une application. L'exemplaire ne définira alors qu'une interface particulière pour l'application concernée qui sera, par la suite, généralisée lors de la description du corps.

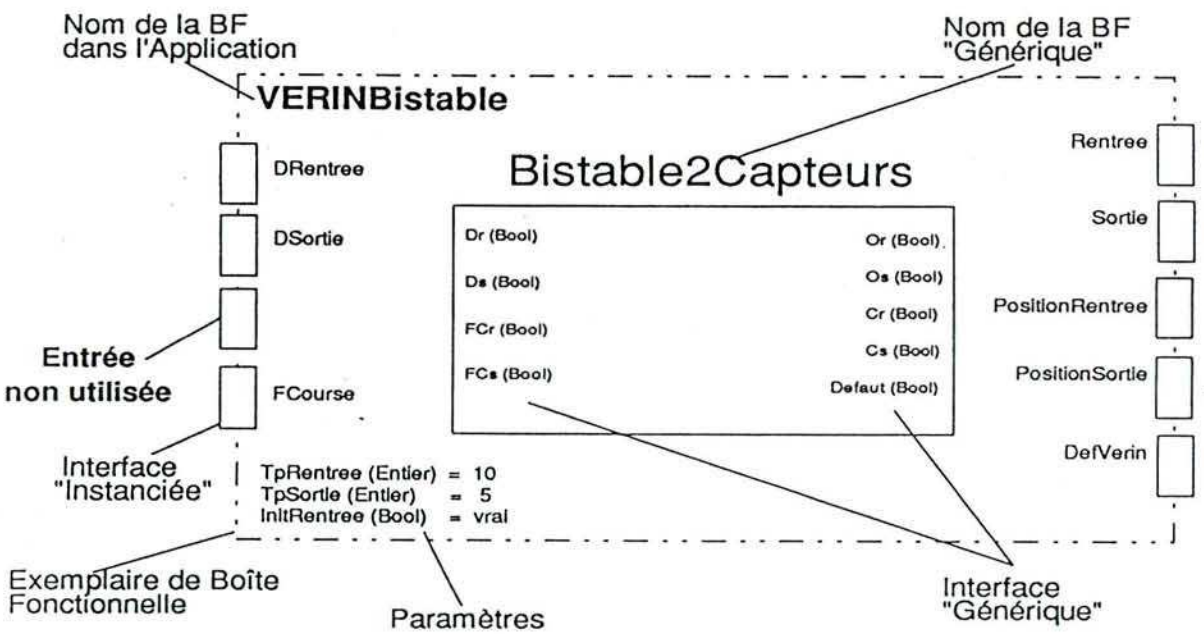


Figure I.15. : Exemplaire de Boîte Fonctionnelle (BFE)

3.4. Le processus de généralisation

Le processus de **généralisation** correspond à la définition d'une **interface** "réputée" **générique** constituée des entrées et sorties d'un corps de boîte fonctionnelle ou déduite de l'interface d'une BFV (générique ou exemplaire).

Une BF Générique (BFG) est donc un composant constitué de :

- une interface, comprenant la liste des variables d'entrée et de sortie,
- un ensemble de variables locales, rémanentes,
- un ensemble de paramètres (à lecture seule),
- un corps exécutable, dont l'objectif est de calculer les sorties en fonction des valeurs des entrées, des variables locales et des paramètres.

Le processus de **généralisation** correspond, au sens "approche objet" à la création d'une classe à partir de sa **spécification formelle** [BOO 88] (interface et comportement).

Une **BF Générique Vide** (BFGV) ne comprend qu'une interface générique, son corps sera décrit à partir de son interface.

L'utilisation d'un exemplaire de BFGV définit automatiquement une **interface générique** en terme de nombre d'entrées et nombre de sorties. Ces variables sont nommées par défaut et ne sont typées que lors de leur première utilisation dans une application. L'édition de la BFGV correspondante importe cette interface générique que l'on peut alors renommer. Les noms génériques des variables d'entrées et de sorties n'étant pas explicites quant à leur "fonction", il est possible de visualiser la correspondance entre leur nom générique et leur nom instancié dans une application particulière. Ceci permet de retrouver l'aspect "utilisation" de ces variables pour ensuite les renommer plus précisément (Figure I.16).

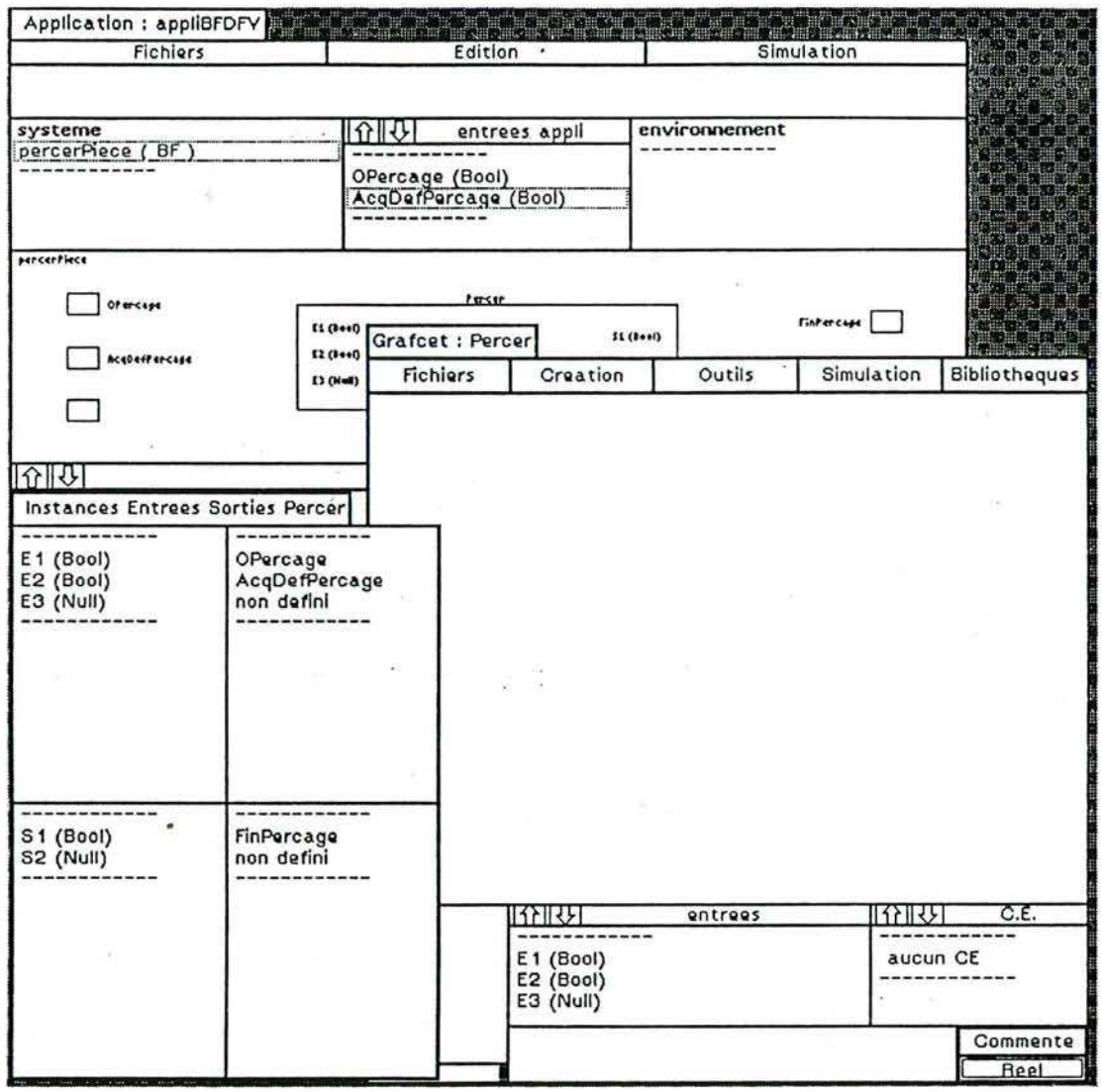


Figure I.16. : Processus de création d'une Boîte Fonctionnelle Générique Vide

4. Le Diagramme Fonctionnel

4.1. Définition

Un **Diagramme Fonctionnel** est constitué d'un ensemble de composants inter-reliés par leur interface et décrivant un comportement complexe. Cet assemblage est archivé en bibliothèque pour être ensuite réutilisé au même titre qu'un modèle décrit dans un langage de base du domaine de l'automatisation. Il possède son interface d'entrée et de sortie et des paramètres de configuration, réunion des paramètres des éléments qui composent sa structure et de la liste des entrées non utilisées (non connectées) de ces mêmes éléments.

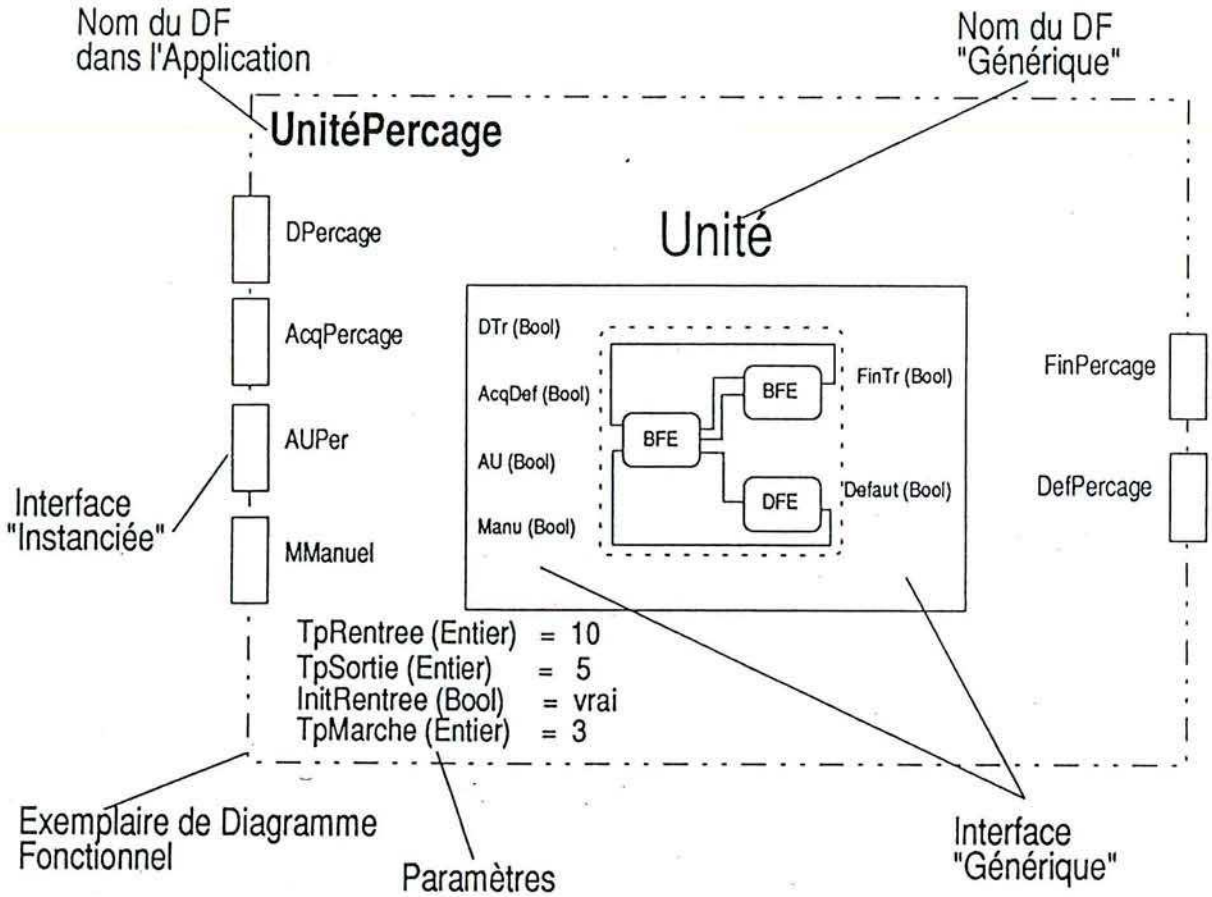
4.2. Le processus de création

4.2.1. Les démarches méthodologiques

1. Démarche ascendante pour la description de comportements

Point de vue de l'automatiseur

Un **Diagramme Fonctionnel** est décrit par l'assemblage d'exemplaires de BFG et/ou d'autres DFG (Figure I.17). Cet assemblage d'exemplaires de comportements permet la description d'un comportement plus complexe. Ce comportement possède son interface d'entrée et de sortie déduite des éléments qui le composent. Ses paramètres de configuration sont la réunion des paramètres des exemplaires qui compose sa structure et la liste des entrées non utilisées (non connectées) de ces BFE/DFE. L'automatiseur n'étant pas habilité à "**standardiser**" ce DF pour d'autres applications, ce composant est donc *typé pour son application* et ne vient pas enrichir la base de savoir-faire de l'entreprise.



Point de vue du Génie Automatiseur

Le **Génie Automatiseur** a pour tâche la description d'un **Diagramme Fonctionnel Générique** (Figure I.6) utilisable par toutes applications. Pour ce faire, il assemble des BFG et/ou DFG en relation par leurs interfaces génériques. Le comportement de l'ensemble est ainsi obtenu par la réunion des comportements des éléments qui le composent et, par les interactions induites par les relations décrites entre ces éléments.

2. Démarche descendante pour la description fonctionnelle d'une installation

Point de vue de l'automatiseur

Le concepteur **automatiseur** peut décrire son installation par une décomposition descendante hiérarchique des fonctions de son système. En effet, il pourra créer des **exemplaires particuliers** de Diagrammes Fonctionnels Vides (DFEV) et/ou Boîtes Fonctionnelles Vides (BFEV), boîtes dont le corps est vide et dont l'interface est parfaitement définie. Par une démarche proche de SADT/IDEF0, il peut ensuite décrire l'intérieur des DFEV soit par d'autres DFEV et/ou BFEV, soit par des exemplaires de BF et/ou DF dont les entrées et sorties seront en relation avec l'interface définie au niveau supérieur (Figure I.7). Il est à noter que l'utilisation d'exemplaires de Boîtes Fonctionnelles Vides nécessite, par la suite, une approche **descendante** pour la description de leur corps et de l'interface déduite de leur "instanciation" dans l'application.

Cette démarche de conception descendante, si la décomposition s'arrête au DFEV permet au concepteur de spécifier l'ensemble des relations entre comportements fonctionnels sans connaître , a priori, les moyens mis en oeuvre pour réaliser la fonction désirée. Cette assemblage de boîte "vides" permet d'offrir une structure d'accueil générique décrivant le "**quoi**" de l'analyse et laissant toutes libertés (dans les limites de l'interface) à l'utilisateur de ce comportement pour expliciter le "**comment**" en associant tel ou tel mécanisme typé application.

Point de vue du Génie Automatiseur

La démarche de modélisation d'un comportement générique par un **Diagramme Fonctionnel Générique** suit exactement le processus de construction pour l'automatiseur. La différence réside dans la nature des objets manipulés, dans le sens où ce sont tous des composants génériques et non des exemplaires particularisés (Figure I.5). Le diagramme fonctionnel ainsi créé offre une structure d'accueil générique réutilisable quelle que soit l'application concernée pour un domaine particulier (Figure I.18).

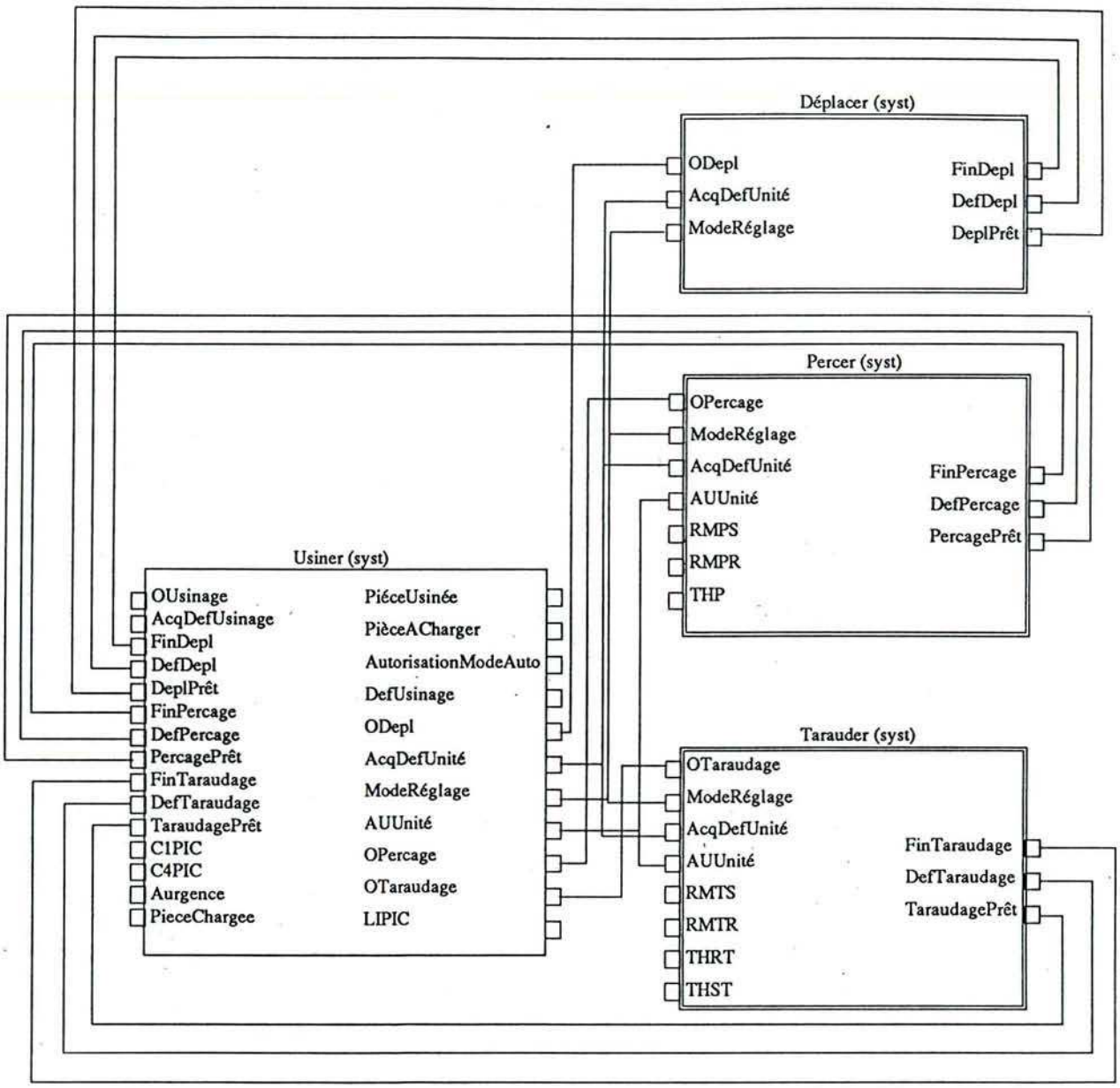


Figure I.18. : Machine Transfert Générique

4.2.2. L'éditeur de Diagrammes Fonctionnels et d'Application

1. Les Vues de l'application

L'application

Nous appelons **Application** l'ensemble d'exemplaires de BF et/ou DF liés les uns aux autres, définissant complètement le comportement d'un automatisme particulier. En ce sens, une application est un **prototype** de Diagramme Fonctionnel (ou exemplaire particulier).

La différence entre Application et DF est **subjective** et réside principalement dans les critères d'arrêt de la décomposition qui induisent un caractère **supposé unitaire** de l'Application à un moment donné de la conception par rapport à la **réutilisabilité** d'un DF.

Pour imager ce propos par une analogie avec le domaine de l'électronique, un DFG composé uniquement de DFGV est le plan électronique,un DFEV est un typon, un DFE est une carte électronique, une BFGV est un plan de cablage de circuit, une BFEV est un support, une BFG est un type de composant, une BFE est un composant particulier.

Dualité Système et Environnement :

Ces différents composants peuvent être classés, selon leur objectif, comme :

- composant **système** qui est le principal objet de l'étude (partie commande),
- composant **environnement** qui modélise un contexte du système (partie opérative et/ou autres systèmes environnants).

Les relations induites par cette séparation de "rôle" amènent, lors de la création d'un DF ou d'une application, à la définition d'un classement automatique des variables d'interconnexion entre composants (entrées application, sorties application, variables systèmes, environnements et mixtes).

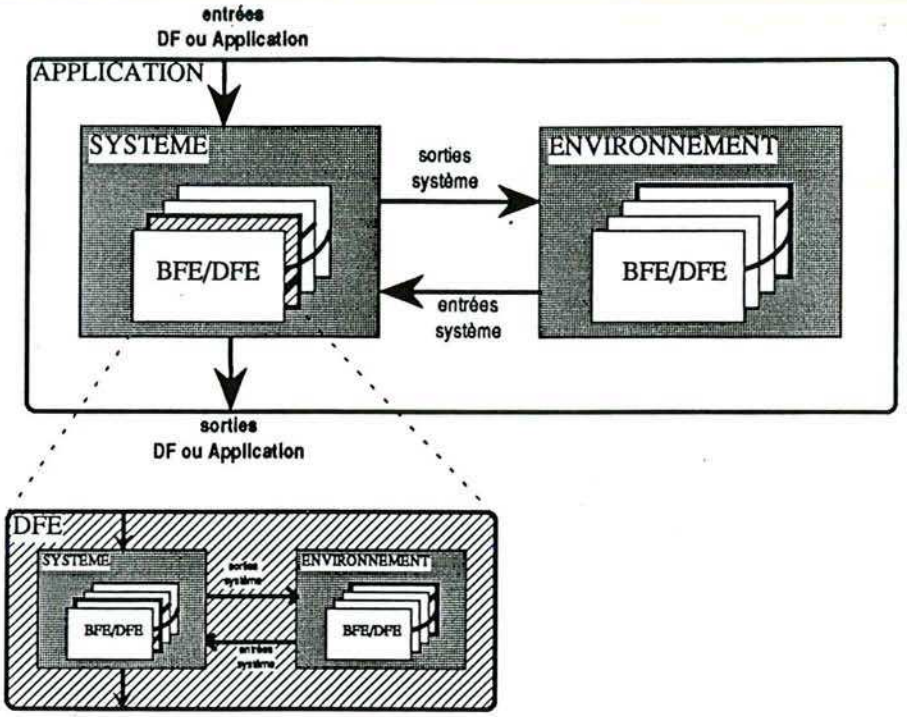


Figure I.19. : Synoptique d'une Application SPEX

L'objectif des composants "environnement" est de permettre la validation de la description du système par une simulation de type "boucle fermée" : la notion d'environnement est ajoutée afin de minimiser les interventions de l'opérateur en simulation en effectuant un bouclage entre sorties et entrées du système fonctionnellement (ou technologiquement, ...) liées. Cette notion est associée aux principes de **Modélisation de Comportement de la Partie Opérative**, introduite par divers travaux [ISM 83], [ADE 84], [LHO 85], [COR 85], [PRU 86], [DEF 86a], [ALA 86] qui ont mis en évidence la généricité de ces comportements. Il est donc intéressant de créer des **Eléments de Partie Opérative** (EPO) [COR 89] modélisant les technologies d'actionnement et les stocker pour une réutilisation ultérieure sous la forme de BF à vocation "Environnement".

Utiliser de tels modèles apporte deux avantages essentiels :

- la simulation est simplifiée, l'opérateur de test ayant moins d'entrées/sorties à gérer,
- la partie opérative (ou tout au moins, la partie animation) doit être connue et formalisée, ce qui contribue à une meilleure connaissance de l'application. Cela

permettra, de plus, de récupérer cette modélisation pour configurer des outils de tests pour recette hors site des logiciels de commande tels qu'ADELAIDE [COR 89] ou le nouveau produit en cours de développement, MAXSIM, construit sur la base de l'outil SPEX [MAX 91]. SPEX, dans sa version prototypée écrite dans le langage Smalltalk80, peut être considéré, à la limite, comme une **structure d'accueil** pour la conception d'autres outils particuliers basés sur les mêmes principes.

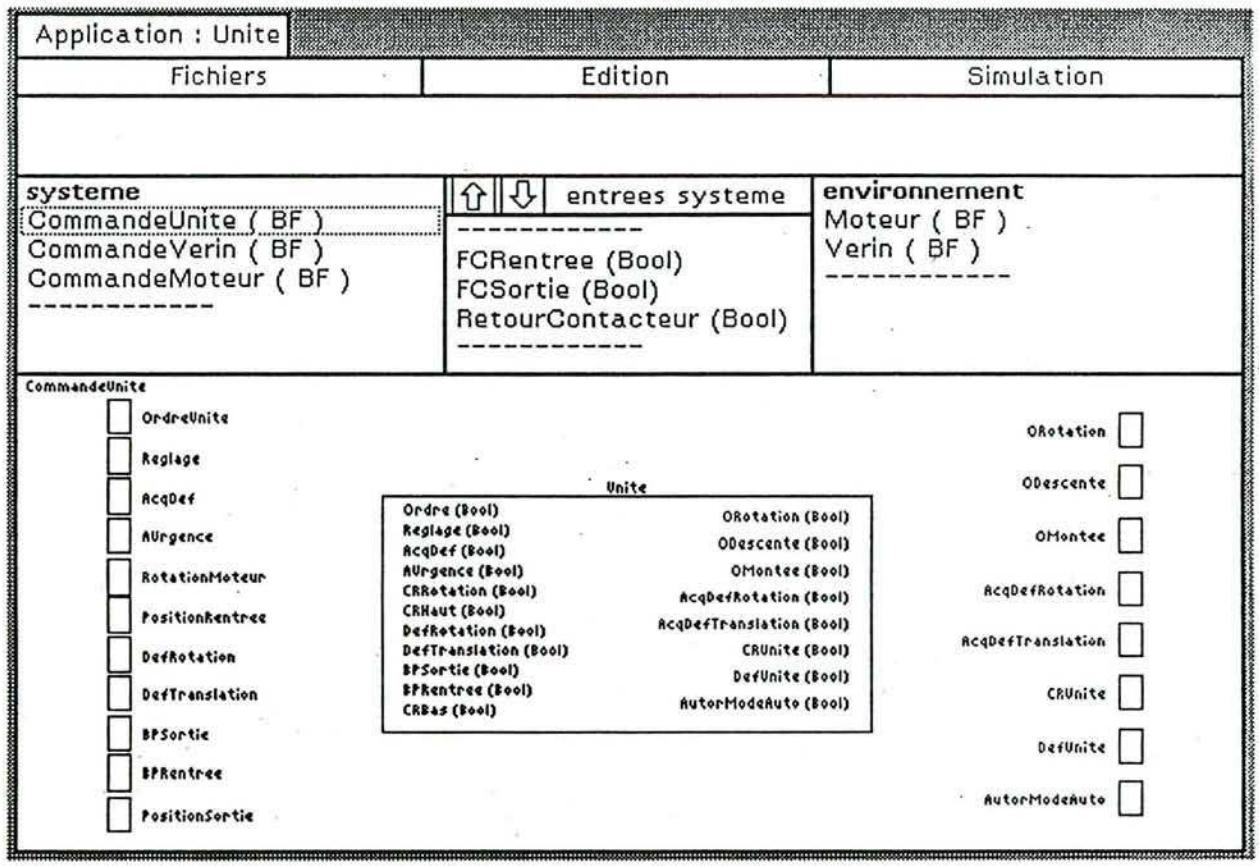


Figure I.20. : L'éditeur de Diagramme Fonctionnel

4.3. Le processus d'instanciation

L'**instanciation** d'un **Diagramme Fonctionnel Générique** instancie automatiquement toutes les Boîtes Fonctionnelles Génériques qui composent sa structure en leur donnant un identificateur par défaut. Il est possible de renommer

tout ou partie des exemplaires de BF ainsi créés pour leur adaptation à l'application particulière. L'ensemble des paramètres du DF ont une valeur par défaut mais peuvent être éventuellement valués différemment pour configurer les éléments qui constituent le DFE. L'instanciation d'un **Diagramme Fonctionnel Vide** offre une *structure d'accueil* définie par son interface et peut éventuellement être complétée par la description de son corps totalement dépendant de l'application qui l'utilise.

Le processus d'instanciation d'un **Diagramme Fonctionnel** peut être identifié, en correspondance avec les "approches objets", à l'instanciation d'une classe **agrégeant** d'autres objets [GOL 83].

4.4. Le processus de généralisation

Le processus de **généralisation** d'un **Diagramme Fonctionnel** permet de l'archiver en bibliothèque pour être ensuite réutilisé au même titre qu'un modèle décrit dans un langage de base du domaine de l'automatique. Le Génie Automatiseur décrit directement un Diagramme Fonctionnel Générique alors que l'Automatiseur décrit un prototype de Diagramme Fonctionnel dans le sens où il manipule déjà des instances d'autres éléments. Dans ce dernier cas, la généralisation consiste à définir une **interface générique** à partir des relations des composants avec le monde environnant et à **reconstituer l'architecture de composants génériques** déduite des exemplaires utilisés et de leurs relations mutuelles. Ce processus aboutit à un Diagramme Fonctionnel Générique, composé d'éléments génériques, identique à celui que peut créer le Génie Automatiseur mais ne prenant pas place dans la base de l'entreprise.

5. Calcul de la complexité d'un comportement

Le problème du concepteur, pour atteindre le niveau de qualité requis sur un facteur donné, est de choisir les critères sur lesquels l'équipe de développement peut agir [FOR 89]. Ces attributs mesurables sont des **métriques** portant sur le code de l'application [BOE 78], la profondeur totale de la décomposition ou la complexité visuelle des graphes [FOR 89], la complexité des types de variables utilisées [VAN 75],

la complexité des structures de contrôle [MCC 76] qui se caractérise par un degré de modularité du modèle [KLO 90].

[FOR 89] propose une adaptation du critère de HALSTEAD [HAL 79] sur la compréhensibilité textuelle du code d'un composant, basé sur le volume de vocabulaire utilisé, et que nous pouvons appliquer ici au calcul de la complexité associée à l'interprétation (complexité textuelle) du corps d'une Boîte Fonctionnelle.

MACCABE [MCC 76] propose, pour la détermination de la complexité structurelle d'un graphe de connexions (complexité logique), de calculer le **nombre cyclomatique** du graphe concerné (calcul que l'on peut élaborer par identification du nombre de surfaces fermées) :

Critère de MACCABE [MCC 76] (Nombre cyclomatique) :

complexité = "Nb d'arcs" - "Nb de noeuds" + 2p

avec p = Nombre de graphe connexes

L'application de ce critère au **Grafcet** pose ici le problème de l'interprétation, en ce sens où une faible complexité *visuelle* d'un Grafcet **peut cacher** une forte complexité *structurelle et textuelle* induite par la structure interprétée que l'on peut décrire en Grafcet (test des états d'étapes, forçages de graphes). Il est donc nécessaire, pour une interprétation correcte des résultats issus de ce critère, d'effectuer aussi une mesure par un critère de complexité textuelle tel que celui proposé par [FOR 89] par exemple, et un calcul de *complexité visuelle* tel que défini par [FOR 89] :

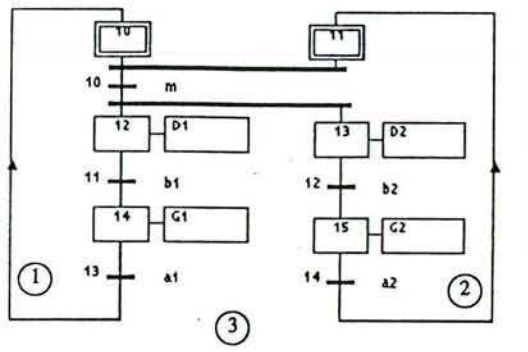
complexité visuelle = "Nombre d'arcs" - "Nombre de noeuds".

Critère de HALSTEAD modifié par FORSE [FOR 89]:

$$\textbf{complexité} = \frac{\text{"Nb d'opérateurs distincts"} * \text{"Nb d'occurences des opérandes"}}{2 * \text{"Nb d'opérandes distinctes"}}$$

Les Figures I.21a et I.21b montrent, sur deux Grafcet décrivant le même comportement mais une structure et une interprétation différentes, un calcul de leur nombre cyclomatique, du critète de complexité textuelle défini par FORSE et du critère

de complexité visuelle précédent. Nous remarquons, sur ces cas, que la complexité visuelle diminue au dépend des complexité textuelle et structurelle.



Nombre cyclomatique = 12 arcs - 11 noeuds + (2 * 1) = 3

Calcul du Nombre cyclomatique par le nombre de surfaces fermées :

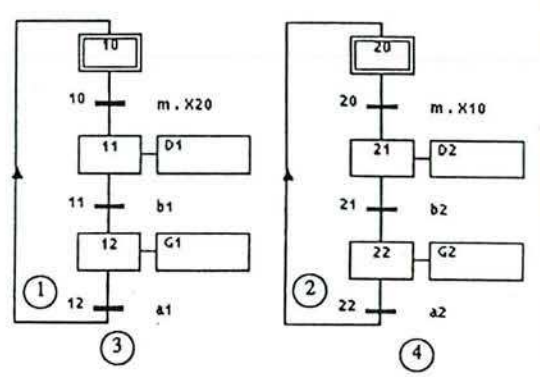
Nombre cyclomatique = ① + ② + ③ = 3

③ est la surface extérieure au graphe

Critère de complexité textuelle $= \frac{1 * 9}{2 * 9} = 0{,}5$

Critère de complexité visuelle = 12 arcs - 11 noeuds = 1

(a)



Nombre cyclomatique = 12 arcs - 12 noeuds + (2 * 2) = 4

Calcul du Nombre cyclomatique par le nombre de surfaces fermées :

Nombre cyclomatique = ① + ② + ③ + ④ = 4

③ est la surface extérieure au graphe connexe gauche

④ est la surface extérieure au graphe connexe droite

Critère de complexité textuelle $= \frac{2 * 12}{2 * 11} = 1{,}09$

Critère de complexité visuelle = 12 arcs - 12 noeuds = 0

(b)

Figure I.21. : Exemples de calculs de complexité

Dans ce cas, une **structure visuelle complexe** alliée à **une interprétation simplifiée** permet, en *simulation*, de déterminer plus rapidement les causes des blocages éventuels mais offre une moindre évolutivité du comportement, et réciproquement.

6. La vérification des prototypes par Simulation

Nous avons vu précédemment que la BF est la base même pour la description du comportement d'une installation. Dès lors, il est crucial de ne mettre à disposition que des modules déjà testés (ou "certifiés"). Ces modules, pour être réutilisables, ont été créés à partir de prototypes, indépendamment de leur contexte d'utilisation. La notion de prototype permet, lors de la phase de test unitaire, de simuler le comportement ainsi décrit pour vérifier la correspondance entre le modèle et le besoin réel du concepteur. Si leur **vérification** semble possible (le module a été bien construit), ce n'est certainement pas le cas de leur **validation** (le bon module a été construit).

Comme le corps d'un prototype est, par définition, exécutable, la vérification s'effectue par une exécution simulée. Cette simulation est de type "boucle ouverte", un opérateur observant les sorties du prototype et agissant en conséquence sur ses entrées.

Si bon nombre de telles simulations sont textuelles [ASA 88], il a semblé nécessaire d'assister l'opérateur dans cette tâche fastidieuse [COR 85], [COR 88], [LLO 87]. SPEX introduit la notion de **tableau de bord** (Figure I.22), ensemble d'afficheurs/générateurs associés à des variables de l'objet simulé. L'opérateur visualise sur ce tableau un ensemble de variables pertinentes pour une vérification de la bonne exécution du prototype.

La simulation choisie ici est de type interactif, destinée à tester "*en boîte blanche*" le prototype, c'est à dire en visualisant l'évolution de son comportement interne. Les entrées sont générées "en ligne", à discrétion de l'opérateur. Les sorties sont observées dynamiquement, pendant qu'une visualisation graphique du corps du prototype en exécution est possible.

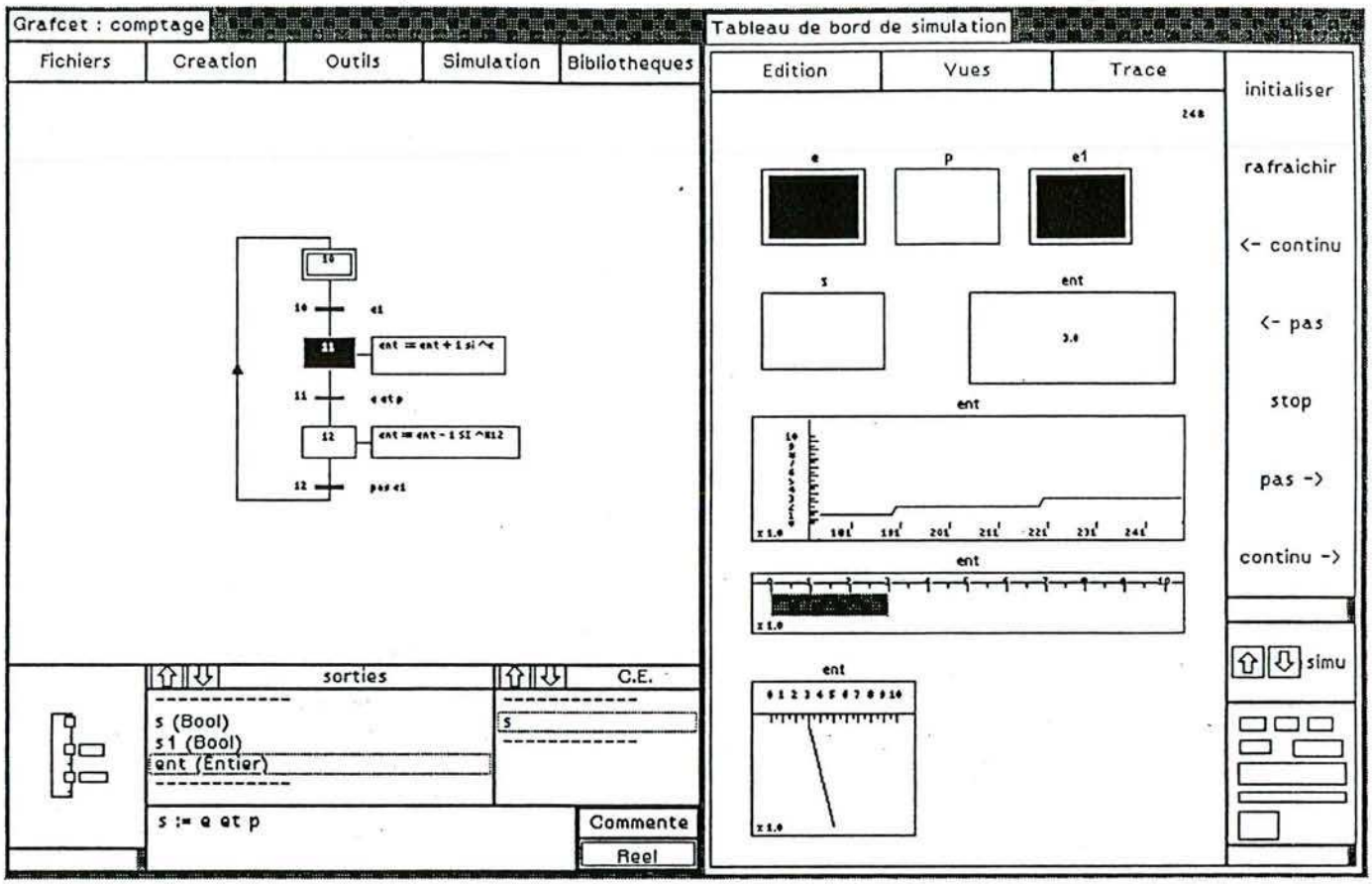


Figure I.22. : Le tableau de bord d'une simulation

Divers utilitaires sont destinés à assister le "vérificateur" : une remise en situation progressive permettant de se remettre rapidement dans un état passé (évitant ainsi les longues séquences de remise en situation); une vitesse de simulation réglable ; une trace textuelle d'évènements ; une marche continue ou pas à pas ; la définition, l'apprentissage et l'exécution de scénarios de test, ...

6.1. Trace textuelle des évènements

Une trace de la simulation permet d'enregistrer sur fichier (.TR) l'ensemble des changements d'état de toutes les variables de l'objet simulé par rapport au cycle de

scrutation de la simulation (Figure I.23a). Cette trace autorise une remise en état passé de la séquence et pourrait être analysée pour une meilleure exploitation des résultats de simulation.

6.2. Scénario de Test sur SPEX

L'apprentissage d'un scénario permet la mémorisation automatique sur fichier (.C), en langage C, de l'ensemble des évènements générés par l'opérateur sur le tableau de bord (Figure I.23b). Ce fichier peut être complété manuellement à l'aide d'un éditeur de texte à la condition de respecter la syntaxe suivante:

pour l'affectation de variables:

booléennes :

booleenAffecter(_NOMVARIABLE, VALEURBOOLENNE)

entières:

entierAffecter(_NOMVARIABLE, VALEURENTIERE)

réelles:

reelAffecter(_NOMVARIABLE, VALEURREELLE)

pour le test de variables :

NOMVARIABLE

Avec NOMVARIABLE = Nom de la variable
VALEURBOOLENNE = "1" ou "0"
VALEURENTIERE = entier
VALEURRELLE = réel

Tout autre code respecte complètement la syntaxe du Langage C.

Le scénario a accès au compteur cycle de scrutation par la variable **compteur**. L'accès au compteur de cycle permet de générer des changements d'états de variables à des instants déterminés. L'opérateur de test peut aussi introduire dans son scénario un "code C" générant l'apparition de défauts aléatoires (par l'utilisation de la fonction

standard du Langage C "rand()") et vérifier ainsi la réaction de son système face à ces évènements imprévus.

L'exécution d'un scénario laisse la possibilité à l'opérateur d'agir sur la tableau de bord, cette action étant prioritaire.

L'association d'un **scénario** de test et d'une **trace textuelle** permet de vérifier, par analyse postérieure de la trace, la concordance d'une conception par rapport au comportement désiré de l'application. Une **spécification exécutable** décrit un comportement général par la définition d'une maquette exécutable à laquelle nous pouvons associer un scénario de test.

Nous pouvons (et nous devons) réutiliser cette spécification (qui a été créée pour cela) en **conception exécutable**. A priori, il n'est pas possible de raffiner le comportement général décrit lors de la spécification car il induit des contraintes temporelles en **conception.** Il est donc nécessaire de redéfinir une structure et des comportements plus fins permettant une vérification de la conception par simulation. La **réutilisation** des scénarios de spécification en conception exécutable fournit le **lien** entre spécification et conception dont les résultats (trace textuelle) respectifs peuvent être **comparés.** Il est ainsi possible de vérifier l'**adéquation** de la conception par rapport aux besoins initiaux.

Le scénario peut aussi être utilisé pour l'affectation d'un "comportement" particulier d'opérateur de conduite à une BFE Pupitre. Il évite ainsi, à l'opérateur de test, la commande "manuelle" des actions opérateurs de conduite et peut générer aléatoirement des **ordres erronés** pour vérifier la prise en compte d'un comportement anormal par simulation.

La simulation d'une **application** ou d'un **DF** utilise les mêmes principes que pour la vérification des BF (Figure I.24) : tableau de bord, utilitaires de simulation (trace textuelle, scénario de test, ...)

Figure I.23. : Exemple de Scénario et de Trace textuelle

```
/*``includes``*/
#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <memory.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "BistableVerinSimpleEffet.h"

AexecuterScenario()
{
 switch (compteur) {
  case 2 :
       booleenAffecter(_DSortie,1);
  break;
  case 4 :
       booleenAffecter(_FCCourse,1);
  break;
  case 6 :
       booleenAffecter(_DSortie,0);
  break;
  case 7 :
       booleenAffecter(_DRentrée,1);
  break;
  case 9 :
       booleenAffecter(_FCCourse,0);
  break;
 }

/* on insere manullement ici le code */
/* à executer à chaque cycle de scrutation */

 if(rand()==compteur)
  booleenAffecter(_FCCourse, !FCCourse);
/* collage aléatoire du capteur FCCourse */

}
```

(a)

```
Trace simulation
  xxx.Xn  etat etape n du graphe xxx
  xxx.CXn  numero cycle d'activation
        de l'etape n du graphe xxx
  structure du fichier
  Nom variable                    Etat    Nouvel
                                  précédent  Etat
COMPTEUR CYCLE 1
   BistableVerinSimpleEffet.X12   0   1
   PositionRentree (Bool          0   1
   BistableVerinSimpleEffet.CX12      1

COMPTEUR CYCLE 2
   DSortie (Bool)                 0   1
   AutoSortie (Bool)              0   1
   BistableVerinSimpleEffet.X12   0   1
   BistableVerinSimpleEffet.CX12  1   2

COMPTEUR CYCLE 3
   PositionRentree (Bool)         1   0
   BistableVerinSimpleEffet.X12   1   0
   BistableVerinSimpleEffet.X13   0   1
   Sortie (Bool)                  0   1
   BistableVerinSimpleEffet.CX13      3

COMPTEUR CYCLE 4
   FCCourse (Bool)                0   1
   Sortie (Bool)                  1   0
   BistableVerinSimpleEffet.X13   1   0
   BistableVerinSimpleEffet.X10   0   1
   PositionSortie (Bool)          0   1
   BistableVerinSimpleEffet.CX10      4

COMPTEUR CYCLE 6
   DSortie (Bool)                 1   0
   AutoSortie (Bool)              1   0
.....
```

(b)

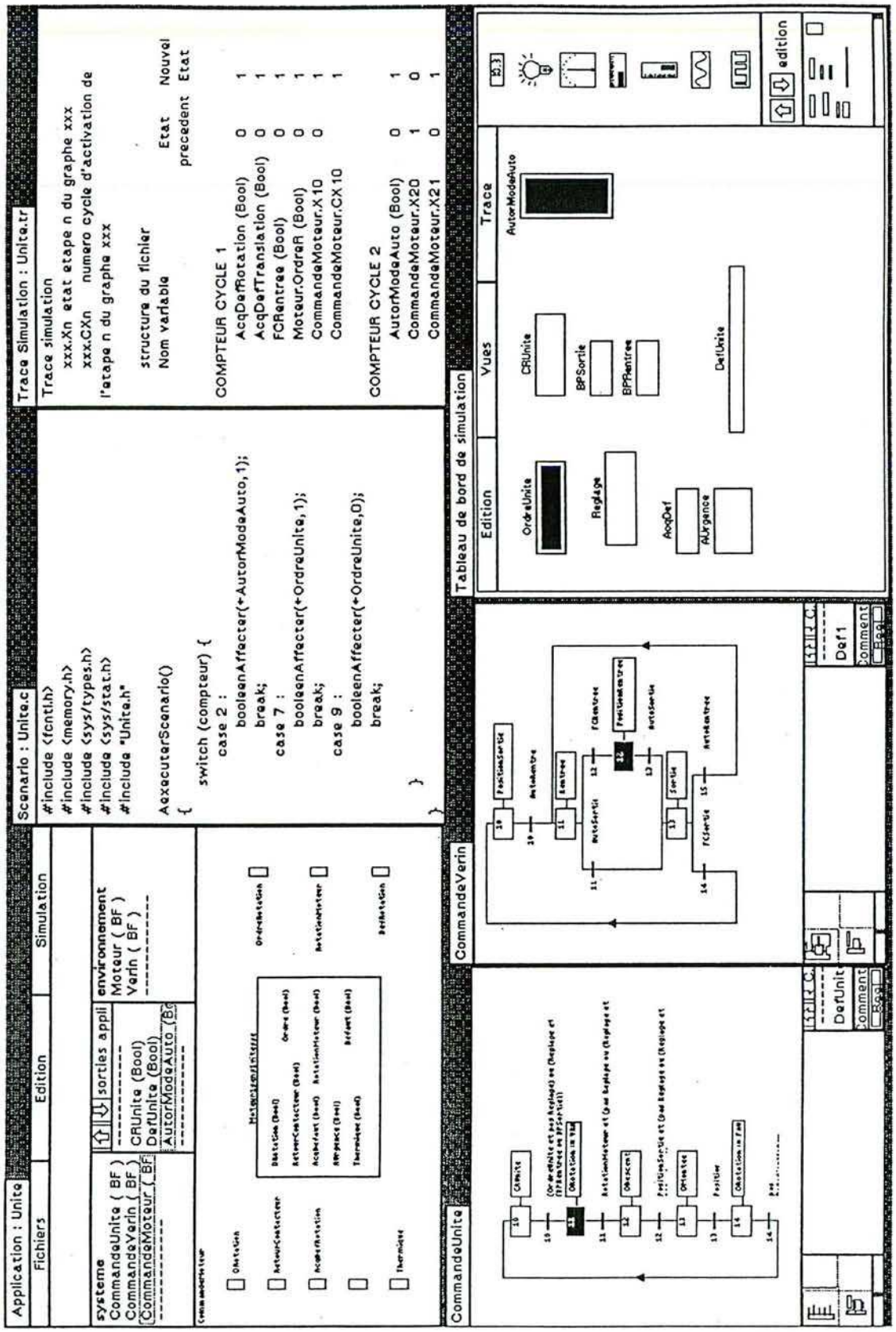


Figure I.24. : Exemple de Simulation d'une Application

6.3. Algorithme d'interprétation

L'algorithme d'interprétation, en simulation de **prototypes** BF/DF et **Application** est celui recommandé par l'AFCET [AFC 83][GRE 85], c'est à dire à **réalisation synchrone** (par rapport aux entrées des "Boîtes") et à **évolution synchrone** sans recherche de stabilité (Figure I.25). L'ensemble des comportements composant la structure d'un Diagramme Fonctionnel est ici mis à plat, ce qui n'introduit aucun retard de propagation de l'information en fonction du niveau de structuration. A noter qu'à l'intérieur d'une BF, le combinatoire est interprété dans l'ordre de classement dans les listes de combinatoires d'entrée et de sortie pour le grafcet, dans l'ordre des réseaux pour les logigrammes et les schémas à relais. L'ordre de définition des exemplaires dans une Application n'a pas d'incidence sur le comportement de l'ensemble.

Notons cependant la difficulté, lors de la création d'un DF ou d'une Application, de tenir compte conceptuellement de ce type d'interprétation qui ne correspond pas au schéma de pensée naturel en analyse qui serait plutôt de type asynchrone ...

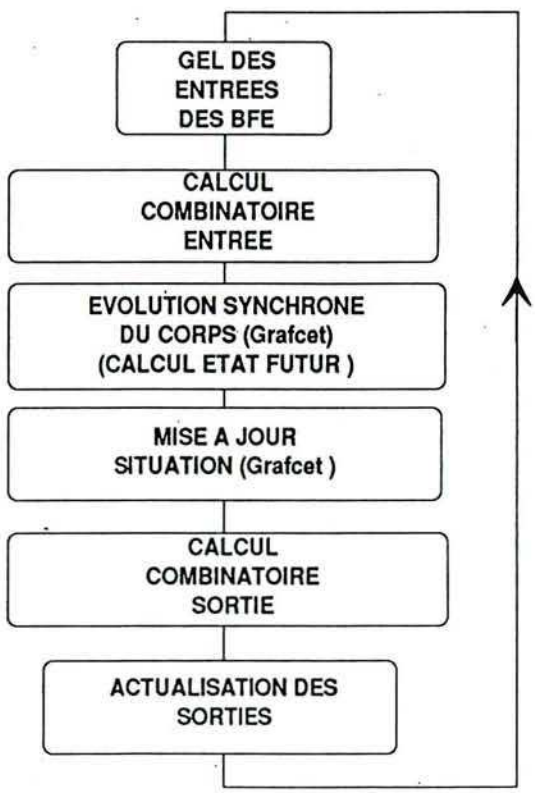


Figure I.25. : Algorithme d'Interprétation AFCET

III. CONCLUSIONS ET PERSPECTIVES

SPEX est un environnemment pour la spécification et la conception de Systèmes Automatisés de Production qui met en oeuvre quelques principes importants du **Génie Logiciel** pour assister l'Automatiseur et le Génie Automatiseur. En aucun cas, il ne fournit une aide quant à la définition **sémantique** des composants de l'application mais uniquement un ensemble de moyens et démarches permettant une amélioration de la **qualité** de l'automatisation par une meilleure réutilisation du **savoir-faire** de l'entreprise ou d'un domaine d'application particulier.

L'outil étant spécifique au domaine de l'automatisation, il permet de modéliser non seulement la partie commande de l'application, mais aussi la partie opérative en vue de son émulation. En fait, lors de la simulation de l'application, SPEX ne fait aucune différence entre sa **partie système** et sa **partie environnement**. Si, dans une première approche de la simulation, cette séparation ne paraît pas indispensable, il est cependant nécessaire, pour être au plus près du comportement réel, que chaque partie (système et environnement) ait sa propre simulation, totalement indépendante et asynchrone l'une de l'autre. Cette perspective d'évolution de SPEX va dans le sens de la définition d'**un modèle organique** que nous présentons au chapître III et dans lequel l'architecture matérielle cible doit être pris en compte ainsi que les aspects multi-processeurs qu'elle induit.

SOMMAIRE
Chapitre II

I. SEMANTIQUE DES OBJETS D'AUTOMATISATION

1. Introduction

SPEX permet la création et la manipulation de comportements mais n'apporte pas de solution quant à la **sémantique** des objets ni de méthodes pour définir les "bons" objets en rapport avec une application particulière.

Si l'on repend les trois dimensions d'un objet (Figure II.1) définies par [QUA], nous pouvons placer les objets manipulés dans SPEX sur les dimensions "**Pragmatique**" (c'est à dire le **fonctionnel**) et "**Syntaxique**" (c'est à dire **structurel**), point fort de SPEX. Il reste la dimension "**Sémantique**" que l'on ne sait pas encore traîter de façon générale.

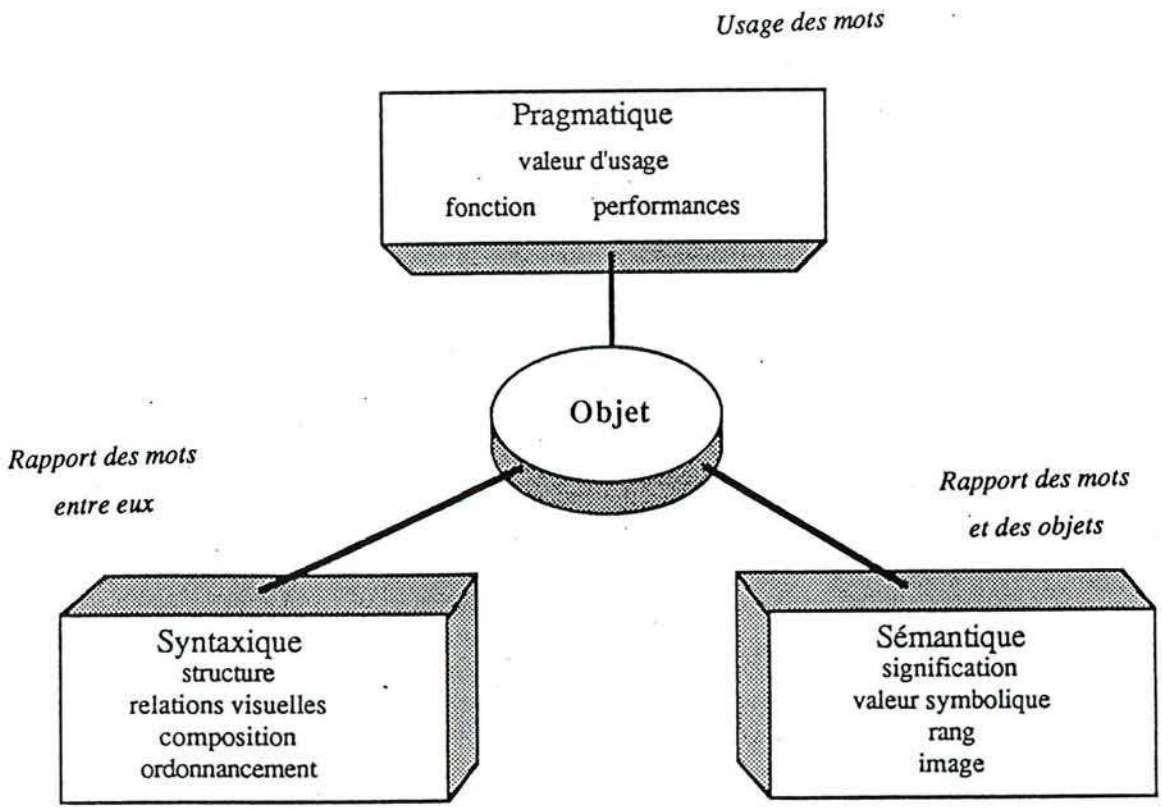


Figure II.1. : Les 3 dimensions de l'objet [QUA]

Chaque composant peut être défini par son comportement propre indépendamment de la fonction qu'il doit remplir au sein d'une application. Ce composant représente un **objet du "monde réel"**, indépendant de l'**espace des solutions** du problème concerné et réutilisable quel que soit le domaine d'application. En complément de l'objet purement informatique dont il reprend les mécanismes (cf Chapitre I), il possède un ensemble de "**vues**" documentaires, logicielles, informationnelles, matérielles, etc ..., chacune d'elles définissant une partie de son comportement selon un point de vue donné. Cependant, l'encapsulation de la connaissance (et donc de l'expérience) que nécessite sa réutilisation effective ne peut guère s'envisager que par classes d'applications.

> "**Un outil-méthode (un objet) d'Automatisation associe, par une synthèse globale ou par activité du cycle de vie, une base descriptive à une démarche dans un environnement dédié à une classe d'application**" [MOR 88]

Pour une classe d'application particulière, il semble donc possible de définir des comportements génériques qui enrichiront la base de comportement de SPEX.

Des méthodes de spécification "orientées objet" sont développées pour déterminer les "objets" et leur **sémantique** à partir des phases de spécifications [BAI 89] [WAR 89] mais aucune n'apporte encore une solution complète, systématique et "intelligente" d'assistance au spécifieur.

[BOO 88], [MEY 88], [ABB 86] stipulent qu'une méthode de conception orientée objet ne concerne que les phases de conception et réalisation du logiciel et donc ne couvre qu'une partie du cycle de vie. Il faut donc associer, en amont, une analyse du problème par des méthodes privilégiant les flots de données. Ce type d'analyse devrait permettre de trouver les objets "opportuns" du système ainsi que la définition de leurs rôles respectifs.

Comme on ne sait pas reconnaître ces objets, des mécanismes "riches" mais relativement peu rigoureux sont prévus dans les environnements de conception pour pouvoir traduire n'importe quelle classification (par exemple, l'utilisation de l'héritage multiple).

2. Les modèles

Ces modèles peuvent être classés, de façon générale, en trois classes en fonction du niveau d'abstraction de la modélisation (Figure II.2). [VOG 88] propose une distinction entre :

- les **modèles pratiques** qui, par un formalisme adapté et des conventions spécifiques d'utilisation [THE 85] [GRA 90], [SFA 90], apportent une sémantique pour l'identification et la spécification de l'expertise ;

- les **modèles cognitifs** constituent, autour de formes simples, des vecteurs de perception et d'interprétation de l'expertise du monde réel. Ils contribuent à une formalisation de la sémantique des objets identifiés pour des classes d'application particulières. Dans le domaine particulier du **Génie Automatique**, il semble que nous puissions classer ces modèles suivant plusieurs approches, non exclusives, en fonction du type de décomposition choisi (ascendante, descendante, mixte) et du point de départ de la modélisation (technologie, topologie physique, produit, fonction).

- les **modèles informatiques** favorisent la structuration et la réutilisation par l'apport de méthodes et méthodologies propres à fournir une aide au concepteur. Ces modèles, par contre, n'apportent aucune **sémantique** quant aux objets manipulés et tendent trop souvent, comme par exemple dans le cas du paradigme objet [MAS 89], à s'éloigner du "réel". Son "*utilisation «sauvage» ... donne souvent naissance à des modèles illisibles et incohérents*" [LEP 88], si nous voulons les utiliser comme modèles pratiques, ou, pire encore, cognitifs.

[VOG 88] "*Les méthodes classiques de spécification et de conception, SADT, OOD, ne prennent en compte que les deux extrémités du vecteur de conception : la formulation du problème en langue naturelle, et sa représentation dans le méta-langage informatique choisi. Il manque à ces modèles le niveau sémantique des catégories sous-tendant directement la formulation du problème, qui serviront partiellement de guide à la représentation formelle finale.*"

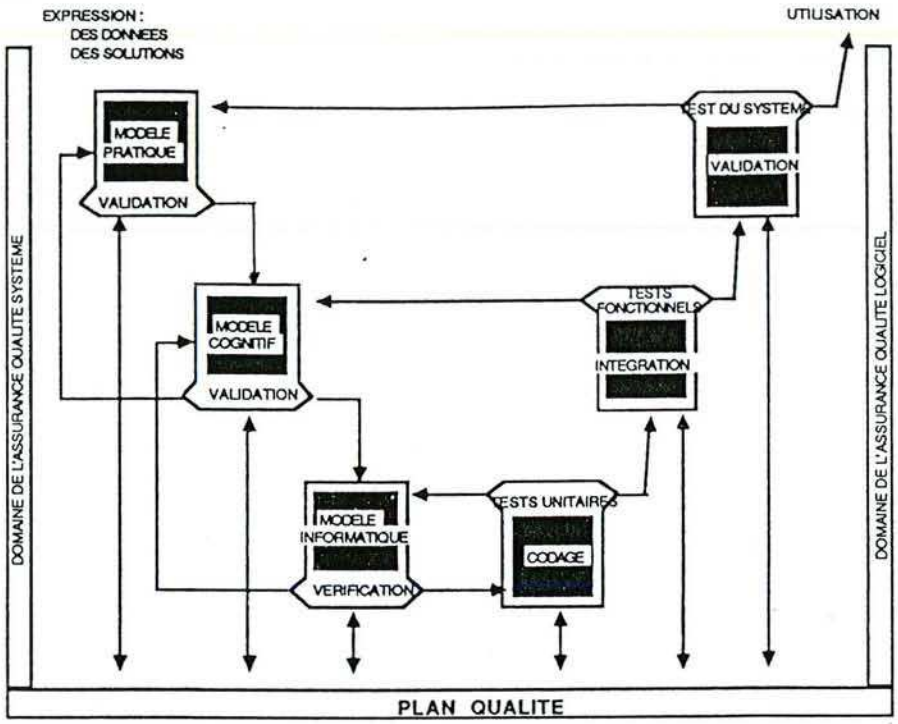


Figure II.2. : Le cycle de vie en diachronie [VOG 88]

SPEX, outil pour le **Génie Automatique**, est, comme beaucoup d'environnements du Génie Logiciel, un **modèle informatique** qui met à la disposition de l'utilisateur, des mécanismes aptes à la manipulation des comportements créés mais ne fournit aucune aide quant au choix des "bons" comportements employés ni de leur qualité propre.

Contrairement au **Génie Logiciel**, le **Génie Automatique** est un domaine liant conjointement des parties logicielles et des parties matérielles. Si, comme cela est le cas en Génie Logiciel, l'identification et la généralisation d'"objets" logiciels ayant un fort degré décisionnel semble encore difficile à réaliser, ce but paraît beaucoup plus à notre portée pour des entités représentant l'aspect comportemental d'éléments opératifs ou modélisant des comportements très typés pour des **classes d'applications particulières**. En effet, une observation des objets du monde réel devrait permettre, dans une certaine mesure, de construire des modèles par une approximation relative de la réalité pour des classes particulières de composants.

Ces modèles apportent une certaine sémantique dans le contexte particulier du **Génie Automatique** par une **correspondance directe** entre réel et comportement.

Nous présentons, dans ce qui suit, quelques modèles pratiques et cognitifs applicables au Génie Automatique. Nous verrons que, dans ce cadre, le modèle **d'actinomies**, considéré par [VOG 88] comme un modèle cognitif pour un domaine

général, est en fait un modèle pratique dans **notre contexte** et nécessite, pour intégrer une certaine connaissance **sémantique**, une formalisation de règles particulières de manipulation.

Par rapport aux définitions initiales de la notion de "Poste" [VOG 87] définies par le L.A.C.N., nous proposons une tentative de formalisation de la sémantique des modèles pour leur implémentation.

1. Modèles pratiques

1.1. Actinomies

L'actinomie [VOG 88] (Figure II.3) est un formalisme représentant l'enchainement des différentes opérations nécessaires à l'accomplissement d'un objectif donné. Elle se constitue par un échafaudage de séquences enchâssées, juxtaposées, alternées ou enchainées lui définissant une trame. Chaque séquence est composée de trois parties (ou actème) suivant un modèle quinaire, c'est à dire décrivant se qui doit être fait **avant, pendant** et **après** le noyau de la séquence:

- **ouverture** de la séquence : l'ouverture prépare une ressource ou l'opération suivante qui, pour être réalisée, présuppose la réussite de la préparation ;
- **noyau** : le noyau est le coeur même de l'actinomie et décrit les opérations réalisant l'objectif fixé. Il peut lui-même être récursivement composé de séquences ;
- **clôture** de la séquence : réelle ou virtuelle, elle libère les ressources éventuelles et restaure l'état initial du système pour l'enchaînement d'une autre séquence.

L'ouverture et la fermeture sont caractérisées par des actions antinomiques qui s'annulent réciproquement.

Ouverture	Charger **une** Pièce *brute*	
	Brider **une** Pièce *brute*	Percer **une** Pièce *brute bridée*
Noyau	Usiner **une** Pièce *brute bridée*	Fraiser **une** Pièce *brute bridée*
	Débrider **la** Pièce *usinée bridée*	Tarauder **une** Pièce *brute bridée percée*
Fermeture	Décharger **la** Pièce *usinée débridée*	

Figure II.3. : Actinomie de la fonction "Usiner Pièce brute"

Cette interprétation de la description d'un séquencement d'opérations associée aux critères antinomiques et à des règles de fusion de séquences [PAR 90] décrit ainsi un modèle cognitif en fournissant une aide partielle à l'élaboration d'une sémantique (d'un comportement) (Figure II.4)

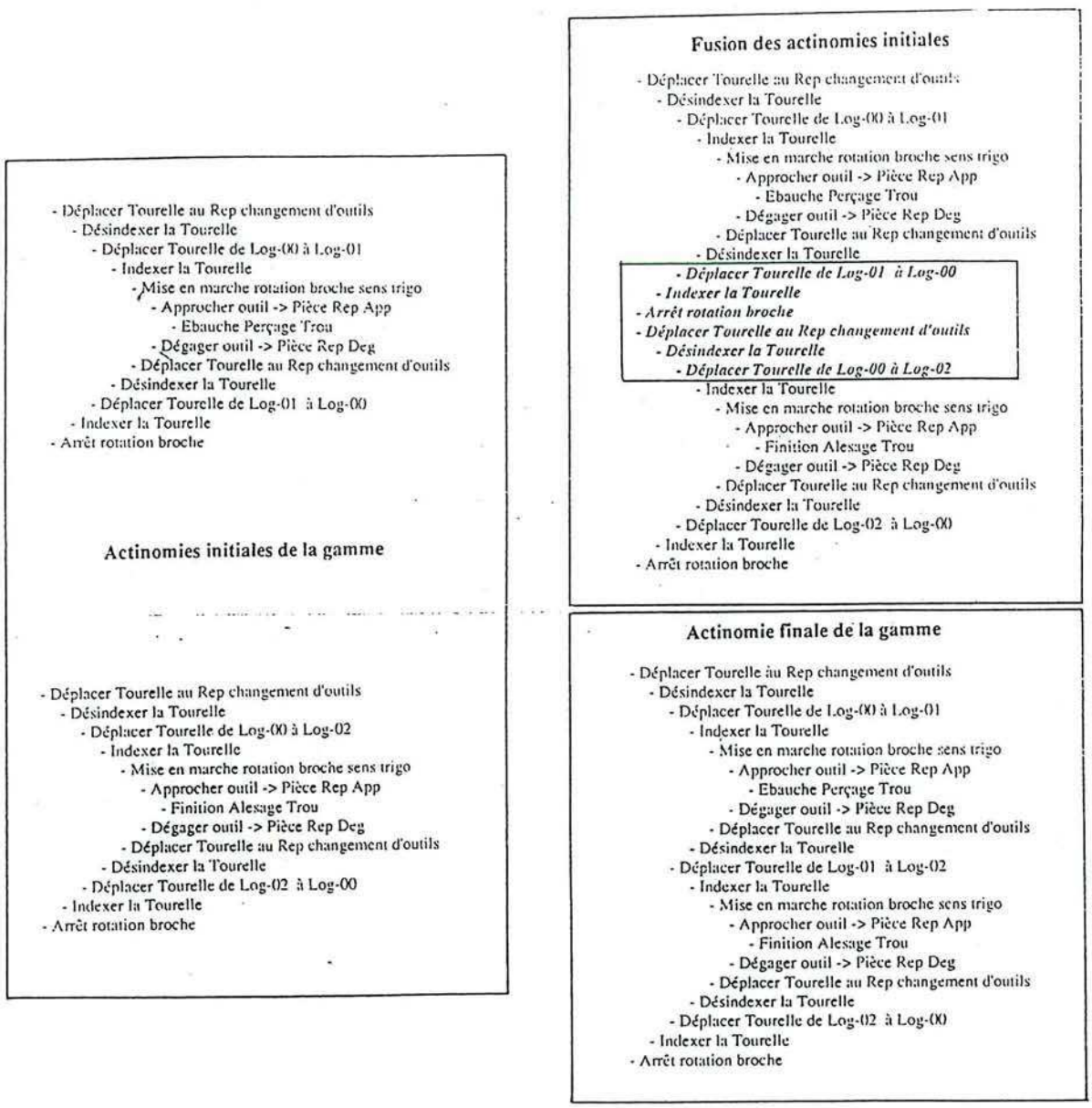


Figure II.4. : Exemple de fusion"

D'autres modèles pratiques permettent d'apporter une certaine sémantique "d'intention" :

- par l'association d'un formalisme et de règles de lecture spécifiques, comme par exemple l'actème [VOG 88] (que nous présentons dans le Chapitre III, § I.2.2), qui est centré sur l'action provoquant un changement d'état d'un objet. Cette transformation est activée par un sujet et elle est supporteé par un instrument.

- par la formalisation d'un **modèle de référence particulier**, comme le montrent les travaux en cours dans le projet ESPRIT/DIAS [DIA 89] alliant, un formalisme SADT et l'approche CMMS (Controle, Maintenance, Gestion Technique) [CMM 89] pour la définition d'un **modèle de référence particulier** DIAS à la classe d'application "Vannes" (Figure II.5), qui constitue déjà un modèle cognitif dans un contexte EDF, et qui sera ensuite généralisé à **modèle de référence partiel** DIAS pour d'autres domaines d'application, dans le même contexte [IUN 91].

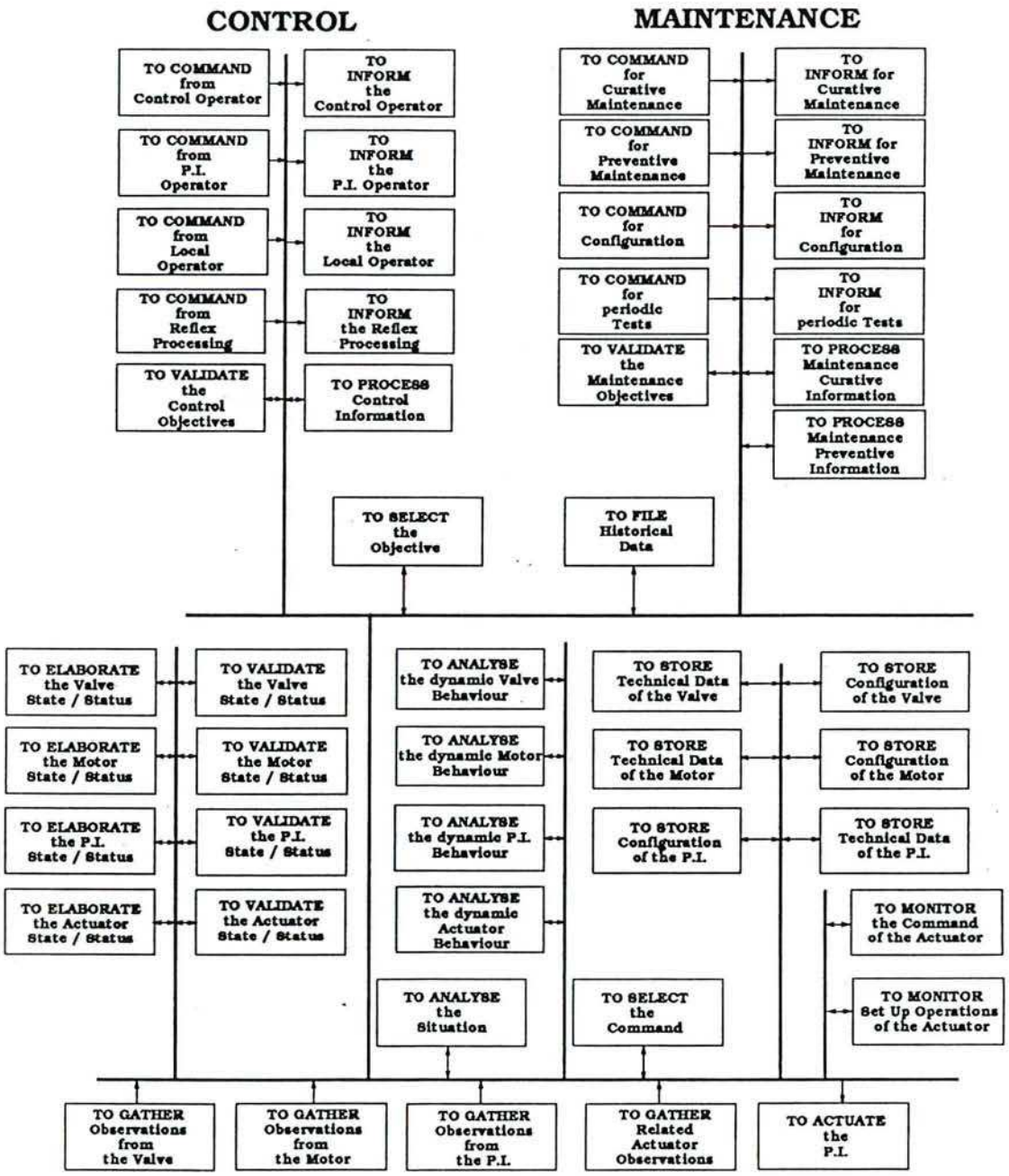


Figure II.5. : Le modèle de référence particulier DIAS

2. Modèles cognitifs

2.1. Approche "orientée objet" informatique

Grady BOOCH [BOO 86] classe les objets en 3 catégories en fonction de leurs rôles au sein de l'application (Figure II.6):

- le rôle **serveur (ou ressource)** : ces objets mettent à disposition d'un niveau supérieur d'abstraction des opérations ou fonctions nécessaires à l'accomplissement d'une opération ou fonction. Ils ne peuvent solliciter d'autres objets de même niveau.

- le rôle **agent** : un tel objet assure un rôle de transformation de ses données d'entrées, il peut solliciter un serveur ou un autre agent, le résultat de son opération peut **profiter** à un autre objet que le demandeur.

- le rôle **acteur** : cet objet à sa propre autonomie et contrôle l'activité des autres objets en leur demandant d'effectuer certaines opérations. Il n'offre aucune opération aux objets de même niveau.

Nous noterons qu'un agent ou un serveur peuvent être décomposés suivant le même principe. L'acteur, ayant sa propre autonomie, est un objet terminal non décomposable suivant ce modèle.

Abbott [ABB 86] distingue en plus, pour un objet, le rôle de **type abstrait de donnée** qui définit les variables du problème au sens structure de données, fichiers, ... Nous pensons, comme l'exprime [CAL 90] que cette donnée est en fait "encapsulée" dans l'objet producteur qui doit la restituer à tout demandeur. Par exemple, l' objet **serveur** "*Base de Donnée*" intègre toutes ses informations propres et ne fournit que celles qui lui sont demandées, sous réserve qu'elles soient d'accès autorisé pour le demandeur.

Cette classification des différents rôles attribués aux objets s'apparente au modèle **Producteur-Distributeur-Consommateur** [THO 90] pour lequel l'acteur est le producteur, le serveur est le consommateur, l'agent est le distributeur.

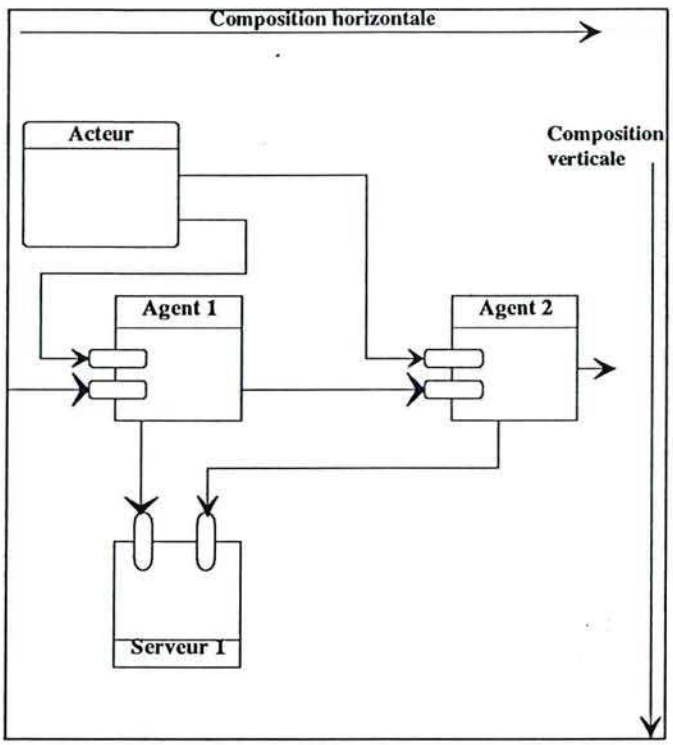


Figure II.6. : Les 3 catégories d'objets [BOO 86]

Nous distinguons ainsi 2 axes de composition des objets :

- *l'axe horizontal* : Il est composé d'échanges d'informations entre agents ou acteurs et agents et exprime la coopération entre les objets.

- *l'axe vèrtical* : il est composé de la hiérarchie des services qui communiquent par requêtes et compte-rendus ou indications et réponses [THO 90]

Ce type de modélisation en fonction des rôles associés aux différents objets favorise la modularité, l'évolutivité et donc leur réutilisation d'où leur possible

intégration au sens CIM (Computer Integrated Manufacturing) dans les systèmes automatisés [GAC 90] mais reste encore, à l'image du Génie Logiciel purement **informatique**. Il doit, pour être praticable, se déduire de l'**architecture physique du système**, ce que nous montrons par la définition d'un modèle "objet réel"/"porte-objet réel" pour lequel nous identifions les "objets" sur la base d'objets existants (ou potentiellement existants).

2.2. Approche orientée "objet d'automatisation"

Approche descendante basée sur la topologie physique

Proposition d'un modèle objet/porte-objet

Le concept de **réutilisation** nous amène à rechercher un modèle de comportement général de tous les acteurs présents dans un **S**ystème **A**utomatisé de **P**roduction de type manufacturier qui permettrait la construction d'une application d'automatisme par **assemblage** d'exemplaires d'un tel composant. Celui-ci doit posséder une **interface de communication** bien définie pour permettre son utilisation quelle que soit l'application [HAU 87] [MUN 88]. La structure de l'automatisme ainsi décrit pourrait être dérivée d'un **modèle de référence d' installation.**

Les principaux **acteurs** d'une scène contribuant à la réalisation de produits manufacturés mettent en jeu un certain nombre de composants tels que les outils de transformation, les pièces, les outils de mesures. Les activités que ceux-ci vont subir sont appelés procédés.

L'automatisation de la production nous amène à gérer l'ensemble des composants interchangeables du SAP réalisant un procédé. Par exemple, un préhenseur (pince) peut être remplacé par un autre en fonction du type de pièce à manipuler. La relation entre avant-bras et pince est alors rompue. Le procédé ici identifié est un assemblage/désassemblage entre un objet (le préhenseur) et son porte-objet (l'avant-bras).

Dans le cas d'un usinage, un changement de point de vue peut nous amener, dans un but de généralisation, à considérer l'outil, non plus comme un objet simple mais comme un porte-objet et le procédé d'usinage comme un procédé d'échange de matière entre l'outil (qui "prend" de la matière) et la pièce restante (qui "donne" une partie de pièce) (Figure II.7). Cette généralisation, appliquée de

la même manière au procédé de prise d'information par un capteur sur un objet, nous permet de proposer, sous ces hypothèses que :

- tout procécé peut être caractérisé comme un échange d'un objet (physique, matière ou information) entre deux porte-objets ;

- tout acteur d'un procédé peut être considéré comme un objet ou un porte-objet à des instants donnés. La différenciation se fait par la nature du procédé qui l'emploie. Un tel "acteur" peut, au cours du processus de fabrication, être considéré comme "objet" lorsqu'il subit le procédé ou "porte-objet" lorsqu'il participe au procédé. Par exemple, au cours du transport d'un préhenseur entre le magasin et le poignet qui doit le recevoir, le préhenseur est un "objet". Lorsque ce dernier est utilisé pour le transport d'une pièce, il est "porte-objet" ; le procédé d'affûtage met en oeuvre un porte-objet (l'outil affûteur) et un objet (l'outil affûté).

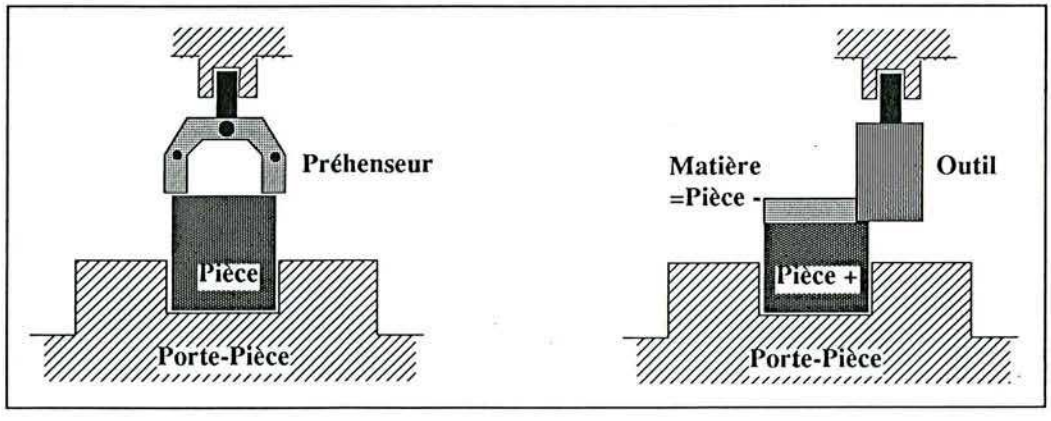


Figure II.7. : Le procédé d'usinage

Résumons les différents procédés dans la liste suivante en tenant compte de notre généralisation :

Porte-Objet1	Porte-Objet2	Objet	Procédé
Porte-Pièce	Porte-Pièce	Pièce	Echange
Porte-Outil	Porte-Outil	Outil	Echange
Porte-Capteur	Porte-Capteur	Capteur	Echange
Pièce	Porte-Pièce	Pièce	Echange
Outil	Porte-Outil	Outil	Echange
Outil	Pièce	Matière	Echange
Capteur	Pièce	Information	Echange
Capteur	Outil	Information	Echange
Capteur	Capteur	Information	Echange

Il est ainsi possible de généraliser l'architecture de commande d'un système de production manufacturière comme un assemblage de tels acteurs en fonction de sa topologie physique (identification des objets, porte-objets et de leurs liens potentiels) [VOG 87] .

L'automatisation d'une installation met en oeuvre des moyens matériels dans le but de satisfaire un objectif, la commande du système de production. Chaque acteur du processus (Figure II.8) prend sa place dans la structure globale du système et participe à la réalisation de l'automatisme. Celui-ci dispose d'un certain nombre de moyens, les ressources, contrairement aux hypothèses implicites et souvent simplificatrices appliquées "inconsciemment" (ou consciemment) par les automatiseurs actuels, qui tendent à n'affecter à un acteur qu'un seul moyen. Cela n'est pas le cas à des niveaux "hauts" de spécification, par exemple, pour le pilotage d'installations flexibles de par la nature du procédé, mais cela a tendance à l'être beaucoup trop pour ce qui concerne les "machines" que l'on ramène souvent à des ensembles de Machine Transfert. L'acteur doit gérer l'enchaînement des différentes opérations effectuées par les ressources dans le but de satisfaire les demandes provenant d'un niveau supérieur. Un Module de Contrôle-Commande (MCC) modélise ainsi le comportement d'un acteur par rapport à celles-ci.

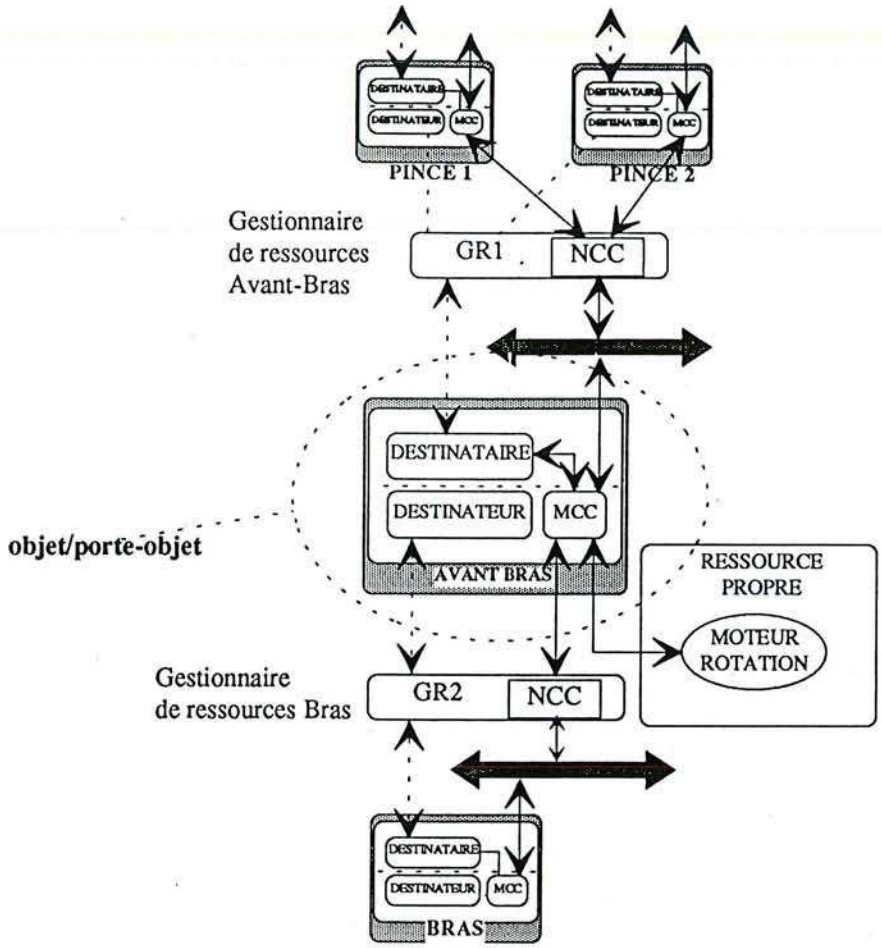


Figure II.8. : Un acteur

Une installation peut alors être vue comme un ensemble d'acteurs (objets et porte-objets) en relation par des gestionnaires de ressources [BEL 90]. Il n'existe actuellement aucune méthode rigoureuse pour l'identification des acteurs et pour la détermination de leur dépendance vis à vis de gestionnaires de ressources. Ces derniers doivent être identifiés et définis de manière aussi formelle que les acteurs.

Notre problème est différent du Génie Logiciel par le fait que nous modélisons une installation qui possède déjà des objets physiques identifiables par les procédés qu'ils mettent en oeuvre, par exemple outil, pièce, porte-pièce, ... et ayant des relations mutuelles caractérisées par la structure morphologique des équipements. Il est donc possible d'établir une nomenclature de ces objets physiques dont le nombre dépendra, bien sûr, de la complexité du problème et du niveau d'automatisation désiré. La difficulté réside dans une organisation logique des relations de ces objets et des gestionnaires de ressources qu'ils induisent.

La Figure II.9 décrit une architecture de commande distribuée suivant ce modèle et dans laquelle, Oi est un objet, POi est un acteur susceptible de devenir porte-objet et GRi un gestionnaire de ressource pour l'allocation dynamique des acteurs nécessaires à la réalisation du processus concerné.

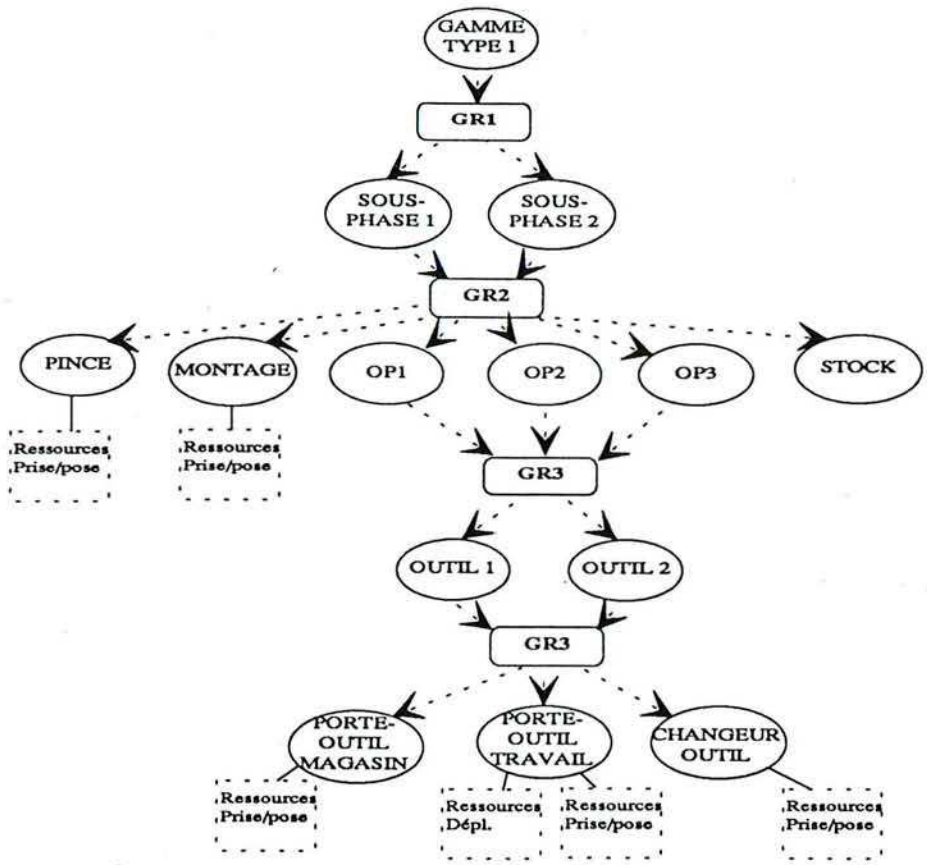


Figure II.9. : Architecture distribuée

Ce type de modélisation met en évidence la nécessité d'une intégration de composants génériques d'automatisation contribuant :

- à **l'ouverture** du système permettant une évolution des spécifications sans remise en cause majeure de l'ensemble de la commande de l'installation. En effet, les gestionnaires de ressources sont les seules interfaces spécifiques

réalisant une adaptation des composants génériques à leurs utilisations dans une application particulière,

- à la **flexibilité** de l'installation par la création dynamique des relations entre les acteurs sans détermination **figée** préalable des moyens réalisant les procédés mis en oeuvre,

- à une meilleure **indépendance fonctionnelle** des différents acteurs par une forte cohérence interne des objets et un faible couplage inter-objets.

- à une meilleure **réutilisatibilité** des acteurs par la définition d'une **sémantique** d'interface standard qui contribue à un polymorphisme des objets de l'architecture de commande.

Après affinage de la définition du contenu détaillé de ces objets d'automatisation et implémentation sur l'outil SPEX, un interfaçage avec PROPEL [PRO 90] est envisagé. Cet interfaçage n'a pas encore été réalisé car il nécessite des algorithmes complexes, que l'on écrira en Langage C, mais qui ne peuvent pas encore être implémentés de façon optimale dans la version prototype actuelle de SPEX.

Le générateur de gamme PROPEL, basé sur un système expert, permet de déterminer les gammes d'usinage possibles d'une pièce en fonction d'une description géométrique de la pièce, de la description des machines et des outils disponibles. Chaque gamme est décomposée en phases, sous-phases et opérations en fonction des changements de machines et outils nécessaires à son exécution. A partir d'une gamme retenue, l'identification des relations entre "objets" et "porte-objets" devrait permettre [PAR 90] de générer automatiquement l'architecture de commande du processus de fabrication en se référant au modèle "objet/porte-objet" présenté et les opérations de transformation et transferts qu'elle induit.

Approche ascendante basée sur la technologie

Modèle d'Elément de Partie Opérative

Classiquement, les études amenant à la réalisation d'une Partie Opérative (PO) induisent une démarche "structurée" partant de la fonction à réaliser sur un produit, par exemple, et aboutissant à la définition de moyens technologiques élémentaires (qui sont généralement des composants standards) [FRA 87] permettant la concrétisation physique de cette fonction. Les composants qui en résultent doivent pouvoir être réutilisés pour la description de la Partie Commande (PC) par la définition d'**Eléments de Partie Opérative** (E.P.O) founissant, chacun, une image d'un comportement élémentaire tout aussi standard (Figure II.10).

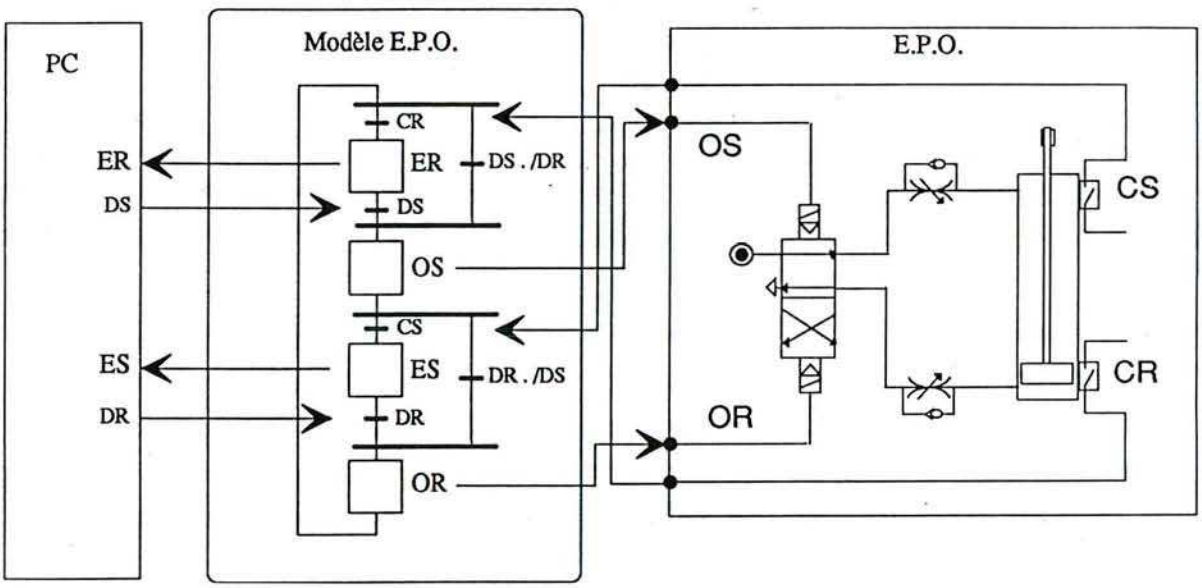


Figure II.10. : Un Elément de Partie Opérative

Le concept de modélisation d'E.P.O. [ISM 83], [ADE 84], [LHO 85] [PRU 86], [DEF 86b], [ALA 86] peut être utilisé suivant deux approches :

- Ces modèles particuliers peuvent être employés pour "**émuler**" le comportement de la partie opérative sans qu'elle ne soit physiquement présente (Figure II.11). Cette première approche permet, pour le test des

logiciels de la partie commande, hors site, de remplacer des manipulations longues et fastidieuses de "boîtes à boutons", tout en améliorant la qualité des tests. Un outil informatisé industriel, ADELAIDE [COR 85], [COR 89] (Emulateur de Partie Opérative) a ainsi été réalisé suivant ce principe. D'autres outils ont été développé sur des bases similaires (PIASTRE [CAZ 83], SIMULA [DER 86], PROSYST [PRO 88]), ...) ;

- Une deuxième utilisation des modèles de comportements d'E.P.O. peut se situer au niveau de la "surveillance" du comportement de la partie opérative d'une installation. Ce modèle appelé "**filtre**" (ou Elément de Commande) [ALA 86] (Figure II.12) est placé entre une Partie Commande fonctionnelle et les cartes interfaces d'entrées/sorties. Il est ainsi possible de détecter :

  . que les ordres reçus de la partie commande sont cohérents avec l'état du modèle d'E.P.O. considéré (et donc avec l'état de la PO),
  . que les retours issus de la PO sont compatibles avec ceux prévus par le modèle.

L'implémentation systématique de "filtres" dans la commande d'installations automatisées améliore la fiabilité et la qualité du système car [SFA 89]:

- il soulage les modules supérieurs de commande de la surveillance ;
- il évite la remise en cause des tests à chaque modification des niveaux de commande supérieurs (puisque la surveillance est propre à l'EPO modélisé et non pas à son utilisation) ;
- il évite la propagation des erreurs.

Les deux utilisations de ce concept peuvent être mêlées pour permettre, en commande d'application, d'offrir la fonction "filtre" et la fonction "émulation" sur la base d'un même modèle de comportement d'EPO.

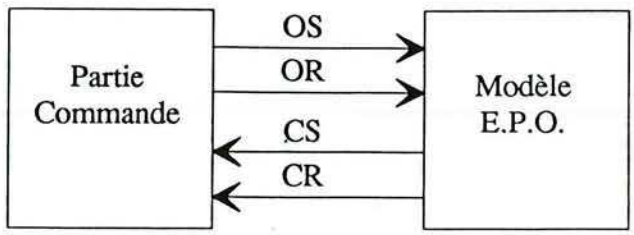


Figure II.11. : Utilisation d'un E.P.O en émulation

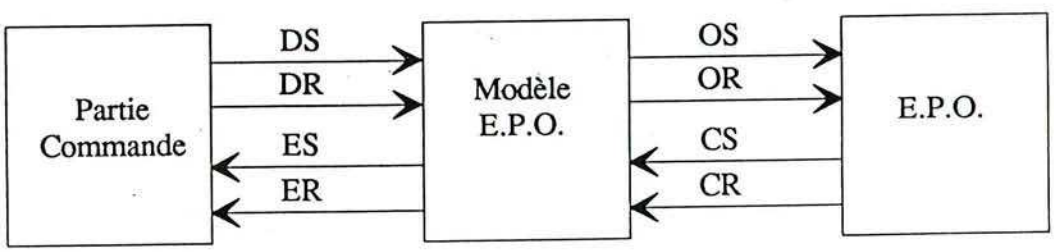


Figure II.12. : Utilisation d'un E.P.O en "filtre"

Une généralisation du concept d'**Elément de Partie Opérative** peut être réalisée à chaque niveau de décision pour la modélisation de comportements fonctionnels. Il doit, en particulier, disposer de "modes de fonctionnements" propres au niveau concerné permettant une reprise correcte d'exploitation après détection et traitement d'un défaut. De plus, la "**mémorisation d'un historique**" des évènements apparus ou de ses états internes successifs devrait permettre une exploitation générale de l'installation ou une maintenance préventive.

Une concrétisation de cette approche de "**généralisation**" et d'extension de la modélisation d'Eléments de Partie Opérative fait l'objet d'un certain nombre de travaux engagés par notre équipe :

- une recherche en collaboration avec les sociétés PEUGEOT et TELEMECANIQUE pour déterminer et implémenter sur la nouvelle gamme d'automates Télémécanique un **OFB** (Option Function Block) (ou un ensemble d'OFB) définissant un **modèle de référence particulier** de la classe d'application "machine transfert" **chez** PEUGEOT. Cet OFB définira donc un comportement "**générique**" pour toutes machines transfert utilisées chez PEUGEOT ;

- Une extension du principe de "filtre" aux **actionneurs intelligents** [CIA 88] est en cours dans le cadre du projet ESPRIT/DIAS [DIA 90] avec EDF ayant déjà permis l'élaboration d'un EPO général pour la classe d'applications "Vannes". D'autres travaux, dans le cadre d'une collaboration avec la Régie RENAULT, dans le projet PROMETHEUS cherchent à appliquer ce principe pour la réalisation de "**capteurs intelligents**" [CIA 87].

2.3. Approche orientée "Objet Produit"

Modèle d'assemblage [BOUa 90]

L'assemblage représente, dans la production industrielle d'objets manufacturés, une part de plus en plus importante des coûts globaux de fabrication. La fabrication de type multi-produit est aujourd'hui une réalité que la production doit prendre en compte. Les ateliers d'assemblage doivent être en mesure de s'adapter continuellement à des modifications, sans pour autant avoir été conçus en intégrant cette faculté. [BOUa 90] propose une représentation hiérarchisée des tâches d'assemblage, basée sur un concept unificateur d'"**objet industriel**" représentant toute composante nécessaire au processus d'assemblage. Une tâche d'assemblage est formalisée par différents niveaux d'abstraction dans la représentation qui permet le passage d'une description purement fonctionnelle à une description technologique ou inversement, et de ne tenir compte que des

effets sur le produit sans préjuger de l'équipement mis en oeuvre pour assurer cette transformation.

Un "objet industriel" est caractérisé par un état traduisant :

- sa structure géométrique, c'est à dire l'ensemble des liaisons géométriques entre les composants que l'on peut décrire par un graphe de composition (Figure II.13) dont les sommets représentent les composants et les arcs les liaisons géométriques correspondantes ;
- sa structure physique mettant en oeuvre des transformations physiques portant sur les composants.

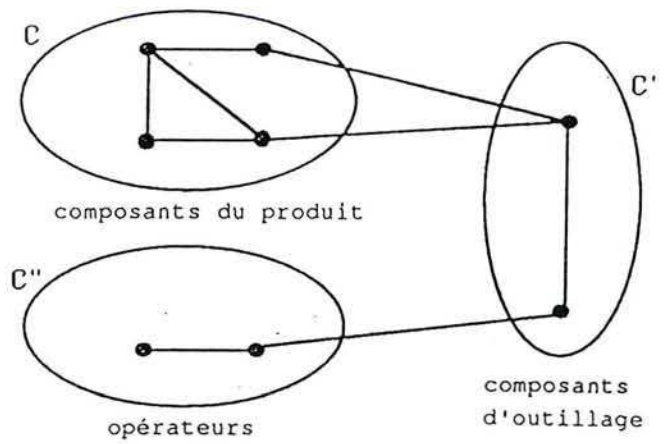


Figure II.13. : Le graphe de composition

La tâche de fabrication peut être décrite par un **schéma opératoire** (Figure II.14) décrivant l'ensemble des transformations (positionnelles, géométriques, structurelle) opérées sur des composants.

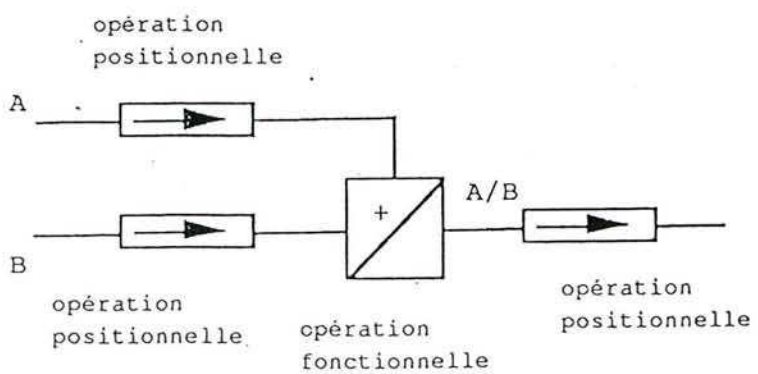


Figure II.14. : Le schéma opératoire

Cette modélisation devrait permettre, à partir d'un produit à assembler et d'un ensemble de contraintes associées, de structurer le processus d'assemblage et founir une aide à l'identification d'une telle architecture.

2.4. Autres Approches

Approche fonctionnelle

[AMA 90] propose une **approche fonctionnelle** qui consiste, par une démarche de décomposition descendante, à identifier les fonctions assurées par le système. Après identification des différents Eléments de Partie Opérative mis en jeu, une démarche ascendante doit permettre d'associer, à chaque fonction identifiée, un EPO particulier. L'ensemble des couples Fonction-EPO définit alors l'architecture structuro-fonctionnelle (Figure II.15) de l'installation réalisant un modèle de référence partiel que l'on peut instancier pour une installation particulière.

Cette démarche propose ainsi la définition de **classes fonctionnelles** génériques qui, associée à des modèles de comportement de la partie opérative, offre, au concepteur, toutes libertés pour la conception de modèles à caractères relationnels à partir d'une base de composants réutilisables ayant un fort degré sémantique. Elle fournit donc une aide méthodique pour la création de la commande d'un Système Automatisé de Production.

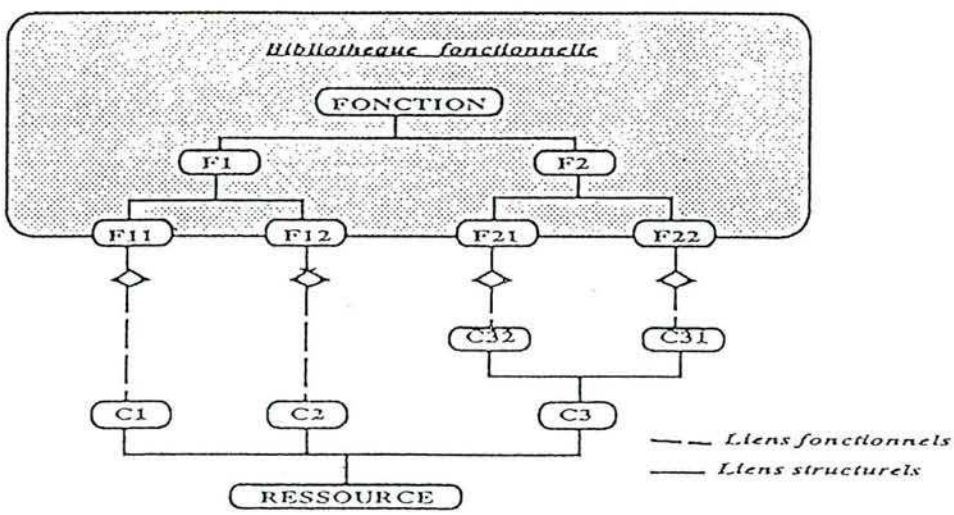


Figure II.15. : Spécification structuro-fonctionnelle

Approche liguistique

Nous pouvons considérer le texte narratif du fonctionnement d'une installation comme un modèle pratique de cette installation. Des voies de recherches, engagées à ce jour, conduisent cependant à s'interroger sur les parallèles existants entre nos démarches et solutions du **Génie Automatique** et celles issues des domaines **linguistiques** [COU 76] pour la définition d'un modèle cognitif général.

Il est vrai que parfois la similitude est frappante. La classification des récits n'a pu se faire qu'après bien des efforts et avec la prise de conscience que la recherche d'éléments **génériques** ne pourrait se faire que dans les formes (structures) et non pas dans les contenus. Il fallait donc pour cela établir une "connaissance" sur la "connaissance" et voir comment se bâtit une idée autour d'une association d'unités élémentaires de sens (étude sémiotique) [COU 76].

De même, la connaissance liée à l'automatisation d'un processus gagnerait grandement à être structurée dans une **Base de Connaissances** commune à tous les intervenants. Cette structure de connaissance devrait faire apparaître trois classes qui se mixeraient dans le programme résultant, support final du message entre le concepteur et sa machine. Citons par analogie au récit [MAR 90], la classe des manifestations actorielles qui expriment les constituants physiques (acteurs) de la machine au travers d'énoncés d'états ; la classe des manifestations actancielles qui révèlent les conditions spécifiques à la production d'un acte "standardisé" avec ses énoncés de transformation ; la classe des manifestations dites "profondes" qui prend en compte les objectifs de l'histoire (du fonctionnement).

Comment ne pas penser alors respectivement aux **Eléments de Partie Opérative**, aux **Activités** issues des analyses fonctionnelles (SADT/IDEF0) et à des modèles d'**Actinomies** ?

II. CONCLUSION DU CHAPITRE II

Les apports des concepts liés au **Génie Logiciel** sont indéniables mais ne suffisent pas pour résoudre tous les problèmes du **Génie Automatique**. Il ne faut pas oublier que le domaine de l'automatisation manipule des objets logiciels ayant un lien **indissociable** avec des objets matériels et qu'une démarche d'Automatisation Intégrée doit être supportée par des outils-méthodes riches d'un point de vue "**sémantique**".

La **réutilisation** de "composants génériques" apporte des solutions à la productivité et à la qualité du développement de systèmes automatisés mais, si des composants technologiques, très bien identifiés, sont modélisables, il reste encore beaucoup à faire pour construire de tels composants à des niveaux fonctionnels et décisionnels. Les approches sémiotiques apporteront peut être des solutions mais restent encore au stade de la recherche.

Le support informatique des différents modèles cognitifs est réalisable dans l'outil SPEX :

- les modèles d'Eléments de Partie Opérative ont déjà été mis en oeuvre dans des applications pour la définition de comportements génériques utilisables tant en "filtre" qu'en "émulation de l'environnement" ;

- l'implémentation du modèle "objet/porte-objet" semble réalisable dans l'outil ainsi que l'interfaçage avec PROPEL ;

- l'approche fonctionnelle repose pour une part, sur une classification, point non supporté actuellement par SPEX (définition de classes et utilisation du mécanisme d'héritage (cf Chapitre I, § I.3.4)).

SOMMAIRE
Chapitre III

I. DE L'INTERCONNEXION D'OUTILS ...

SPEX est un outil du **Génie Automatique** qui doit pouvoir s'intégrer au sein d'une structure d'accueil plus générale constituant un **Atelier de Génie Automatique**. Pour ce faire, il est nécessaire de montrer qu'il peut être "interfacé", en amont, avec des outils-méthodes d'analyse pour la spécification et la conception générale de l'automatisme et, en aval, avec des générateurs automatiques de code et des outils de tests hors-site de la partie opérative. La Figure III.1 montre la place de SPEX et des outils amont et aval dans un cycle de vie d'automatisation [CCG 87] telle qu'elle a été proposée dans la réponse à l'appel d'offres du MIPTT/SERICS [MIP 87].

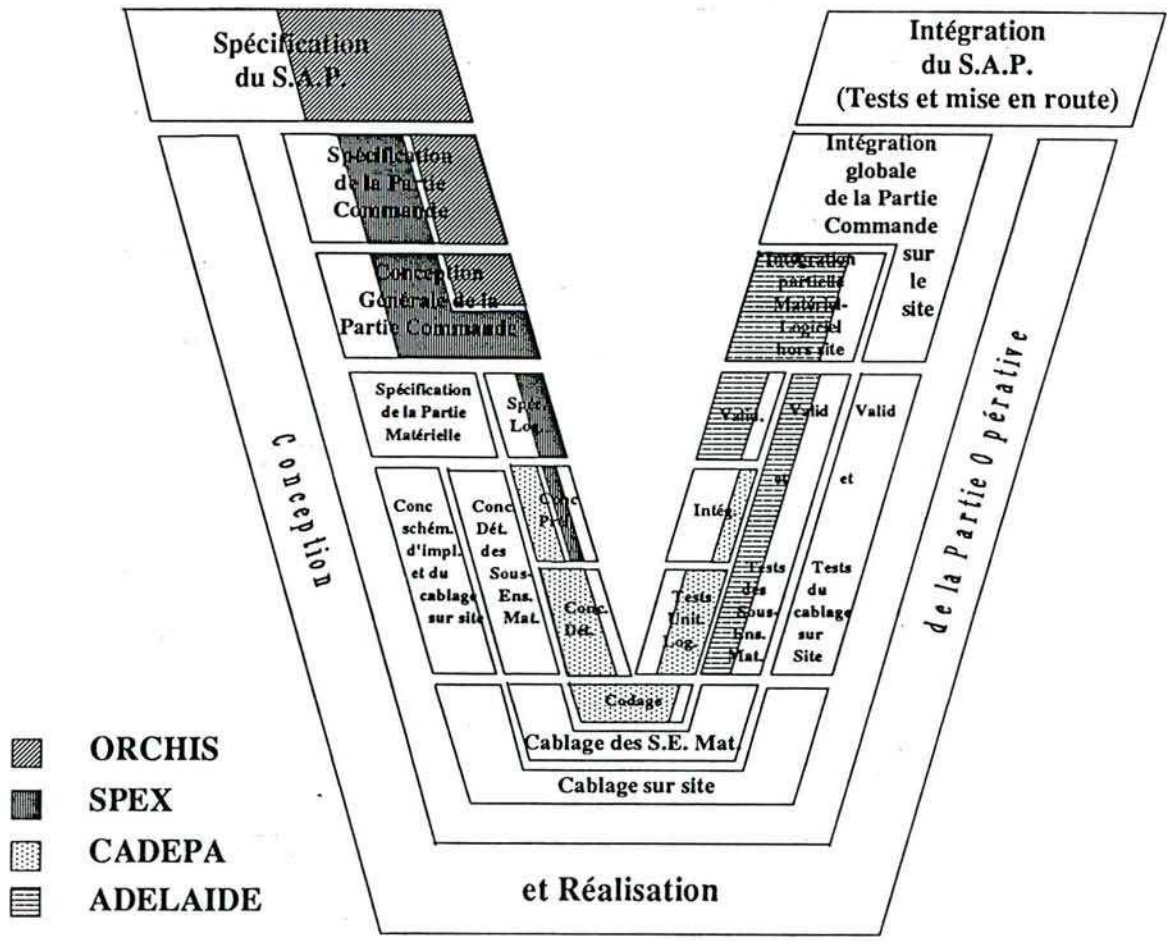


Figure III.1. : Les outils dans le cycle de vie d'automatisation

Nous présentons, dans ce chapitre, des exemples d'interfaçage amont avec une analyse de type SADT/IDEF0 [IGL 89], supportée par l'outil Orchis [TNI 89]. Ces interfaçages correspondent, soit à un **passage direct** d'un modèle SADT à un modèle SPEX, soit à une **transformation progressive** d'un modèle SADT par l'application d'un **modèle de référence particulier** comme, par exemple celui présenté en [LHO 85] puis, une **traduction** de l'architecture identifiée en composants manipulables par SPEX.

Nous montrons, ensuite, une faisabilité pour la connexion de SPEX avec le générateur de code pour automates, CADEPA [SGN 90].

SPEX étant indépendant de l'architecture cible d'exécution des comportements, il présuppose une installation mono-processeur. L'aspect multi-processeurs nécessaire pour des applications complexes impose ou bien l'intégration d'une fonction de **répartition organique** à SPEX, ou bien, la création d'un nouvel outil de conception organique intégrant les fonctionnalités de simulation dérivées de SPEX.

Nous exposerons une étude de faisabilité concernant l'informatisation d'un tel outil sur les bases de SPEX.

La connexion de SPEX à l'émulateur de Partie Opérative MAXSIM [MAX 91] termine la présentation de cette chaîne d'outils.

1. Interfaçage d'outils existants autour de SPEX

1.1. Vers les phases amont du cycle de vie

1. Correspondance directe entre IDEF0 et SPEX

La correspondance entre un modèle SADT (structure statique fonctionnelle) et un modèle comportemental (structure dynamique) peut être réalisée par trois types d'approche non exclusive :

- l'approche **traduction** correspond à la définition de règles permettant une traduction directe des différents symbolismes employés dans un modèle SADT pour en déduire une structure équivalente dans un autre formalisme. Le résultat de cette traduction peut ensuite être utilisé pour l'enrichissement du modèle d'un point de vue dynamique (point de vue non supporté par SADT). DESIGN-IDEF [ISD 90], outil support de SADT et

EDDA [BRO 88], traducteur SADT-Pétri sont deux outils qui, en relation, permettent la traduction d'un modèle SADT en un réseau de Pétri Coloré que l'on peut ensuite interpréter en simulation, moyennant l'association d'un marquage et d'une interprétation adéquats.

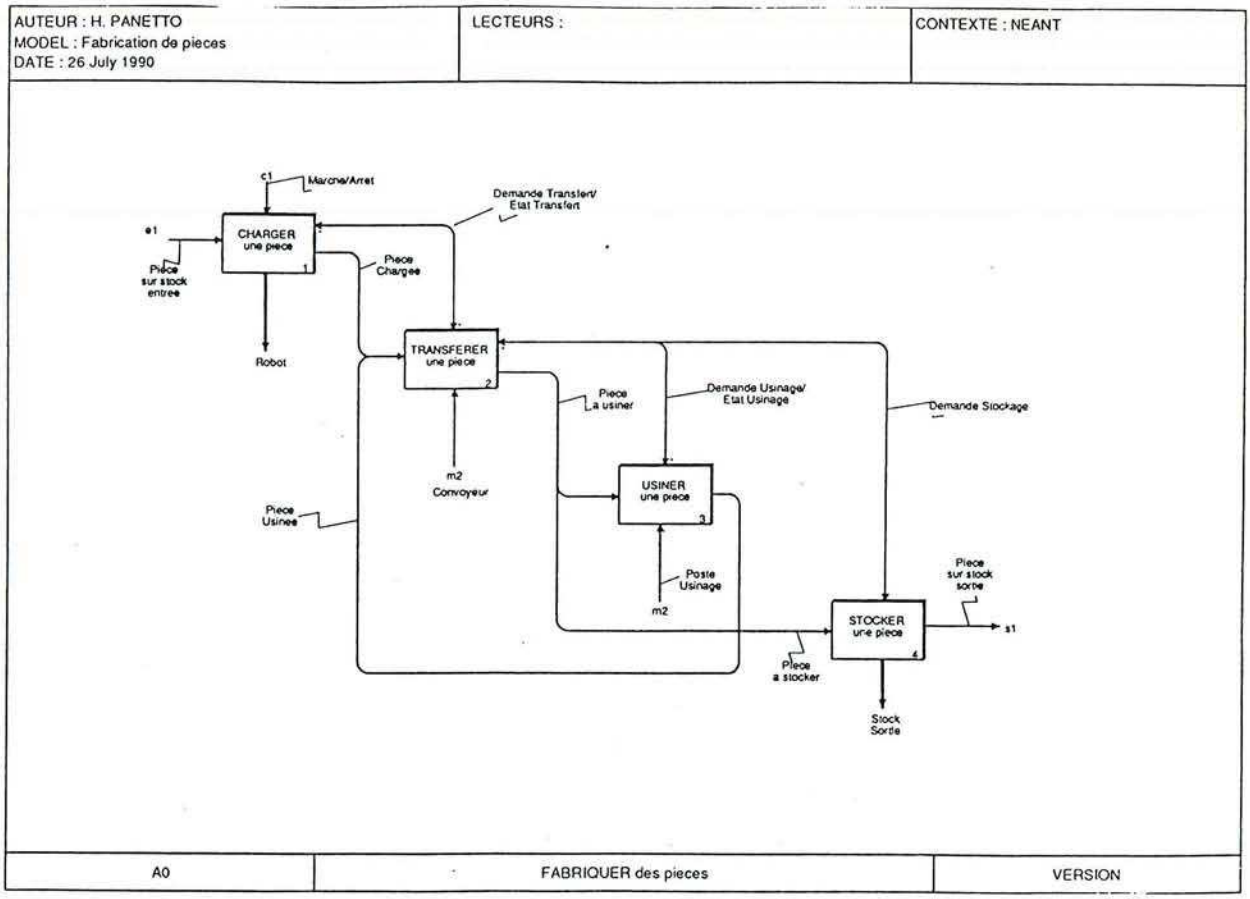
- l'approche **transformation** cherche à appliquer, de façon progressive, un **modèle de référence particulier** induisant une certaine méthodologie d'utilisation de SADT. Ce modèle de référence permet l'enrichissement du modèle par la définition d'activités spécifiques à un domaine d'application particulier.
- l'approche **association** conserve la structure hiérarchique des activités fournie par le modèle SADT et associe, en fin de décomposition, un modèle dynamique à l'ensemble des boîtes "feuilles" de la hiérarchie. ASA [ASA 88] est un outil permettant l'association, pour chaque activité "feuille" d'une décomposition, d'un comportement dynamique décrit par un automate à états finis, ceci en vue de la validation du comportement global de l'application par simulation.

Nous présentons ici, notre approche pour une **traduction** directe entre le modèle SADT et les composants de base de SPEX qui sont les **Boîtes Fonctionnelles** (BF) et les **Diagrammes Fonctionnels** (DF) décrivant un comportement (donc une dynamique).

Cette traduction doit respecter quelques règles de transposition :

- Un diagramme SADT correspond à un exemplaire de DF nommé par le nom de l'activité SADT ;
- A une activité terminale (feuille) correspond, en fonction des choix de décomposition ultérieurs ou de la disponibilité d'un comportement générique adéquat, un DF vide ou une BF vide ou un exemplaire de BF générique nommé par le label associé à la flèche d'appel à mécanisme correspondante ;
- Chaque flèche doit être détaillée pour définir les entrées des "Boîtes" (flèches en entrée et contrôle de SADT) et les sorties (flèches en sorties de SADT). Une flèche non détaillée ne sera pas importée dans SPEX (flèches ayant une sémantique "produit" par exemple).

Le réseau fléché du modèle SADT induit ainsi les relations entre modèles de comportements instanciés d'une application SPEX (Figure III.2).



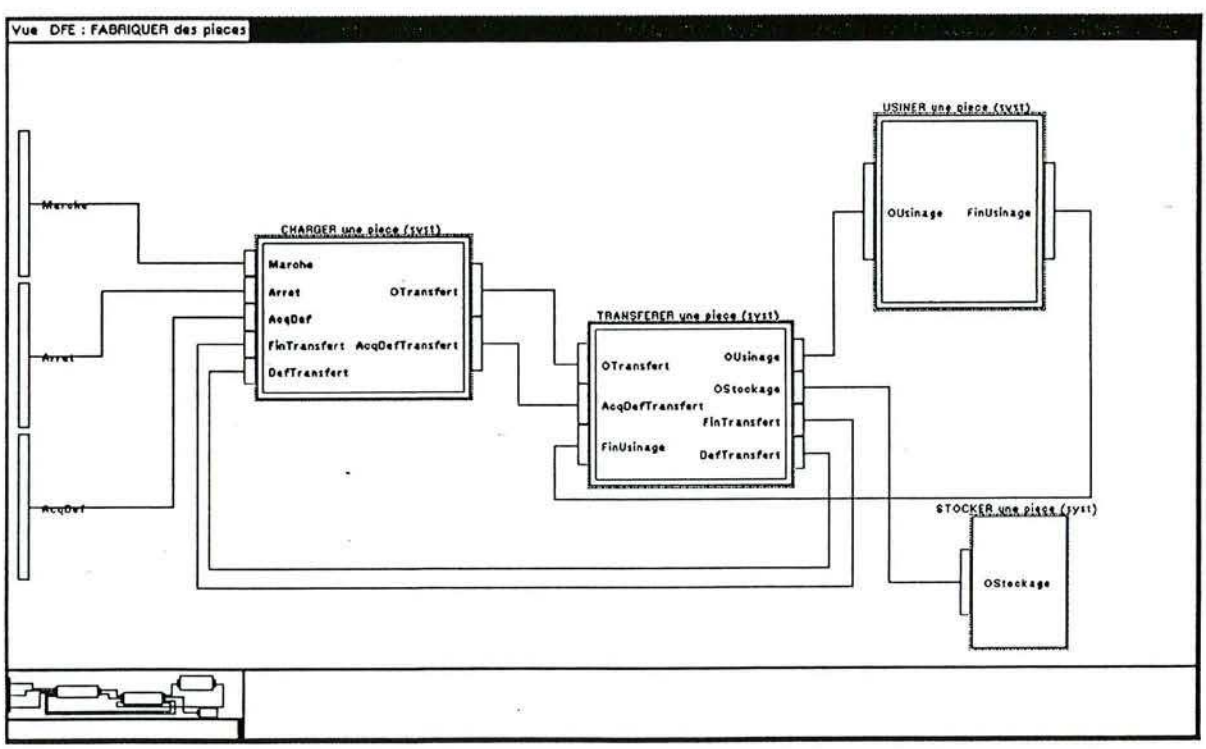


Figure III.2. : Correspondance entre un modèle SADT et un modèle SPEX

Ce passage d'un modèle SADT à un modèle SPEX admet cependant une limite concernant les différentes interprétations explicites.

En effet, la simulation appliquée à la structure comportementale dans SPEX est **synchrone**, alors que le schéma de pensée pour l'élaboration d'un diagramme SADT relève plutôt d'une interprétation de type "**asynchrone**". Il est donc nécessaire de tenir compte de cette interprétation lors de la construction du modèle SADT si l'on désire, ensuite, le traduire pour une application SPEX.

2. Informatisation d'un modèle de référence particulier de Conception des MSAP

2.1. Position du problème

Une décomposition hiérarchique d'un système par une méthode de type SADT/IDEF0 laisse au concepteur **toutes libertés** dans ses choix fonctionnels et dans les relations qui vont intervenir entre les différentes activités mises en oeuvre. Cette "liberté d'expression" favorise certainement son esprit créatif mais induit par ailleurs une certaine **subjectivité** dans l'interprétation de ses besoins réels. En effet, aucune démarche intellectuelle n'étant imposée par la méthode, l'ensemble de la commande de l'automatisme sera dispersée dans la structure fonctionnelle au gré des besoins de *coopération et/ou coordination* entre activités apparaissant en cours d'analyse.

Il s'avère en effet que l'utilisation conjointe de la coopération entre processus et la coordination par des fonctions de contrôle laisse apparaître une flexibilité *non maitrisée* alors que la restriction à une description coopérante des différents processus ne permet pas une analyse de tous les cas de figure [NIV 89].

Toute méthode a pour but de structurer la pensée de l'utilisateur et doit l'amener à se poser les "**bonnes**" questions au "**bon**" moment mais plus elle est flexible, plus il devient difficile d'orienter les choix dans la bonne direction.

La réutilisation et l'enrichissement d'un savoir-faire antérieur ne peuvent être entrepris que par la formalisation d'un **guide de conception** imposant un "fil conducteur" dans l'organisation de la modélisation et, par une standardisation de règles d'utilisation d'une méthode indépendamment de l'application concernée.

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

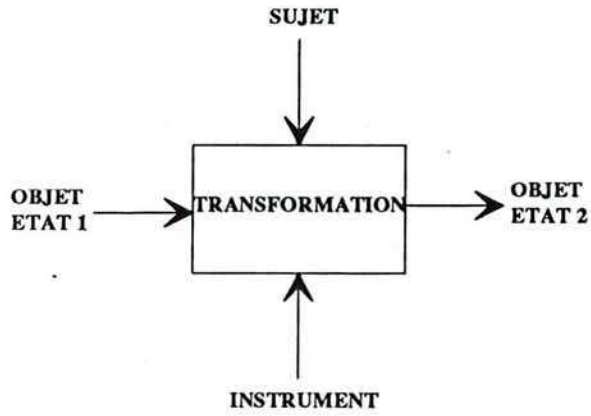


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

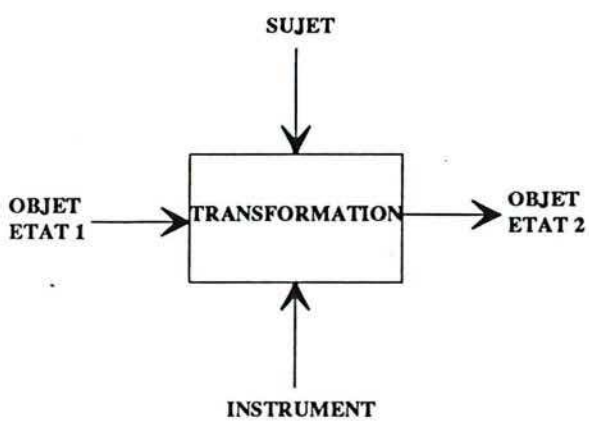


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

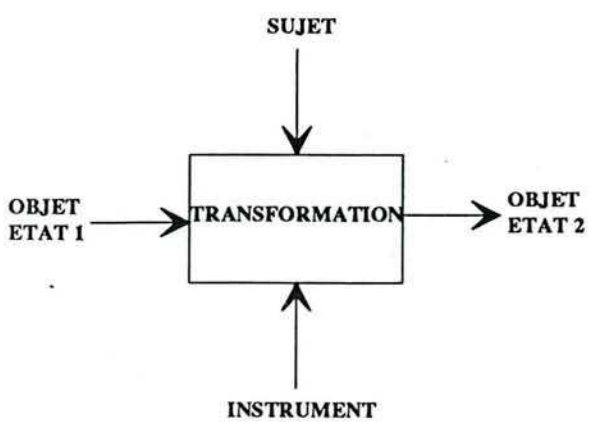


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

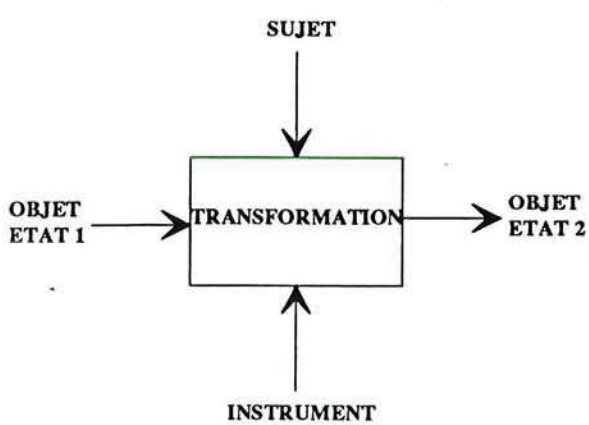


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

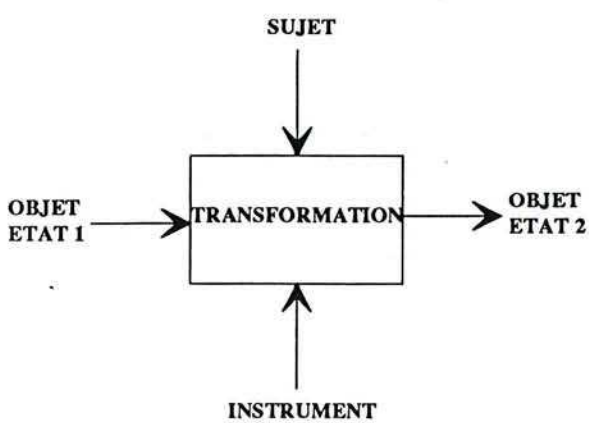


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

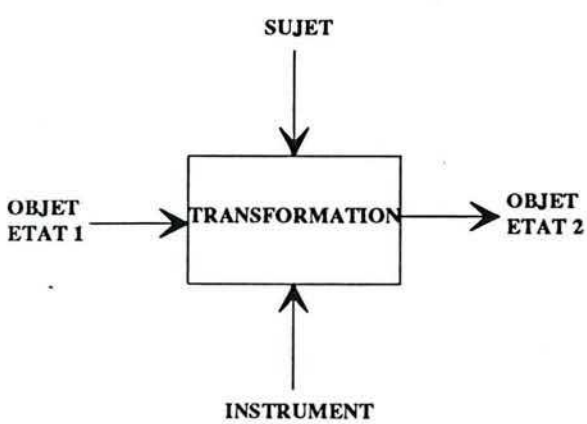


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

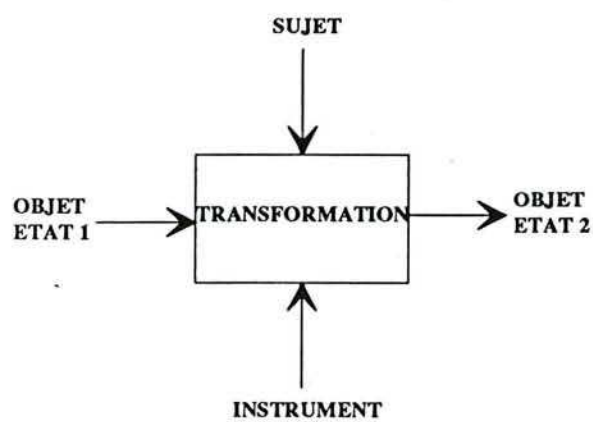


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

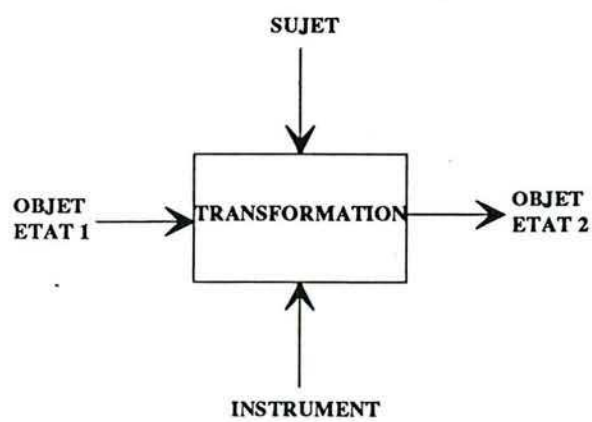


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

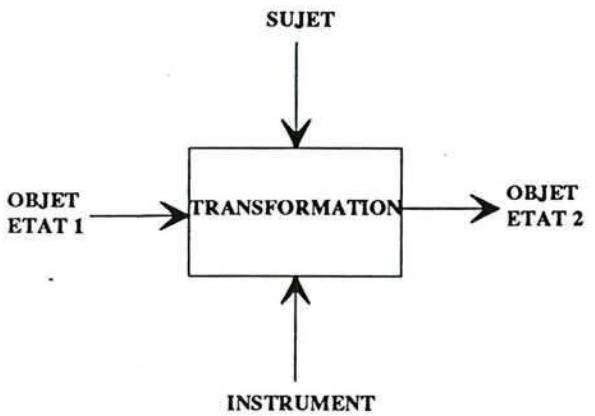


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

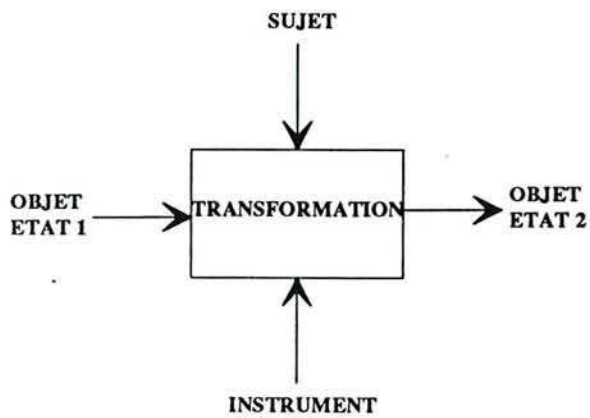


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

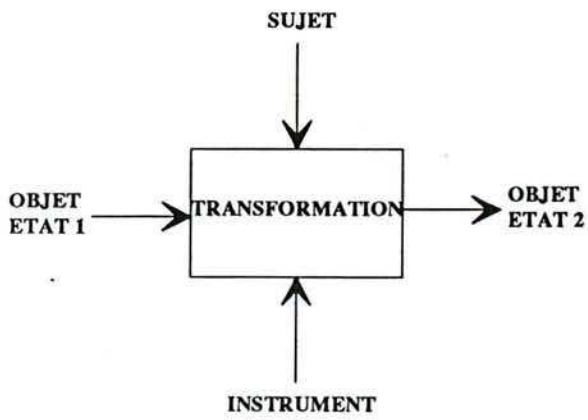


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

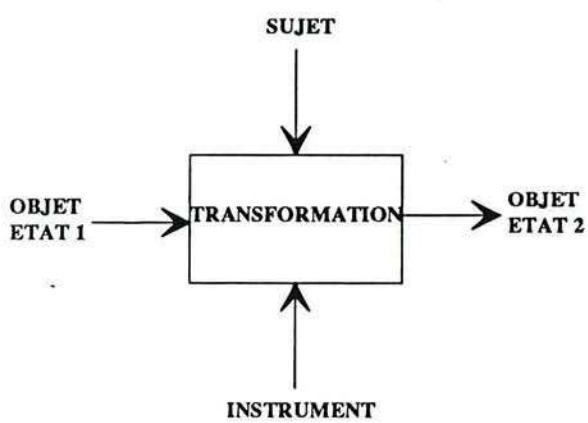


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

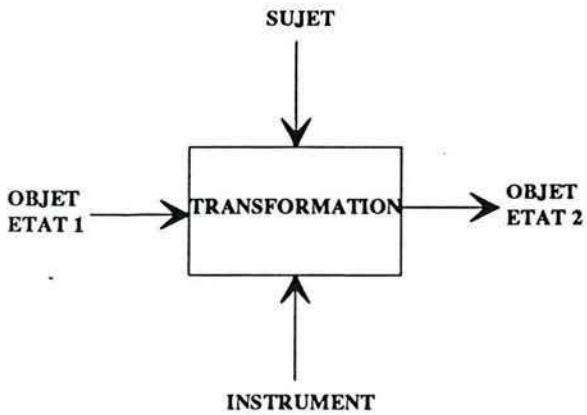


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

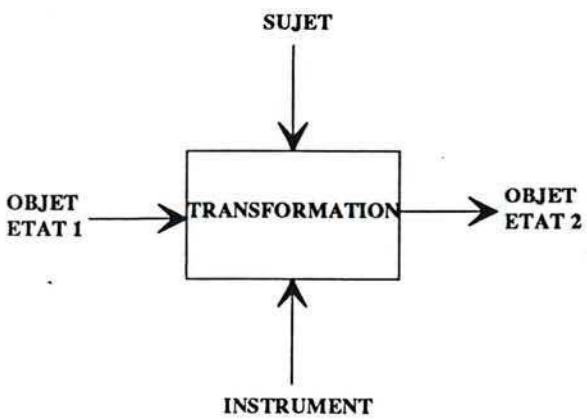


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

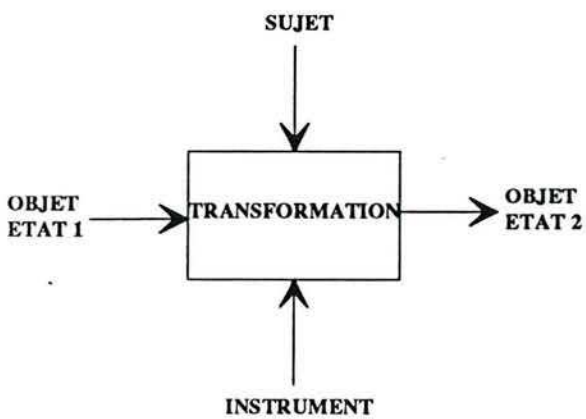


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

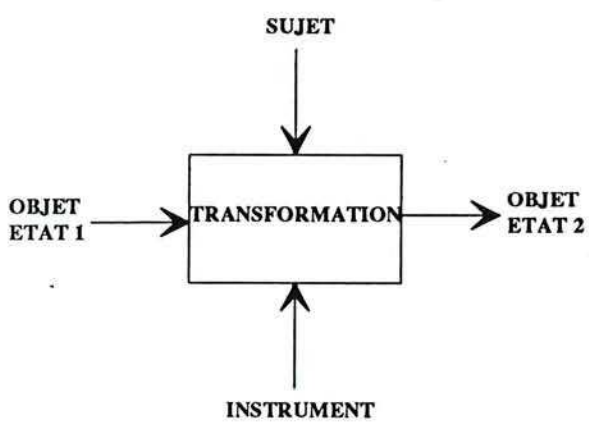


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

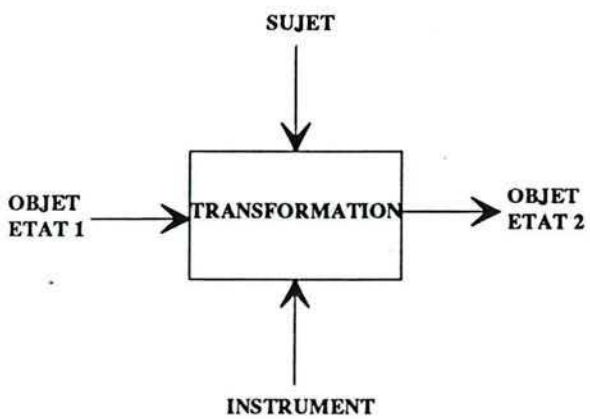


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

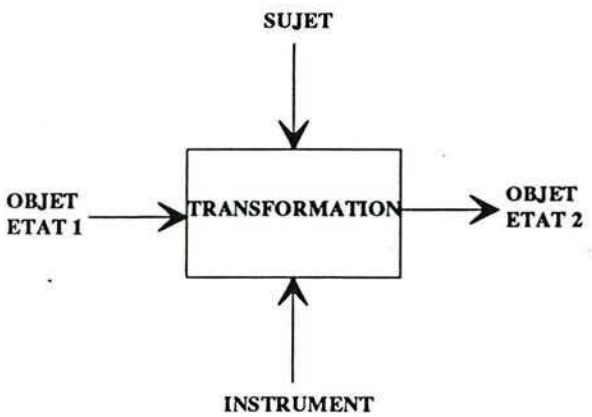


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

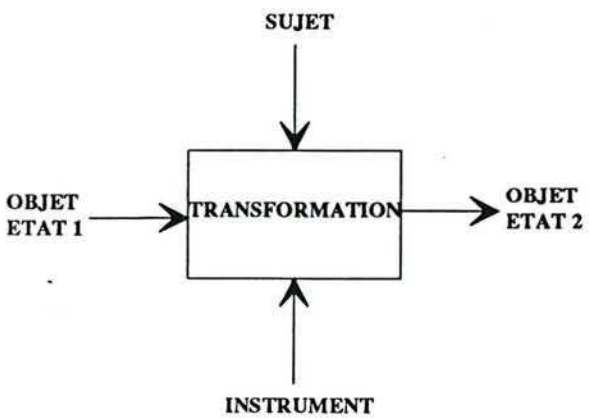


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

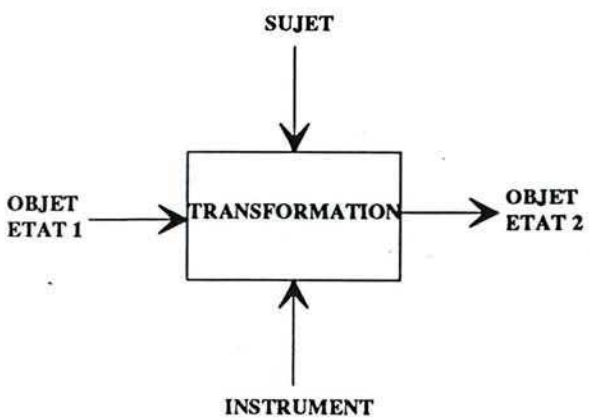


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

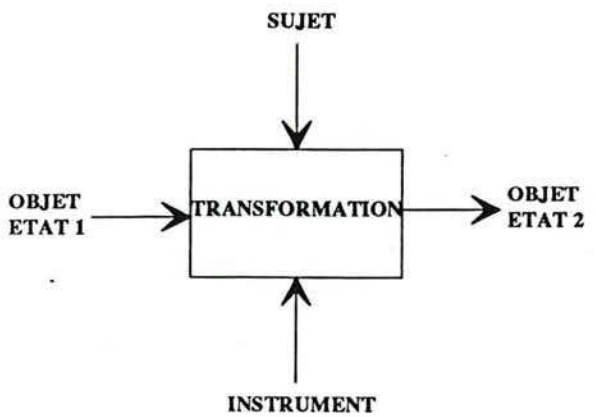


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

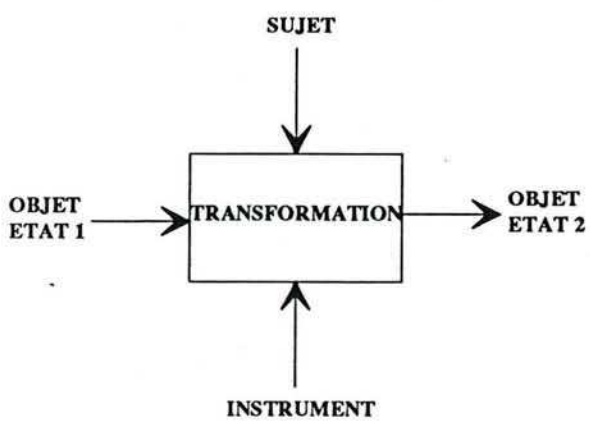


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

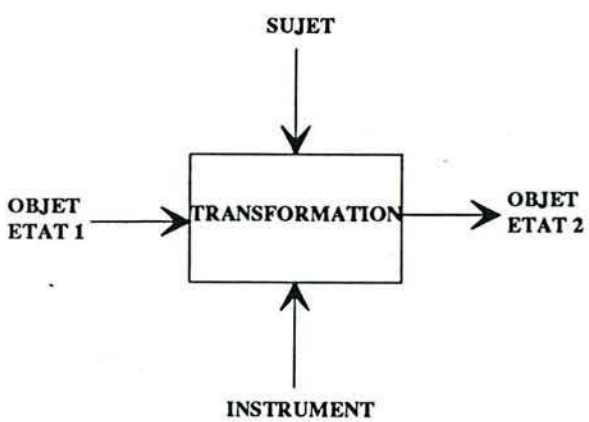


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

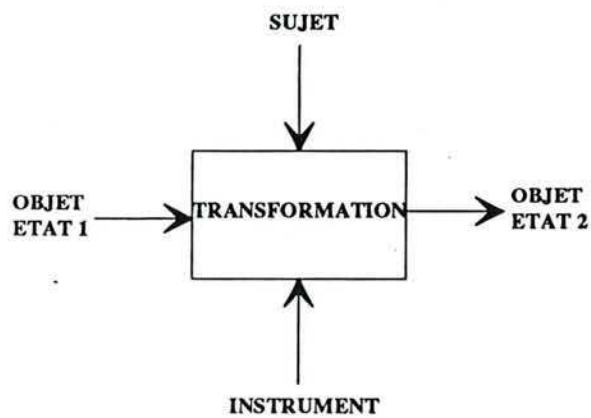


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

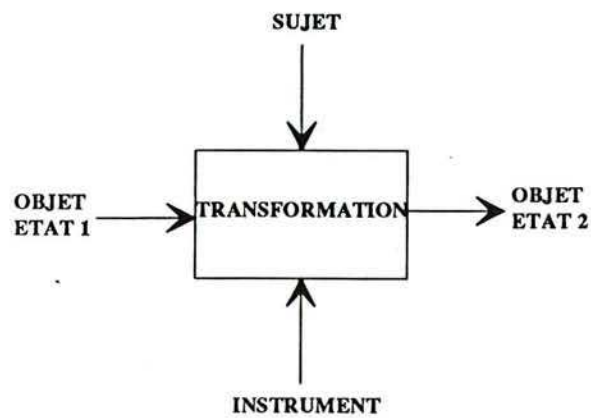


Figure III.3. : L'actème [VOG 88]

Cette nécessité implique, en outre, une mémorisation des composants répétitifs ou réutilisables mais aussi une conservation (après formalisation) de la (ou des) démarche(s) les utilisant et ayant déjà permis de passer d'un problème à sa solution.

2.2. SADT/IDEF0 en spécification de conception

La spécification de conception d'un système tend à répondre à la question **QUOI**, c'est à dire décrire ce que l'on veut faire sans en détailler **COMMENT** le réaliser [IGL 89]. Une analyse fonctionnelle par la méthode SADT/IDEF0 permet une spécification hiérarchique des fonctions du système en modélisant les changements d'état des objets ainsi que les ressources utilisées mais les rôles de chaque fonction sont attribués de façon très laxistes, et le manque de sémantique sous-jacente rend peu à peu impossible le contrôle et la critique des modèles ainsi construits.

L'utilisation de SADT/IDEF0 peut être complétée par l'association de **règles de lecture** apportant une sémantique à la modélisation compatible avec le domaine concerné. [VOG 88] propose un ensemble de règles de lecture permettant une spécification par un modèle pratique à base d'**actèmes** (Figure III.3).

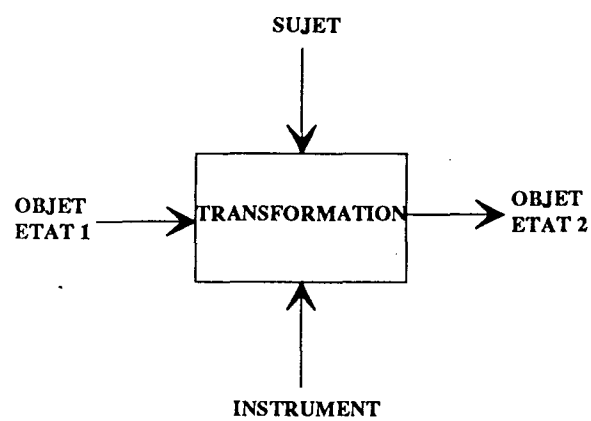


Figure III.3. : L'actème [VOG 88]

Le modèle de l'actème est centré sur le changement d'état (état physique, état informatif, état positionnel, ...) de l'objet rentrant et sur un "typage" sémantique des flèches en entrée, sortie, contrôle et mécanisme. Le **sujet** *transforme* l'**objet** de l'*état avant transformation* à l'*état après transformation* à l'aide de l'**instrument**. La succession des actions se fait sur la diagonale du schéma et représente l'**actinomie** du niveau concerné.

[LHO 85] utilise SADT pour la description des processus intervenant dans l'automatisation des systèmes de production par une approche Contrôle-Commande qui privilégie la coordination de tout sous-processus d'un niveau donné par une **fonction de contrôle**. Celle-ci détermine, en particulier, le séquencement des sous-processus concernés et centralise toutes les informations issues de ces mêmes sous-processus, y compris celles de type "coopérant". Les sous-processus n'ont de relations explicites entre eux que par les flux de produits transformés.

2.3. L'approche méthodologique pour la conception de M.S.A.P.

La démarche de conception

Cette approche méthodologique de conception propose une démarche descendante basée sur de la méthode SADT/IDEF0 et associée à un **guide de conception** précisant les activités (ou **processus**) dont l'automatiseur peut avoir à tenir compte dans son étude. Cette approche "guide de conception" fournit au concepteur un "**canevas**" type pour conduire son analyse et permet une "**standardisation**" des règles de conception au sein d'une entreprise. La contrainte ainsi induite favorise la communication entre les différents personnels amenés à intervenir pour la conception, la réalisation et la maintenance de l'automatisation.

Cette méthodologie doit proposer des règles de description autour d'un **méta-modèle "métier"** où les différentes activités sont bien déterminées et non ambigües pour le domaine concerné.

Cette démarche peut être reliée aux approches **systémiques** [DUR 90] [LEM 84] qui se proposent de décrire un système dans sa **totalité** comme un ensemble de sous-systèmes (ou activités) dont les interactions induisent une

organisation fonctionnelle et structurelle. Cette organisation conduit à la définition de l'objectif de la modélisation en termes de finalité, relations avec l'environnement, définition des activités et de la structure d'ensemble, évolution du système, indépendamment de la réalisation effective des éléments qui la composent.

1. Le "guide de conception" proposé

1ère étape : Décomposition d'un processus en sous-processus

La décomposisiton du processus en sous-processus, première étape de la méthode, doit permettre de définir progressivement les différentes fonctions que doit réaliser le processus. Pour le domaine qui nous concerne, c'est à dire l'industrie manufacturière, les "fonctions" ou "sous-processus" correspondent à des actions de transformation sur le produit que l'on peut classer en :

- **Fonctions de Transformation Positionnelle** : la transformation consiste dans ce cas à faire évoluer la position du produit ou encore fixer cette position (ex : manutention, chargement, positionnement, ...) ;

- **Fonctions de Transformation d'Etat** : la transformation est alors une modification des caractéristiques physiques du produit (ex : usinage, assemblage, ...) ;

- **Fonctions de Transformation Informationnelle** : dans ce cas, le produit est considéré comme porteur d'informations et ces fonctions n'agissent plus sur le produit directement mais sur l'Information qu'il porte avec un degré plus ou moins élevé (ex : mesure, pesage, détection de passage ou de présence, ...) ;

- **Fonctions "Annexes"** : ces fonctions n'agissent pas directement sur le produit mais contribuent à la réalisation des fonctions précédemment citées (ex : arroser pendant l'usinage, descendre une broche porte-outil, ...).

Un processus sera donc décrit par une collection de fonctions "sous-processus" auxquelles on associera systématiquement une **fonction de "Contrôle-Commande"** qui représentera la partie décisionnelle du processus ainsi que les interractions de nature informationnelle devant exister entre sous-processus pour assurer, dans un cadre systémique, non seulement leur "survie" (sécurités) mais aussi leur bon fonctionnement (synchronisations). Dans un premier temps, on ne développera pas cette fonction dont la description sera entamée lors de la seconde étape de la méthode. L'intérêt de cette démarche est de favoriser, dès le début de l'analyse, une structuration des fonctions de "Contrôle" en correspondance avec des niveaux fonctionnels de sous-processus. Un autre intérêt est de distinguer nettement sur un même diagramme les flux de données des flux de ressources (i.e. les produits).

L'existence d'une fonction de Contrôle à chaque niveau de décomposition imposera donc, conformément à S.A.D.T./IDEF0 de décrire un processus par au moins deux et au plus cinq sous-processus (ou regroupements de sous-processus similaires).

Un Sous-Processus sera donc représenté par une "Boîte" SADT à laquelle seront affectés :

- un VERBE représentant l'activité du sous-processus,
- une ou plusieurs flèches sur le coté gauche de la boîte, labellée par un nom, représentant le flux de produit en ENTREE du sous-processus,
- une ou des flèches sur le coté droit, labellée par un nom, représentant le flux de produit en SORTIE du sous-processus,
- une ou des flèches sur le coté supérieur de la boîte, labellée par un nom, représentant le flux d'informations en ENTREE et SORTIE du sous-processus (on utilisera la syntaxe "flèche bidirectionnelle" de SADT),
- une flèche sur le coté inférieur de la boîte, labellée par un nom, représentant le support du sous-processus.

Une fonction de Contrôle sera décrite par (Figure III.4):

- une boîte SADT, labellée par le verbe "CONTROLER" suivi du nom du processus que l'on décrit,
- une sur le coté supérieur de la boîte, labellée par un nom, représentant les informations provenant de la fonction de contrôle du niveau immédiatement supérieur,
- une flèche sur le coté droit de la boîte, labellée par un nom, représentant le retour d'informations vers de la fonction de contrôle du niveau immédiatement supérieur,
- une ou des flèches sur le coté droit de la boîte, labellé par un nom, représentant l'interface avec les sous-processus "coordonnés" du processus que l'on décrit (syntaxe "flèche bidirectionnelle").
- une flèche d' "appel à mécanisme" (au sens SADT) sur le coté inférieur, labellée par un nom, représentant le changement de point de vue nécessaire à la décomposition de la fonction de contrôle (cf étape 2 de la méthode).

Ce type de représentation impose que :

- les seules relations explicites entre sous-processus sont des flux de ressources (produits, énergies, ...) et jamais des flux d'informations,
- les seules relations entre fonctions de contrôle et sous-processus sont des flux d'informations,
- chaque sous-processus pouvant être décrit selon le même modèle, les seules relations entre fonctions de contrôle de niveaux successifs sont des flux d'informations,
- outre les relations entre fonctions de contrôle à des niveaux successifs, un sous-processus ou une fonction de contrôle d'un niveau ne peut pas être directement en relation avec un sous-processus d'un autre niveau.

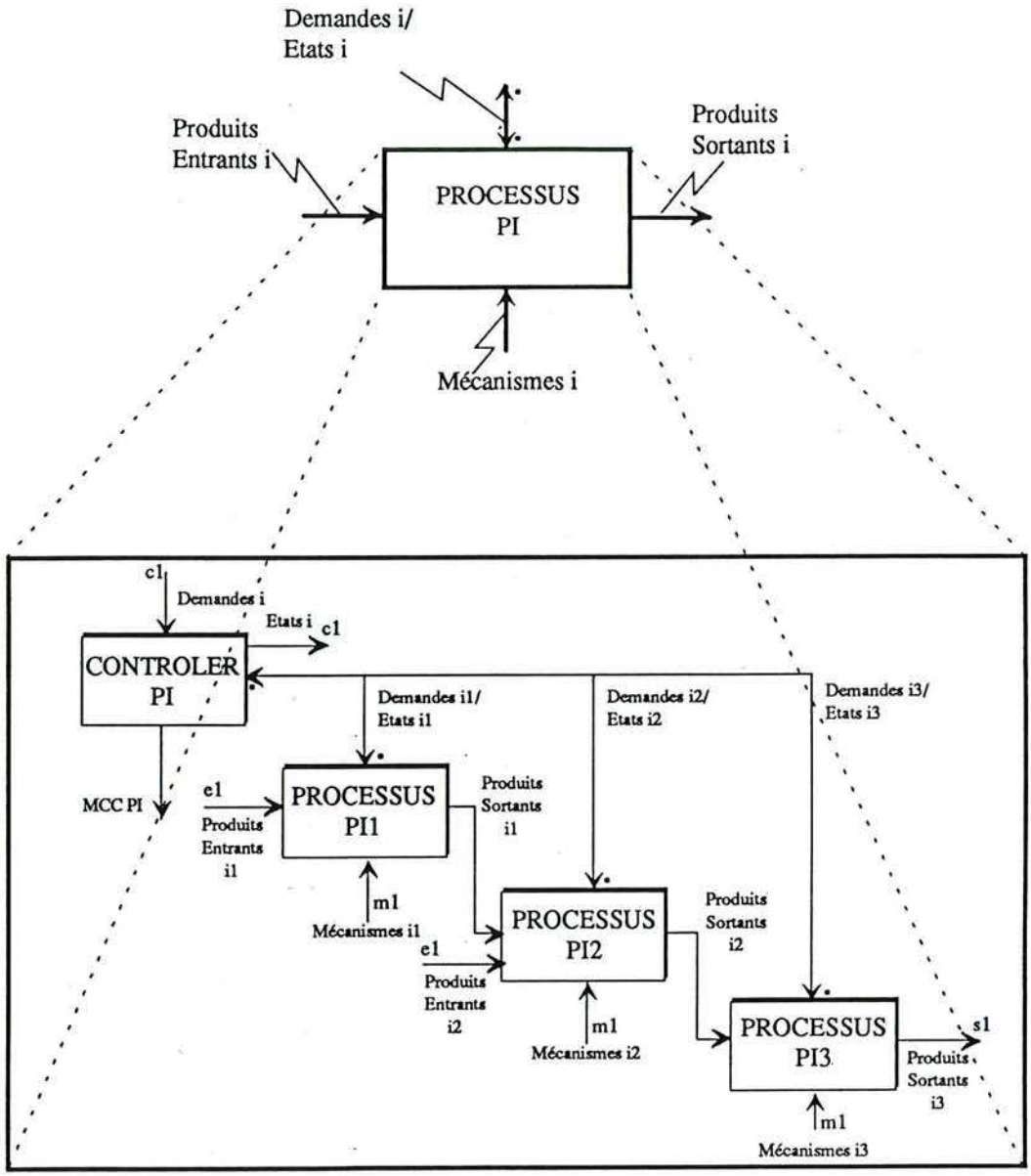


Figure III.4. : La décomposition d'un processus

La décomposition respectant ces règles est répétée autant de fois que nécessaire en considérant qu'un sous-processus est un processus et donc qu'il peut être décomposé de la même manière. La décomposition doit s'arrêter lorsqu'un niveau de détail supplémentaire amène à tenir compte de la technologie d'actionnement associée à la réalisation de la fonction (passage du "**QUOI**" au "**COMMENT**").

Dans ce cas, on considère que le point de vue sur le problème est différent et qu'il faut donc construire un autre modèle représentant cette fois l'aspect "réalisation technologique" de la fonction. On exprime cet arrêt de décomposition d'un sous-processus en associant à la boîte SADT qui le représente une flèche sur le coté inférieur de la boîte en syntaxe d' "appel à mécanisme" avec un label faisant référence à un autre modèle représentant donc l'actionnement du sous-processus élémentaire (Figure III.5) (notion de réseau de modèles en SADT).

Cette description des mécanismes technologiques, encore appelée "**mécanigramme**" ou "**diagramme de mécanisme**", doit faire apparaître des fonctions typées (Figure III.6) :

- comme précédemment, une fonction de contrôle (dont le label sera "**CONTROLER**" suivi du nom du sous-processus élémentaire auquel elle se réfère) jouant un **rôle d'interface logique** entre la fonction à réaliser et la technologie d'actionnement utilisée (cette fonction de contrôle hérite de la totalité de l'environnement informationnel du sous-processus auquel elle est associée) ;

- une activité dont le label est "**PREACTIONNER**" suivi du nom du sous-processus concerné. La boîte SADT associée dispose en entrée, de flèches représentant les ressources énergétiques, en sortie, de flèches représentant la distribution de ces énergies et/ou des informations représentant l'état du préactionneur et retournées à la fonction de contrôle, en contrainte, de flèches représentant la "commande élémentaire" du préactionneur par la fonction de contrôle, en mécanisme (ou plutôt appel à mécanisme) une flèche labellée par le nom du préactionneur (type, repère dans l'installation, ...) ou une référence à une notice technique du préactionneur par exemple ;

- une activité dont le label est "**ACTIONNER**" suivi du nom du sous-processus. La boîte SADT associée dispose en contrainte, de flèches représentant le potentiel énergétique délivré par l'activité "Préactionner", en sortie, de flèches représentant les grandeurs physiques produites (vitesse d'un axe, position d'une tige, ...), en mécanisme (ou plutôt appel mécanisme), le nom de l'actionneur (type, ...) ou une référence à une notice technique de l'actionneur ;

- une activité dont le label est "**OPERER**" suivi du nom du sous-processus. Cette activité joue un **rôle d'interface physique** entre la fonction à réaliser et la technologie d'actionnement utilisée en ce sens où elle seule transforme les produits en entrée du sous-processus en produits sortants. La boîte SADT associée dispose en entrée, de flèches représentant les produits en entrée de la fonction (issus directement des entrées du sous-processus), en sortie, de flèches représentant les produits en sortie de la fonction (issus directement des sorties du sous-processus), en contrainte, de flèches représentant les grandeurs physiques en provenance de l'actionneur contribuant à l'opération, en mécanisme (ou plutôt appel à mécanisme), le nom du module mécanique support ou une référence à une notice technique de cette mécanique ;

- une activité dont le label est "**CAPTER**" suivi du nom de la grandeur mesurée. La boîte SADT associée dispose, en contrainte, de flèches représentant la grandeur mesurée, en sortie, de flèches représentant l'information de mesure de la grandeur physique, en mécanisme (ou plutôt appel à mécanisme), le nom du capteur (type, ...) ou une référence à une notice technique de ce capteur. La grandeur mesurée peut être issue indifféremment (mais en fonction du type de capteur utilisé) des activités "Préactionner", "Actionner" ou "Opérer".

Pour certains mécanismes complexes, il peut être difficile de décrire directement en terme de Préactionneur, Actionneur, Mécanique, Capteur et on admet donc de pouvoir représenter une combinaison de ces éléments par une boîte SADT intitulée "**EXECUTER**" suivi du nom d'une sous-fonction technologique de la fonction à réaliser. En conséquence, cette boîte peut avoir des ressources

(entrées), des contraintes (flèches de contrôle) et des productions (sorties) calquées sur celles des activités "Préactionner", "Actionner", "Opérer" et "Capter".

Ce typage de boîtes SADT est à rapprocher de celui préconisé en enseignement dans les filières techniques T.S.A. [TSA 89] (Technologie des Systèmes Automatisés).

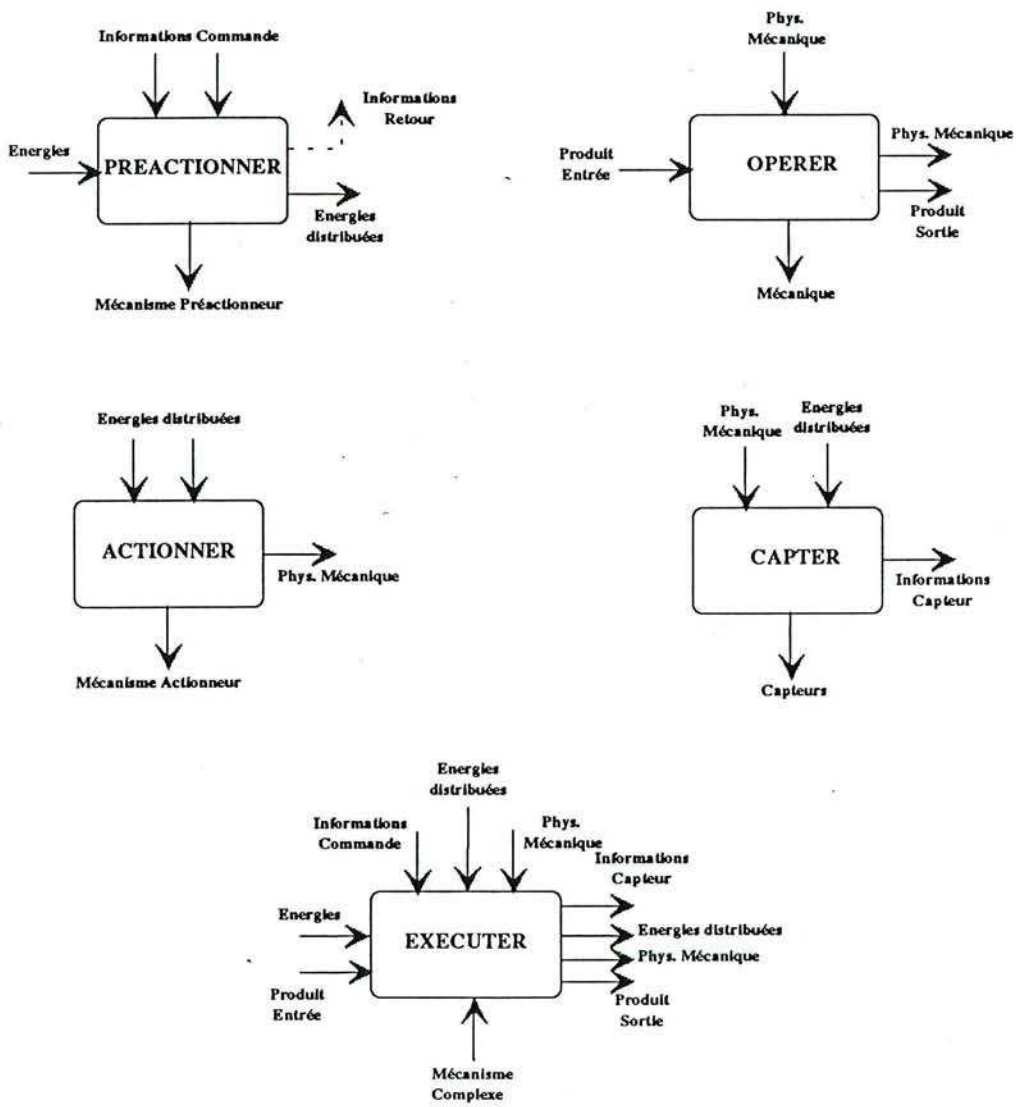


Figure III.5. : Les activités d'un mécanigramme

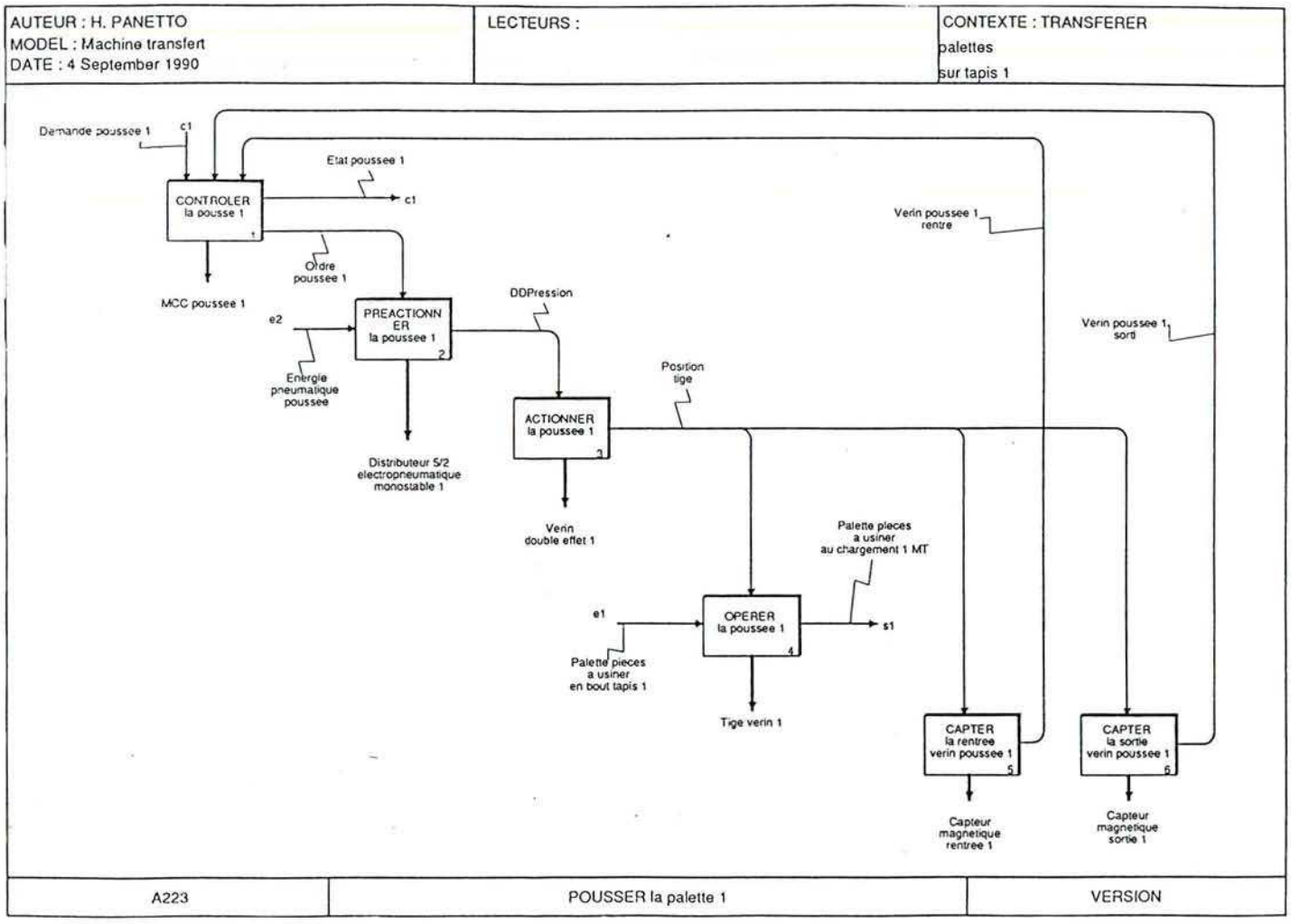


Figure III.6. : Un mécanigramme

Un mécanigramme peut être considéré comme "inutile" d'un point de vue commande et pourraît être remplacé avantageusement par un schéma technologique du mécanisme, pour information. Néanmoins, il permet d'identifier précisément l'**interface physique** (activité "OPERER") toujours préexistante car elle représente la mécanique opérante de la fonction et ainsi d'en déduire l'**interface logique** concrétisant la notion de "**Filtre d'Elément de Partie Opérative**" [LHO 85], [ALA 86], [COR 88], seul "objet" de comportement générique qui semble être reconnu à ce jour.

2ème étape : Détermination des fonctions de contrôle

La décomposition du processus en sous-processus permet d'identifier une "structure" de fonctions de "CONTROLE" (encore appelées **Modules de Contrôle-Commande** (MCC)) qui confère à chacune d'elles un **rôle de décision** à un niveau de processus donné. Les règles de décomposition impliquent que toutes les fonctions de contrôle soient liées les unes aux autres sauf :

- au premier niveau : la première fonction de contrôle est liée, en amont, de manière globale à l'environnement du processus étudié ;
- au dernier niveau : une fonction de contrôle est liée, en aval, à des activités de type "Préactionner"/"Capter".

La phase de "détermination des fonctions de contrôle" doit permettre de détailler :

- le flux d'informations qui lie les fonctions de contrôle les unes aux autres, avec l'environnement et avec les préactionneurs et capteurs du processus étudié,
- préciser le rôle que doit tenir chaque fonction de contrôle (**Sous Fonctions de Contrôle**) ainsi que les liens nécessaires entre les sous-fonctions retenues (Figure III.7).

Il s'agit donc de déterminer la nature et les spécifications des **fonctions de contrôle** à mettre en oeuvre pour satisfaire la demande d'automatisation. Il est ainsi possible, pour la majorité des processus manufacturiers, de classer ses fonctions de contrôle en trois grands types ayant chacun des horizons de décision différents [LHO 85] :

- les fonctions de "**coordination**" avec un horizon temps réel ayant des temps de réponses très courts compatibles avec des actions de type réflexe ,
- les fonctions de "**conduite**" caractérisés par des dimensions temporelles à échelle humaine,
- les fonctions de "**pilotage**", à horizon plus général et des temps de réponse moins critiques permettant des prises de décisions d'ensemble et optimisée.

Ces fonctions se distinguent aussi par la nature des traitements qu'elles mettent en oeuvre et par la nature des informations qu'elles traitent.

Ces fonctions de contrôle sont figées quelle que soit l'installation étudiée et ont un rôle précis dans l'expression des besoins du concepteur. Nous leur associons donc **un modèle "standard" de référence** [LHO 84], c'est à dire indépendant de toute application, défini comme étant la réunion de trois fonctions à un premier niveau de détail (Figure III.8) :

- PILOTER
- CONDUIRE
- COORDONNER

Chacune de ces fonctions peut être elle même décomposée à un niveau de détail supplémentaire de la manière suivante :

- pour PILOTER : "SUIVRE", "OPTIMISER", "REGLER"
- pour CONDUIRE : "PERCEVOIR", REFLECHIR", "AGIR"
- pour COORDONNER : "SURVEILLER", "REAGIR", "COMMANDER"

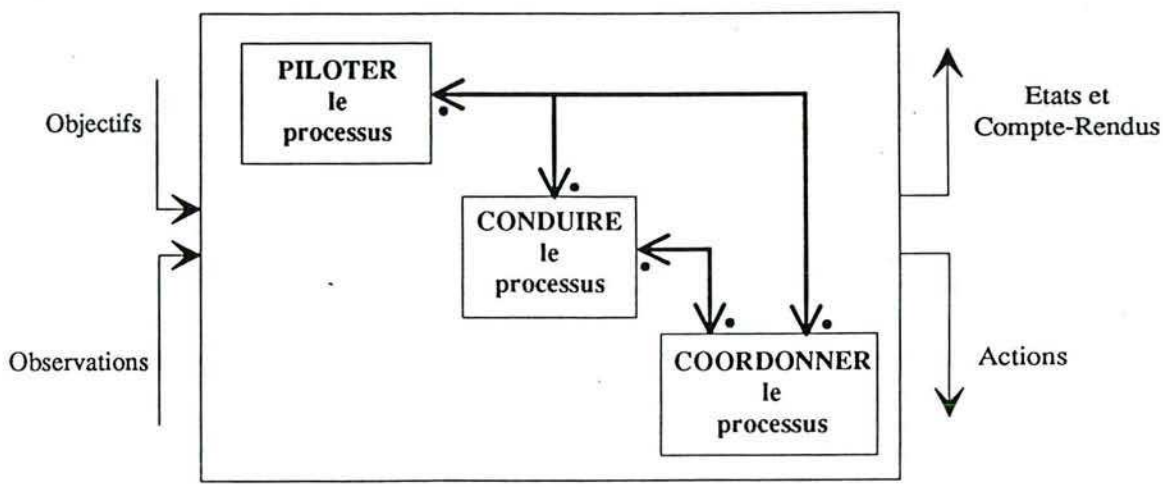


Figure III.7. : Structure conceptuelle d'un MCC [LHO 85]

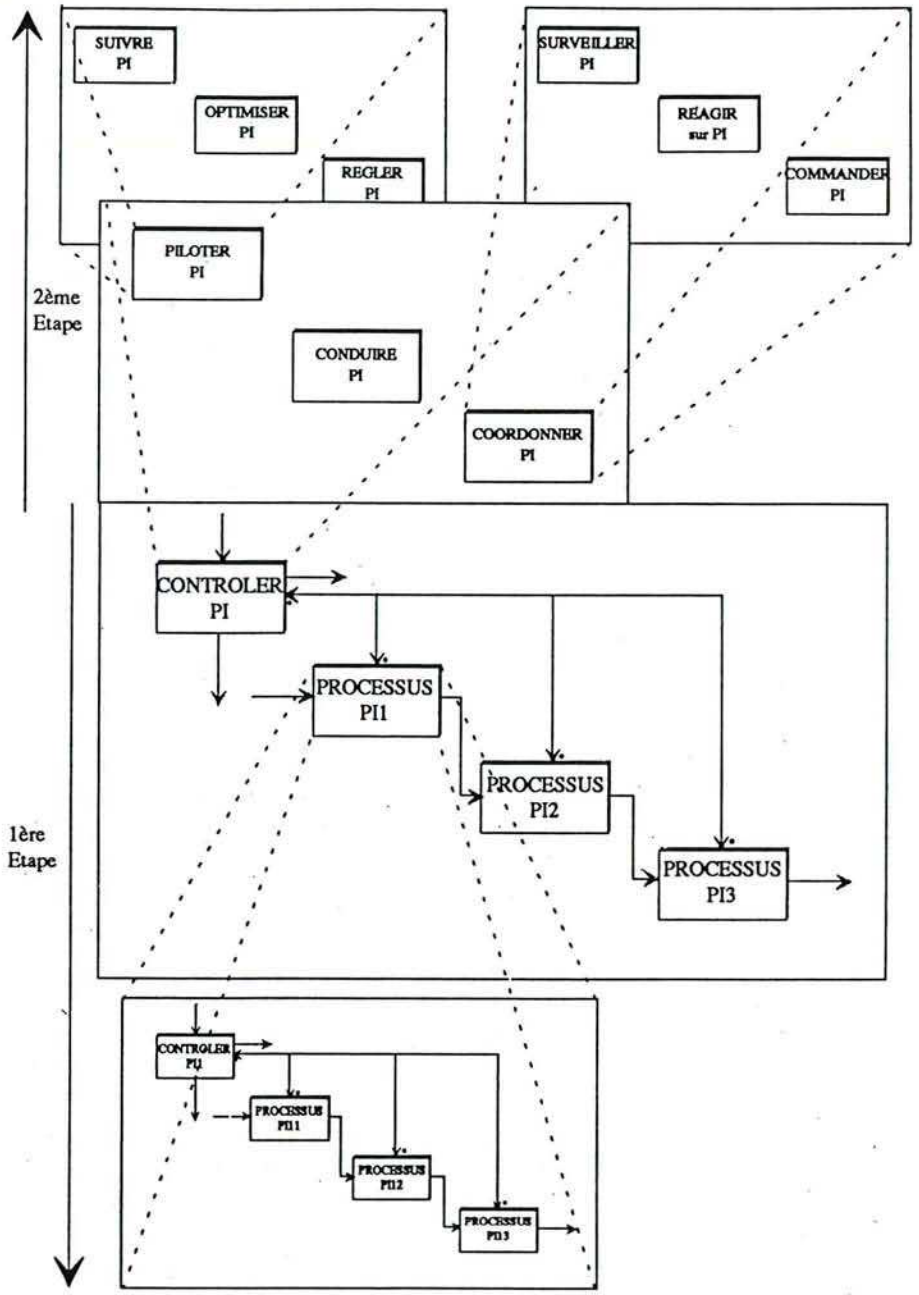


Figure III.8. : Processus de décomposition d'une fonction de contrôle

2.4. L'outil associé : GRILLES

Toute méthode ou démarche d'analyse s'avère difficile à mettre en oeuvre lorsqu'elle reste au stade "papier". L'expérience de la méthode SADT/IDEF0 a vite montré ses limites tant qu'il n'existait pas de support informatique permettant un contrôle de cohérence des modèles et simplifiant au maximun les saisies, permettant ainsi de concentrer toute l'attention sur la qualité des modèles produits. L'outil Orchis [TNI 89], support de la méthode IDEF0, outil que nous avons spécifié, nous permet de décrire la structure hiérarchique des processus mis en oeuvre mais ne permet pas l'étude exhaustive et explicite (tout du moins graphiquement) des fonctions de contrôle-commande. Cela nécessite, de la part des concepteurs qui en ont la charge, une prise de décision quant aux rôles confiés aux différentes fonctions de contrôle "PILOTER", "CONDUIRE", "COORDOONER" ainsi qu'une description par affinements successifs des flux d'informations entre les différentes fonctions identifiées. SADT n'étant pas adapté à la modélisation des données, [LHO 85] a défini un ensemble de grilles de description des missions de contrôle-commande permettant la modélisation des fonctions de contrôle et de leurs sous-fonctions ainsi que l'identification des données transitant entre les différentes fonctions.

1. Méthodologie de conception

L'outil **GRILLE** informatise ainsi les grilles de description des missions de contrôle-commande préconisées par [LHO 85] en liaison directe avec les résultats de modélisation initiale effectuée sur Orchis. A chaque **fonction de contrôle-commande** et chaque sous-fonction retenue est associée une grille de description qui importe toutes les informations saisies en phase de modélisation SADT et permet de détailler les différents flux d'informations, l'ensemble des données ainsi recueillies étant centralisé dans un **dictionnaire de données**. Ce dictionnaire de données permet d'enrichir les grilles des autres fonctions de contrôle lors de leur édition en respectant ainsi une cohérence implicite des informations, toute modification effectuée dans une grille étant répercutée sur les autres grilles en relation.

A un premier niveau de détail, une première grillle sera affectée à "CONTROLER" (Figure III.9a et III.9b). Le principe utilisé est très simple : on représente sur un axe abscisse et un axe ordonnée les sous-fonctions "Piloter", "Conduire", "Coordonner" ainsi que les sous-processus identifiés dans le modèle SADT et une fonction représentant le niveau de contrôle supérieur. On oriente la grille de telle manière à donner une signification aux cases issues du quadrillage (ex : case numérotée 13 = informations provenant de "Piloter" et allant au sous-processus 1).

Pour déterminer les sous-fonctions de contrôle, on commencera par détailler les relations importées du diagramme SADT (relations entre "Contrôler" et les Sous-Processus et relations entre "Contrôler" et le niveau de Contrôle Supérieur). Après avoir réparti et détaillé ces relations (ex : la relation qui apparaît sur le diagramme SADT entre "Contrôler" et le Sous-Processus 1 se détaille sur la grille dans les cases numérotées 13, 17, 23 et, en retour, 14, 18, 24), on considèrera les relations internes à la fonction Contrôler (en fonction des sous-fonctions de contrôle retenues) en détaillant les cases numérotées 7, 9, 11, 8, 10, 12.

Pour chaque degré de détail supplémentaire sur une des sous-fonctions de contrôle, on éditera un deuxième type de grille dont la construction est similaire à la première, mais fait intervenir le contexte d'une manière plus globale en représentant les "ENTREES" et les "SORTIES" de la sous-fonction étudiée. Ces cases "Entrées" et "Sorties" sont directement importées de la première grille (Figure III.10). Par exemple, pour donner le détail d'une fonction "Coordonner" dans la case "Entrées" de la grille correspondante ("Surveiller", "Réagir", "Commander"), on retrouvera les informations identifiées dans les cases 5, 9, 11 ,24 ,30 ,36, 40, 42 de la grille "Contrôler" (verticale de la fonction) et dans la case "Sorties", les informations des cases 6, 10, 12, 23, 29, 35, 39, 41 (horizontale de la fonctions). Ceci permet d'assurer la continuité de la démarche : en effet, une modification à un niveau de détail va pouvoir se répercuter sur les niveaux amont et aval.

Après avoir rempli ces grilles détaillées, le concepteur dispose d'une meilleure spécification de ces fonctions puisqu'il a identifié les ressources d'informations dont dispose chaque fonction ainsi que ses productions attendues.

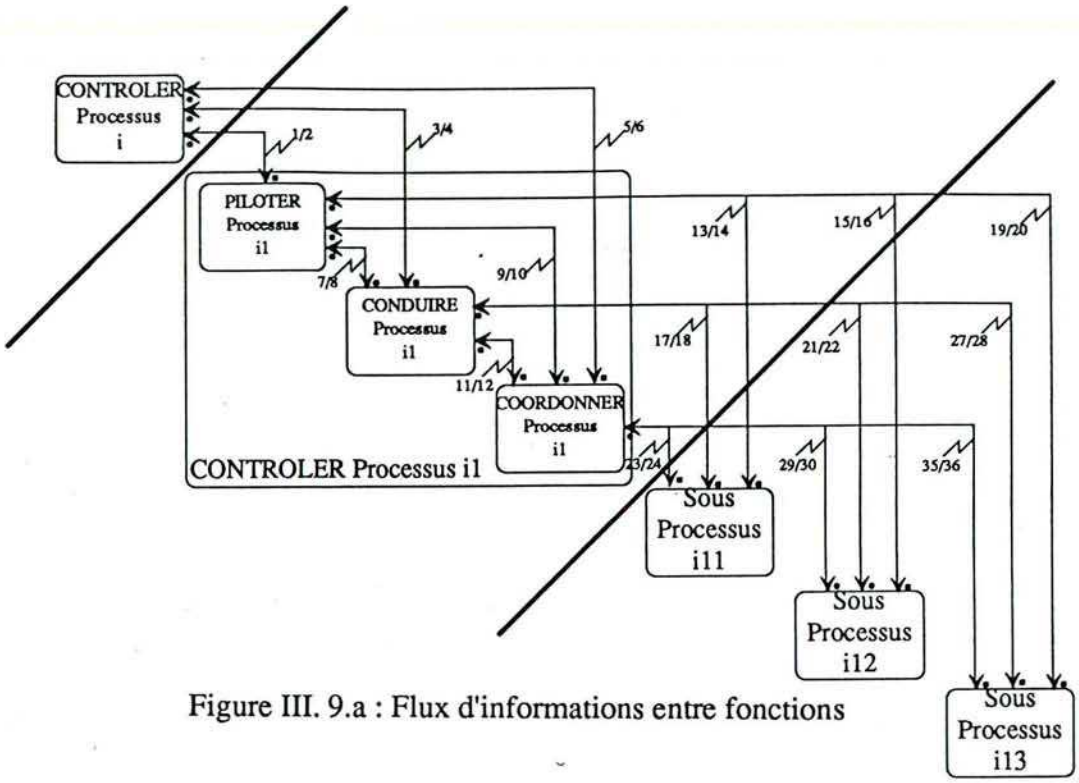


Figure III. 9.a : Flux d'informations entre fonctions

DIAGRAMME : CONTROLER Pi1						GRILLE : MCC Pi1			A1
Informations	CONTROLE SUPERIEUR	CONTROLER			SOUS-PROCESSUS No 1	SOUS-PROCESSUS No 2	SOUS-PROCESSUS No 3	SOUS-PROCESSUS No 4	SOUS-PROCESSUS No 5
		PILOTER	CONDUIRE	COORDONNER					
CONTROLE SUPERIEUR	CONTROLER P1	1	3	5					
CONTROLER PILOTER	2	PILOTER	7	9	13	15	19	25	31
CONTROLER CONDUIRE	4	8	CONDUIRE	11	17	21	27	33	37
CONTROLER COORDONNER	6	10	12	COORDONNER	23	29	35	39	41
SOUS-PROCESSUS No 1		14	18	24	P111				
SOUS-PROCESSUS No 2		16	22	30		P112			
SOUS-PROCESSUS No 3		20	28	36			P113		
SOUS-PROCESSUS No 4		26	34	40				P114	
SOUS-PROCESSUS No 5		32	38	42					Pi15

Figure III. 9b : Grille associée à une fonction de Contrôle-Commande

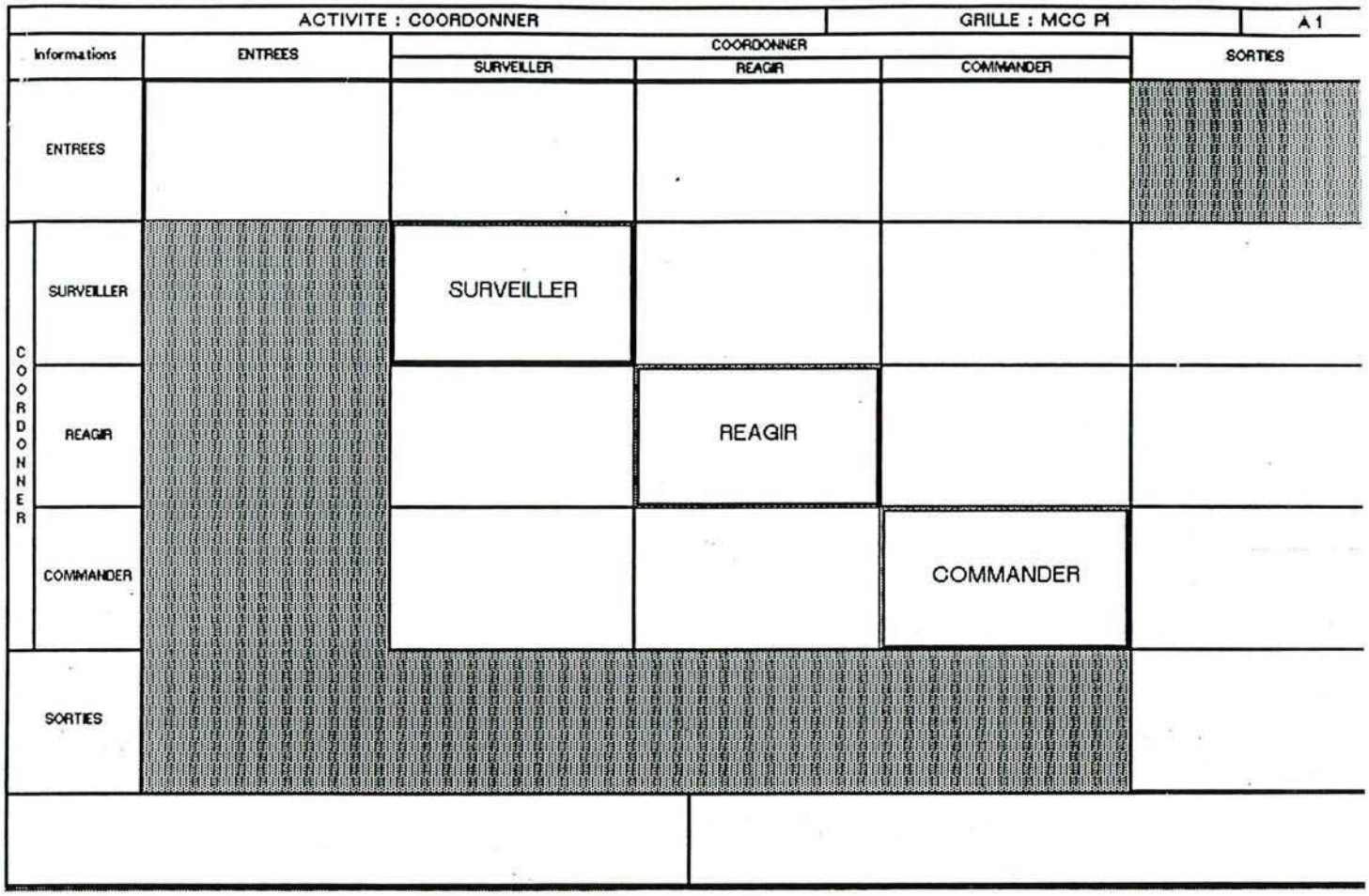


Figure III. 10. : Grille associée à la sous-fonction "COORDONNER"

Les figures III.11, III.12 et III.13 représentent, sur un exemple, respectivement un diagramme SADT dont on désire décrire la fonction de contrôle, la grille de premier niveau associée à cette fonction montrant l'ensemble des informations "importées" du diagramme puis cette même grille que l'on a enrichi par un détail des informations et des sous-fonctions de contrôle. La figure III.14 montre la grille de définition de la sous-fonction "COORDONNER" une fois complétée.

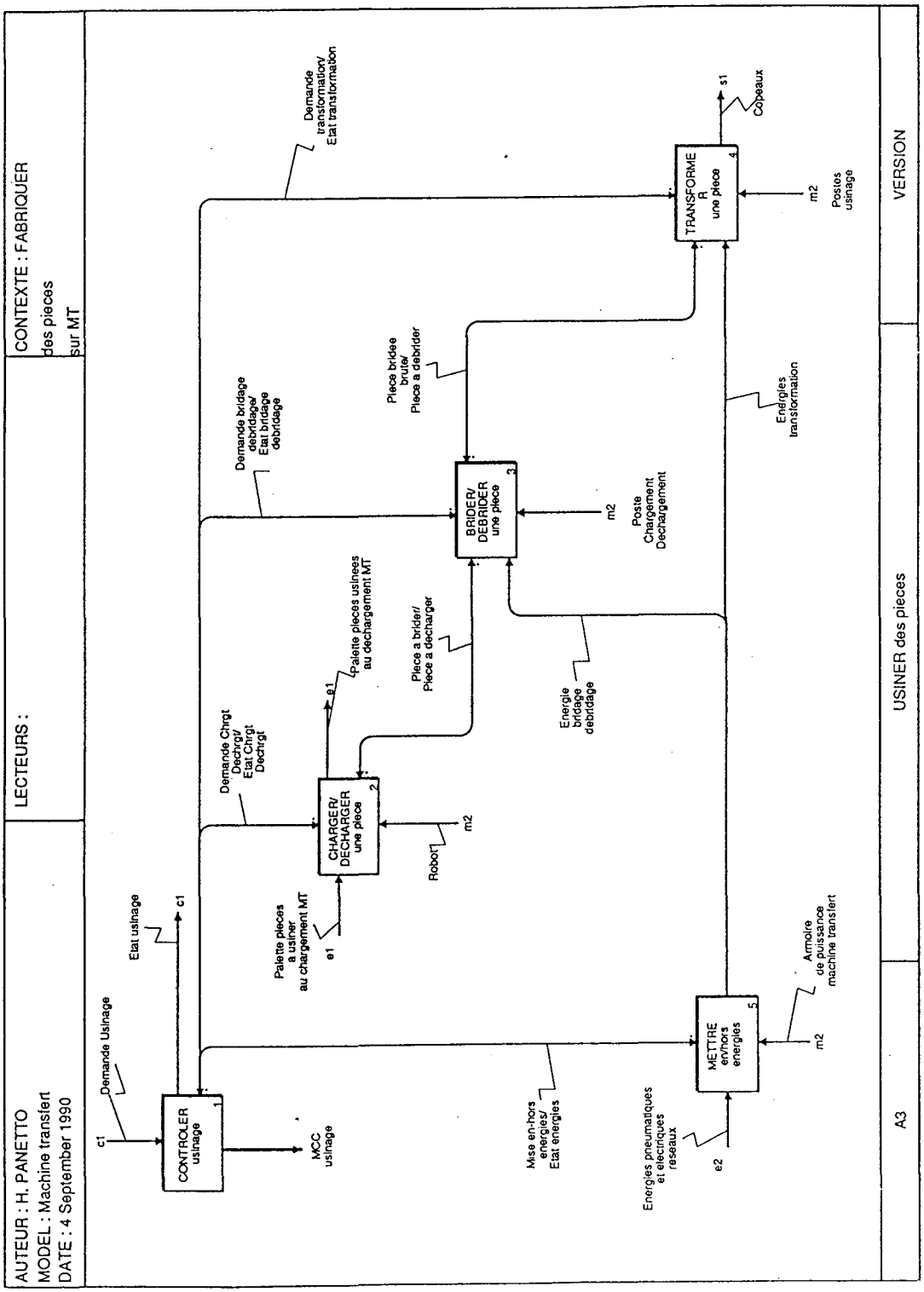


Figure III.11. : Exemple de diagramme Orchis dont on désire étudier la fonction de Contrôle-Commande

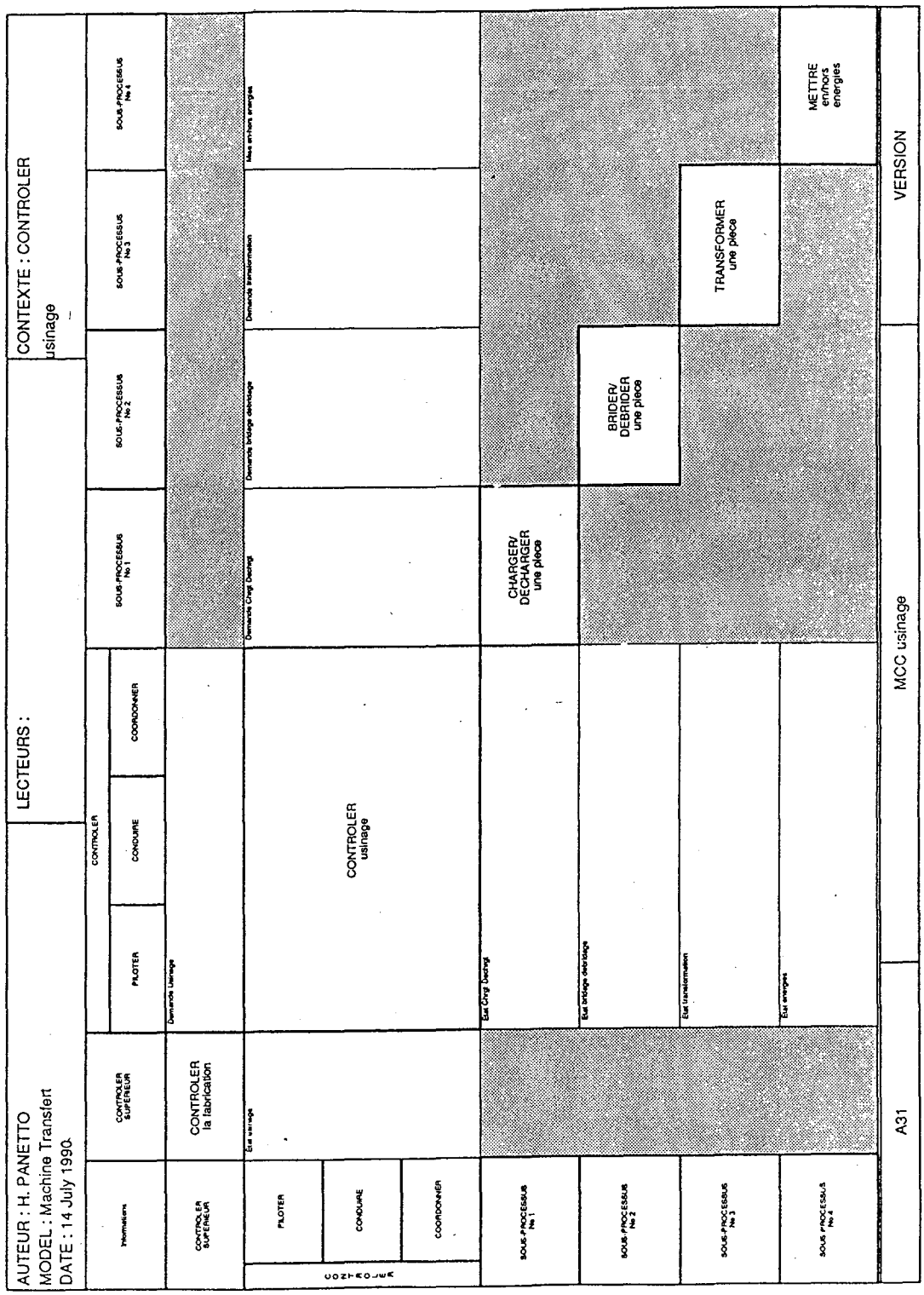


Figure III.12. : Grille associée à la fonction de contrôle commande : 'CONTROLER usinage' avant édition

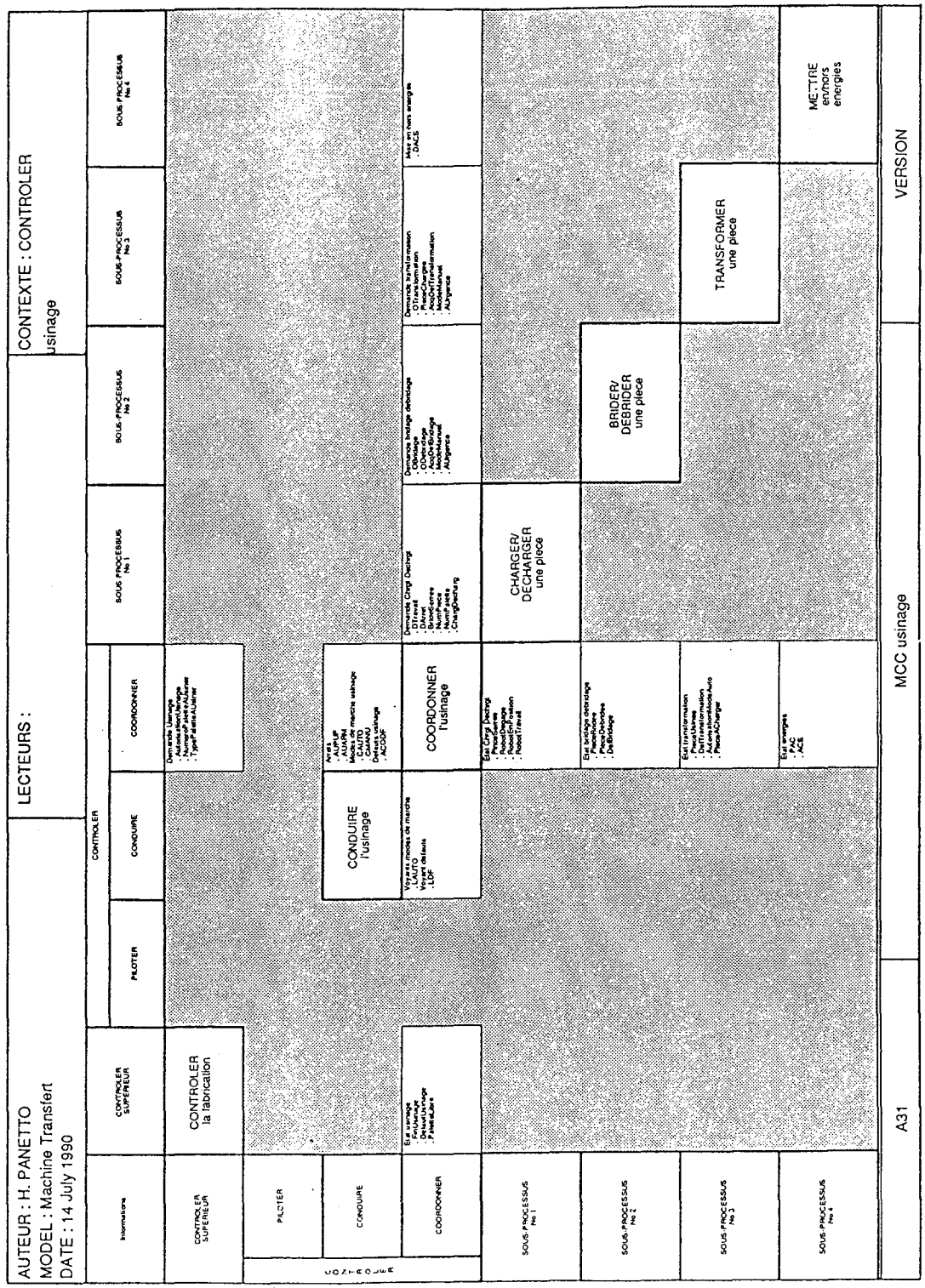


Figure III.13. : Grille associée à la fonction de contrôle commande : 'CONTROLER usinage' après édition

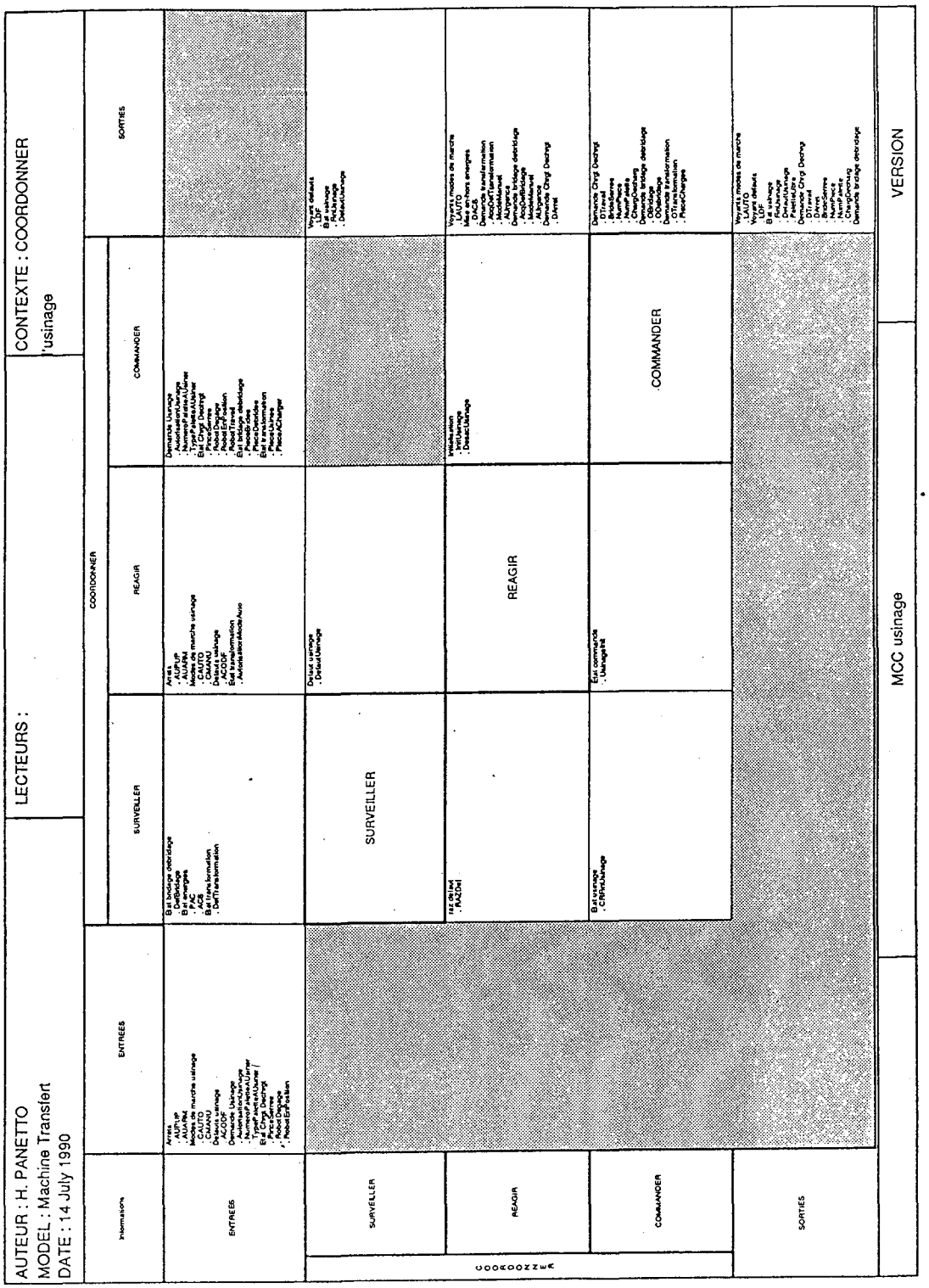


Figure III.14. : Grille associée à la sous-fonction de contrôle commande : 'COORDONNER l'usinage'

2. Typage "sémantique" particulier des informations

Les règles de décomposition et les relations inter-fonctions peuvent impliquer un certain déterminisme quant à la nature sémantique des informations échangées. La "**standardisation**" de la structure peut amener à une définition standard du détail des informations associées aux différents labels employés dans le diagramme SADT [LHO 88a]. Nous pouvons donc "typer" ces informations d'un point de vue sémantique et ainsi induire un certain nombre de questions que devra se poser systématiquement le concepteur de l'automatisme. Nous distinguons plusieurs types d'information en fonction des rôles affectés aux "producteurs" et "consommateurs" de ces données (Figure III.15).

Par exemple les relations entre la **fonction de contrôle** du processus étudié et le **niveau de contrôle supérieur** ou les **sous-processus "contrôlés"** sont du type "Demande/Etat" où la "Demande" peut être constituée de :

- des ordres de marche (Ordre),
- la propagation de modes de marches particuliers (Mode manuel local, Maintenance, ...),
- les arrêts normaux ou anormaux (Arrêt normal, Arrêt d'urgence, ...),
- les acquittements (acquittement défaillance),
- des données de configuration (numero pièce),
- .... ;

et l'état peut être défini comme :

- des compte-rendus de fin d'opération ou d'états particuliers (initialisation),
- des anomalies de fonctionnement,
- des informations sur les modes de marches propres aux sous-processus,
- .....

Les relations entre les **sous-fonctions de contrôle** "CONDUIRE" et "COORDONNER" peuvent être du "type" "Modes/Visualisations" où les "Modes" peuvent être détaillés par :

- des changements de modes de fonctionnement **locaux** au processus étudié,
- des demandes d'arrêt ou marche opérateur **locaux** au processsus étudié,
- des commandes manuelles d'actionnement du processus.
- .....

et les "Visualisations" constituées :

- de voyants de défaillances,
- d' alarmes et autres dispositifs d'alerte,
- de voyants de modes de fonctionnement ,
- .....

Cette interprétation **sémantique** des échanges d'informations incite ainsi le concepteur à déterminer au préalable la *pertinence* des données manipulées et à effectuer un *choix*, non trivial mais nécessaire, quant au niveau de contrôle où elles doivent apparaître.

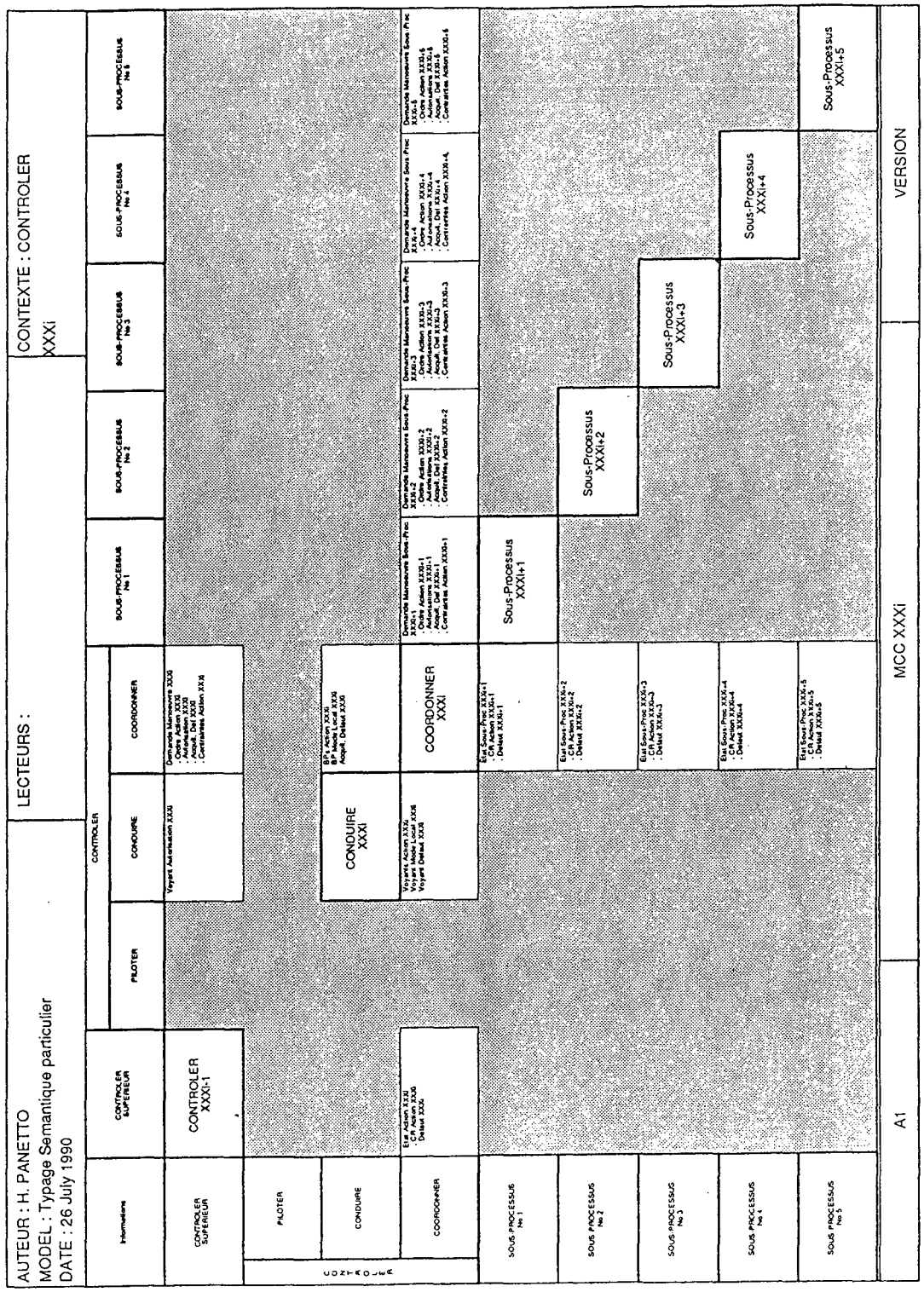


Figure III.15. : Exemple de typage sémantique des informations

3. La documentation

L'aspect documentation n'est pas négligé, le dossier de conception intègre une vue de la grille éditée ainsi qu'un glossaire des différents termes et variables employés. L'ensemble du dictionnaire de données (Figure III.16) peut être édité ainsi que toutes les références croisées associées aux différentes informations utilisées.

Etat translation broche fraisage
. BrocheFraisageHaut
. BrocheFraisageBas
. DefTransBrocheFraisage

Etat percage
. FinPercage
. DefPercage
. PercagePret

Demande translation broche fraisage
. QDescenteBrocheFraisage
. QMonteeBrocheFraisage
. AcqDefTranslationFraisage

Demande percage
. QPercage
. AcqDefUnite
. AUUnite
. ModeReglage

Etat convoyage
. PaletteAmontEnPlace
. PaletteAvalEnPlace
. TypePaletteAmont
. TypePaletteAval
. PaletteAmontLibre
. PaletteAvalLibre

Figure III.16. : Exemple d'édition du dictionnaire de données

2.5. L'interface GRILLE-SPEX

Alors que les étapes précédentes correspondent à une démarche d'analyse "**descendante**" (du plus général au plus détaillé), la phase suivante amène une démarche "**mixte**", c'est à dire "**ascendante**" pour la description des fonctions de contrôle définies dans la hiérachie de décomposition (phase de conception) et "**descendante**" pour une description des relations entre fonctions de contrôle (spécification).

Cette étape correspond à la phase de "Conception Générale" telle que menée classiquement par les concepteurs automatiseurs.

L'ensemble des informations définies précédemment sont utilisées pour l'intégration dans l'application d'un ensemble d'exemplaires de modèles de comportement génériques définis dans l'outil SPEX (Figure III.17) par leur corps et leur interface (BFG, DFG) ou leur interface seule (BFGV, DFGV). En effet, chaque activité COORDONNER, CONDUIRE et PILOTER sera modélisée dans l'outil par un **exemplaire de Boîte Fonctionnelle (BFE ou BFVE)**, l'activité CONDUIRE modélisant une **partie du pupitre opérateur (BFE décrivant un pupitre)**. Les relations entre les différents exemplaires de modèles définissent la structure de l'application qui pourra être testée par simulation. Cette architecture correspond, lors de l'utilisation de comportements "**vides**", à une **spécification exécutable** (maquette) du système puis, lorsque ces comportements sont précisés, à la **conception exécutable** (prototype) de la partie commande de l'installation.

Nous importons, pour chaque activité de contrôle identifiée, l'ensemble de ses données d'entrée et de sortie. Ces données sont utilisées pour la définition des relations entre BFE (ou BFVE) (**Modèles de Contrôle-Commande : MCC**) prétestés et archivés en bibliothèque.

La structure hiéarchique de la décomposition peut être réutilisée par l'assemblage, au sein d'un **Diagramme Fonctionnel**, de l'ensemble des modèles de contrôle-commande associés à des sous-processus de même niveau hiérarchique.

Exemple d'assemblage de modèles

MCC Verin Percage exemplaire BF **Bistable**

+

MCC Moteur Percage exemplaire BF **moteur1Sens1Vitesse**

+

MCC Percage exemplaire BF **Unité**

+

Flux d'informations entre "*COORDONNER le percage*" (MCC Percage) et les sous-processus contrôlés (Verin et Moteur)

=

DF **UnitéPercage** qui après instanciation donne le Sous-processus **PERCER**

2.6. Apports de l'informatisation de la démarche

L'informatisation de cette démarche lui confère une praticabilité accrue et offre ainsi un support en enseignement, en particulier pour son utilisation dans le cadre de projets d'étudiants de 2ème et 3ème cycle, et en formation continue. Cette approche a, d'autre part, montré sa praticabilité par des mises en oeuvre industrielles dans :

- l'automatisation d'un montage d'usinage embarqué [LHO 89],
- l'automatisation d'une machine transfert [PAN 90],
- la rédaction d'un cahier des charges d'automatisation pour la rénovation d'un tour à commande numérique en vue de sa réalisation par une société de services [AIP 91]

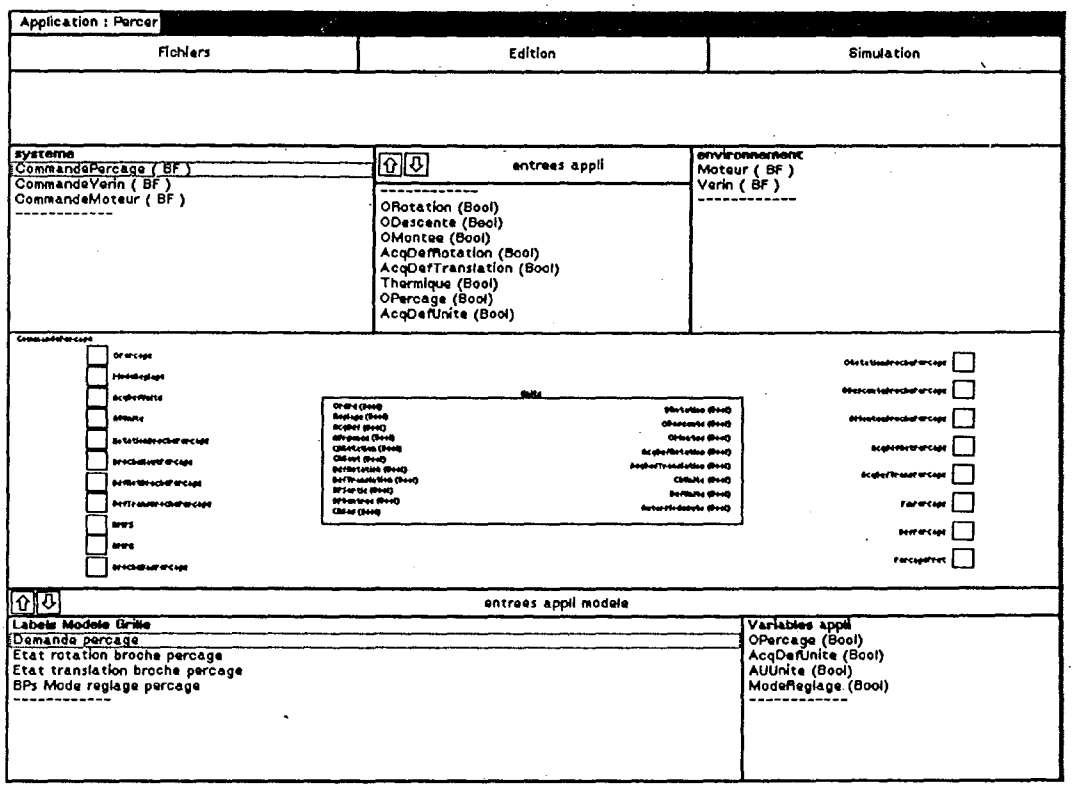


Figure III.17.: Exemple d'application SPEX

3. Généralisation de la méthode et informatisation

[BEL 89] propose d'identifier les *Modules d'Analyse de Comportement, de Contrôle et de Commande* (MA3C) (Figure III.19), [EDF 90] décrit le fonctionnement **d'actionneurs intelligents** [CIA 88] par des *Modules Fonctionnels d'Automatisme* (MFA) (Figure III.20) dans une optique CMMS (Control, Maintenance and technical Management System) [CMM 90], [LHO 85] préconise une recherche des *Missions de Contrôle Commande* (MCC) (Figure III.7) d'un système automatisé comme vu au paragraphe précédent, ... Ces modèles, aussi différents soient-ils, s'adressent tous à la conception de systèmes automatisés. En fait, chacun n'est qu'un modèle particulier de référence (Figure III.18) au sens CIM [GAC 90]. Pour la création d'un outil apte à supporter ces différents modèles, deux statégies différentes peuvent être engagées :

- sur la base de ces modèles particuliers, déduire un modèle de référence partiel (ou méta-modèle) ;
- offrir une structure d'accueil configurable pour chaque modèle.

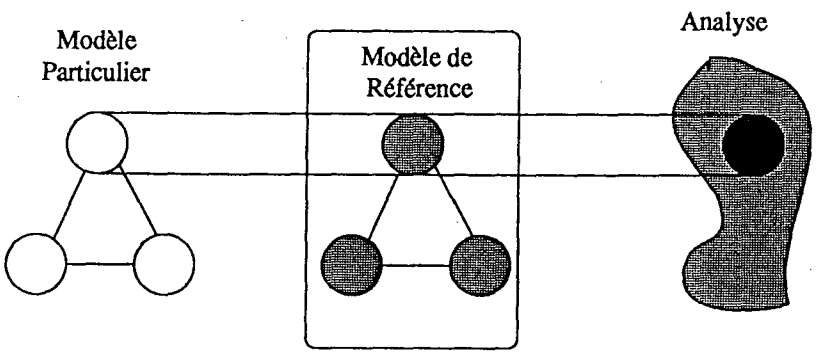


Figure III.18. : L'application d'un modèle de référence

C'est cette deuxième solution que nous avons retenu, la constitution d'un modèle de référence partiel demandant encore un important travail de recherche.

Nous avons, donc, en généralisant le principe de "grille" exposé précédemment, fournit un outil de "configuration de grilles" permettant la définition et l'application d'un de ces modèles particuliers pour la conception des Machines et Systèmes Automatisés de Production. Le configurateur peut ainsi être assimilé au méta-méta-modèle de l'outil GRILLE permettant, par instanciation, la création d'un méta-modèle particulier qui définit un modéle de référence particulier.

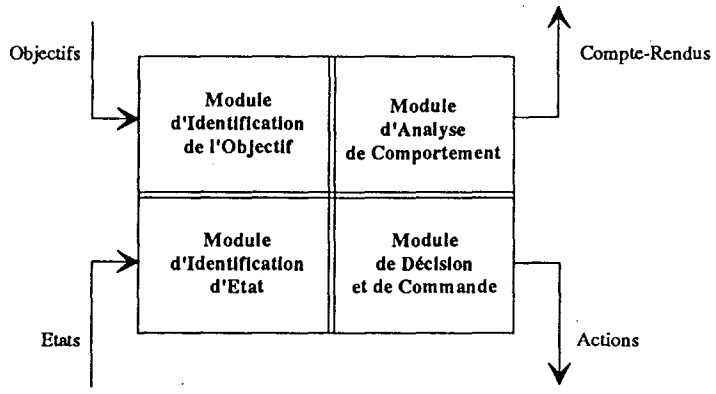


Figure III. 19. : Structure conceptuelle d'un MA3C [BEL 89]

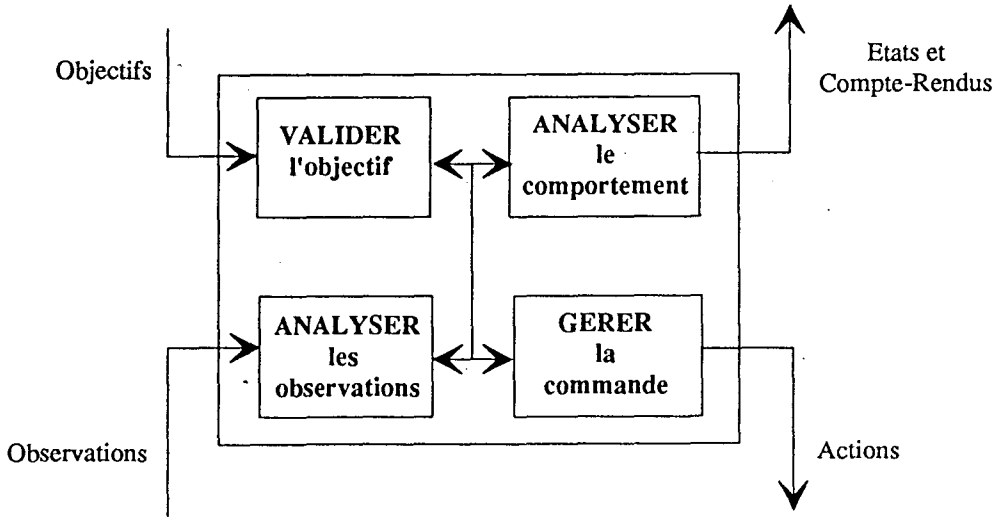


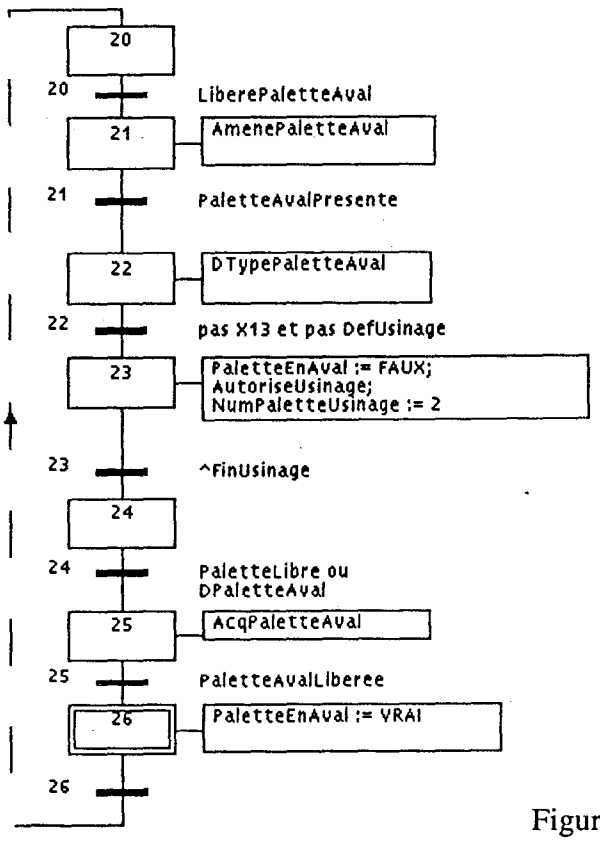
Figure III.20.: Structure conceptuelle d'un MFA [EDF 90]

1.2. Vers les phases aval du cycle de vie

1. Générateurs de code

SPEX est un outil qui s'adresse principalement à l'**Automatiseur** et au **Génie Automatiseur**. Il permet d'exécuter une spécification et une conception en vue de vérifier, par simulation, le comportement attendu de l'installation. Cette tâche est, le plus souvent, réalisée en *"Bureau d'études"* et doit servir de point de départ au codage de la partie commande de l'installation réelle.

Les outils de codage tel que CADEPA [SGN 90] doivent rester des outils de terrain, c'est à dire à proximité des installations automatisées pour permettre le téléchargement des programmes générés et l'animation graphique des comportements pour la mise au point puis, pour la maintenance. CADEPA permet la conception d'un automatisme par une description graphique en formalisme Grafcet et/ou Schéma à relais du comportement de commande de l'installation. Il intègre, de plus, une fonction d'"émulation console" avec une animation dynamique des graphes saisis, CADEPA n'étant pas capable de la reconstituer à partir de la description textuelle.




```
% "Grafcet fabrication" (X26)
C
X26>Y26>X20>Y20>X21>Y21>X22>Y22>X23>Y23>X24>Y24>X2
5>Y25>X26
C
Y20 [LiberePaletteAval]
Y21 [PaletteAvalPresente]
Y22 [pas X13 et pas DefUsinage]
Y23 [↑FinUsinage]
Y24 [PaletteLibre ou DPaletteAval]
Y25 [PaletteAvalLiberee]
Y26 [=1]
C
X21 [AmenePaletteAval]
X22 [DTypePaletteAval]
X23 [PaletteEnAval := FAUX;
AutoriseUsinage;
NumPaletteUsinage := 2]
X25 [AcqPaletteAval]
X26 [PaletteEnAval := VRAI]
C
```

Figure III.21. : Format interne CADEPA

L'interface entre SPEX et CADEPA peut être réalisée par la création d'un fichier contenant une description textuelle du grafcet, format d'entrée des générateurs de code de CADEPA (Figure III. 21). La faisabilité de cette interface a été montrée mais n'a pas été retenue car il est nécessaire, pour une utilisation de la fonction animation de CADEPA que SPEX lui fournissent une image des graphes.

L'interface envisagée est donc une transmission des "vues" graphiques de comportement décrit dans SPEX dans un format compatible avec les éditeurs de CADEPA. Il aura ensuite la charge de générer le code API et d'animer les grafcets correspondants. Cette interface nécessite cependant une ouverture des traducteurs quant au format de fichier graphique.

Toute modification apportée à la conception du système de commande devrait débutée par une remise en cause des comportements décrits dans SPEX, puis une fois vérifiée par simulation, être transmise au traducteur. En fait, il s'avère que les délais imposés au service maintenance ne permettent pas d'effectuer cette "**rétro-conception**" [MORa 89] en ligne mais doit être répercutée a posteriori.

L'interface ainsi définie doit donc être **bidirectionnelle** en ce sens où, toute modification de la conception sur l'outil d'édition du traducteur doit être répercutée en amont sur l'outil SPEX pour **maintenir une cohérence** entre la spécification de conception et le codage réel.

Il est à noter que SPEX est indépendant de l'architecture de commande cible, il ne tient pas compte des particularités d'implémentation relatives aux automates supportés par CADEPA. Il est donc nécessaire d'homogénéiser les formalismes entre les éditeurs de SPEX, non contraignants, et ceux de CADEPA, particularisés pour chaque type automate en terme de nombre de symboles, utilisation de Blocs Fonctionnels standards, ...

Par exemple, un réseau de schémas à relais édité sur l'outil SPEX n'est limité que par la taille de l'écran alors que CADEPA limite le nombre de lignes et de contacts par ligne en fonction de l'automate cible.

2. Le diagramme Opératif : architecture répartie de la commande

La conception de la partie commande des Machines et Systèmes Automatisés de Production aboutit généralement à une description hiérarchique fonctionnelle du système à automatiser, cette décomposition étant indépendante de l'architecture matérielle cible. En simulation, l'accent est surtout porté sur le **résultat désiré** et présuppose, généralement, un fonctionnement global de l'installation sur un seul système de commande. En fait, un Sytème Automatisé de Production (SAP) est composé de deux parties : une partie matérielle qui engendre l'évolution des fonctions du système, une partie logicielle qui personnalise le comportement du matériel pour que l'évolution globale soit conforme aux spécifications détaillées de l'application.

La complexité croissante des installations automatisées industrielles montre que la phase de conception ne se restreint pas à la seule description des différents comportements élémentaires ainsi que de leurs interactions respectives, mais elle nécessite aussi une définition précise de l'organisation matérielle multi-processeurs avec toutes les contraintes que cela implique d'un point de vue synchronisations et communications inter-processeurs de commande [CAL 90].

Il est donc nécessaire de réaliser un "lien" (ou encore une interface) entre le **domaine fonctionnel**, propre au concepteur de l'application, et le **domaine organique**, propre au réalisateur de l'application (Figure III.22). Cette "interface" doit, comme cela est le cas entre la spécification et la conception, **réutiliser**, sous certaines contraintes, les résutats de conception pour la description de l'architecture organique de commande.

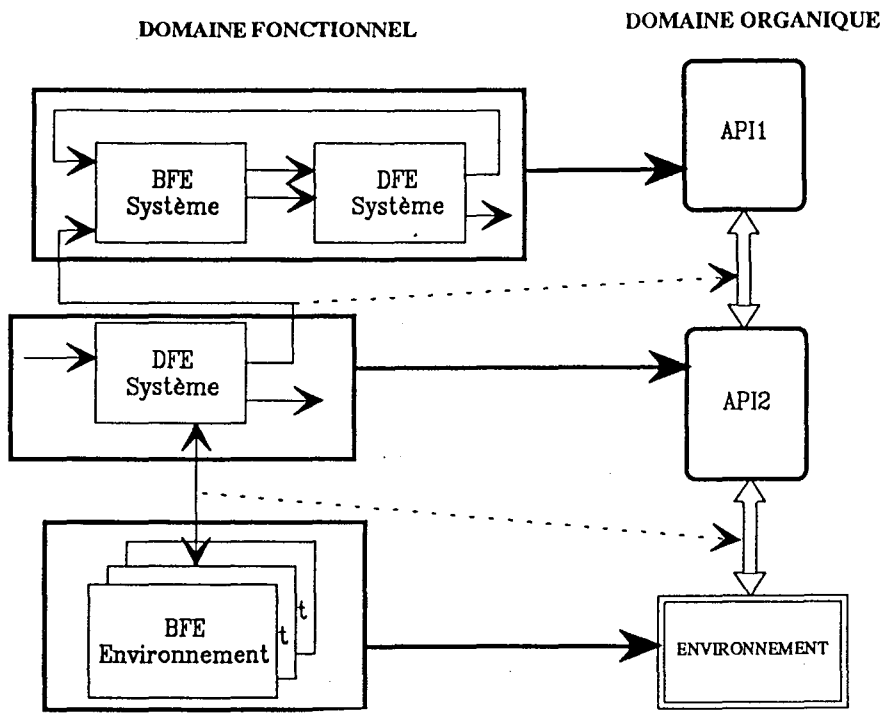


Figure III.22. : Du domaine "fonctionnel" au domaine "organique"

C'est en ce sens que nous proposons un modèle de conception **organisationnelle** [LON 88] qui reprend le découpage fonctionnel de l'application effectuée lors de la phase de conception détaillée et affecte chaque composant ou groupe de composants à une structure d'accueil, le **Diagramme Opérationnel** (DO) exprimant les caractéristiques de regroupement et les contraintes des supports matériels. Les liens entre ces fonctions (relations entre différents processeurs), déduits de l'architecture de l'ensemble des DO identifiés, sont modélisés par des **Diagrammes de Communication** (DC). Ce modèle décrit ainsi un comportement d'ensemble de l'application proche de l'architecture réelle d'implémentation.

2.1. Architecture de commande multi-processeurs

Les différentes fonctions assurées par un système de commande sont réparties sur un grand nombre de processeurs de commande inter-reliés par des liaisons physiques (entrées/sorties, réseaux, ...).

Le modèle organisationnel doit donc tenir compte de cette architecture distribuée et permettre de modéliser les différentes communications mises en oeuvre pour assurer la synchronisation de l'ensemble et la bonne adéquation entre le comportement prévu de l'installation et son comportement réel et, éventuellement, remettre en cause, tant la conception fonctionnelle que l'architecture de commande (communications trop ou pas assez chargées, processeurs inadéquats, etc ...)

2.2. Le DO : regroupement fonctionnel mono-processeur

A partir de la description fonctionnelle de l'ensemble de l' installation, nous proposons de définir une structure topologique de **Diagrammes Opératifs** (DO) regroupant dans une même "*enveloppe*" les fonctions supportées par un même processeur de commande (réel ou virtuel). Un DO sera donc dédié à une partie commande particulière dont nous devrons modéliser le comportement d'un point de vue communication, synchronisation et multi-tâches. Ce découpage peut être réalisé à partir de :

- la décomposition de l'application en "**entités fonctionnelles**" autonomes supportées par un processeur unique,
- l'organisation hiérarchique de ces entités et les informations qu'elles s'échangent,
- et enfin, les nécessités de synchronisation entre les différentes parties commande de l'installation.

Le regroupement de différentes fonctions sur un même processeur ne doit pas être réalisé de façon **empirique** mais doit satisfaire une multitude de critères, autant économiques que physiques qu'il faut pouvoir évaluer. Trop souvent, l'architecture de commande est la première réponse à un problème d'automatisation car elle implique des coûts importants, mais sans critère réellement tangible par rapport à la description fonctionnelle (évaluation des tailles mémoires, subjectivité quant au choix des supports de communications, expériences passées, ...). On assiste ainsi à une approche "**technologiste**" où l'architecture de commande devient une contrainte de l'étude d'automatisation alors qu'elle devrait en être le résultat. Voici par exemple quelques critères pouvant guider ces choix [LON 88] :

- *le couplage entre fonctions* : il peut être **temporel** ou **informationnel**. Plusieurs fonctions fortement corrélées impliquent un échange important d'informations et ainsi une charge maximale de communication, aussi bien en ce qui concerne les temps de réponse des interfaces que la sûreté de transfert de l'information. Le regroupement de ces fonctions, si cela est possible d'un point de vue implémentation, tend à garantir une forte cohérence logique du comportement désiré et une validité accrue des informations échangées ;

- *les contraintes d'implantation* : la localisation géographique des fonctions sur un site peut être imposée et nécessite alors le regroupement de celles-ci sur une architecture commune ;

- *les contraintes de synchronisme critique* d'un ensemble de fonctions peuvent induire leur exécution par un même processeur afin de garantir des réactions rapides à des évènements extérieurs communs ;

- *les contraintes de sûreté de fonctionnement* : elles peuvent se traduire par l'ajout de processeurs pour des traitements redondants ou de surveillance.

Le DO est défini, en outre, par un contrôleur d'exécution de la commande décrivant le comportement relatif de l'architecture matérielle indépendamment du logiciel de commande qui décrira son comportement dans l'application, ce modèle étant spécifique à une architecture matérielle.

Toute partie commande (DO) peut être considérée comme une "tâche" réagissant à des signaux externes ou traîtant des messages ou données. Chaque processeur de commande peut transmettre des informations (messages ou données) ou générer des "signaux" (interruptions) vers les autres organes de commande.

Un DO correspond donc à un comportement décrit par une association de comportements supportée par un processeur unique et une interface d'entrées/sorties déduite de cet ensemble. Son contrôleur d'exécution élabore un vecteur de sortie en fonction de son vecteur d'entrée. Un ensemble d'évènements

déduits de ses relations avec l'environnement, par l'intermédiaire éventuel de **Diagrammes de Communication**, peut agir sur son comportement.

2.3. Le DC : modèle de communication inter-systèmes de commande

Le **Diagramme de communication** (DC) modélise un "protocole" de communication entre plusieurs Diagrammes Opératifs, mettant en oeuvre éventuellement des "objets" de communication (boîte à lettre, sémaphore, partage de ressource, ...) et réagissant éventuellement à des évènements extérieurs asynchrones. Le DC pourra être interne à un DO dans le cas où son implémentation se fait sur le même processeur, ou externe si son support physique est indépendant des processeurs de commande, par exemple un système de multiplexage/démultiplexage, une interface d'adaptation de signaux, ...

La nécessité de modéliser les communications apparaît dans les contraintes de temps de transmission et dans les besoins de transformation des informations induits par l'hétérogénéité des architectures matérielles utilisées et la répartition géographique croissante des diverses installations.

Les DC permettent un dialogue entre les différents processeurs de commande et la gestion des synchronisations et des partages de ressources liés au découpage choisi. Un DC doit alors tenir compte des différents processeurs qui lui sont connectés et permettre ainsi de présenter ou stocker les messages qu'il reçoit ou qu'il doit transmettre en fonction d'un protocole particulier. Il apparaît aussi entre la commande de l'installation et la partie opérative pour la commande de mécanismes simples (définition des borniers d'interfaçage parallèle par exemple), de mécanismes complexes (régulation de vitesse, guidage de chariots) ou dans l'avenir, pour un échange d'informations avec des "**capteurs intelligents**" [CIA 87] ou des "**actionneurs intelligents**" [CIA 88].

2.4. Le contrôleur d'exécution

Le **contrôleur d'exécution** correspondant à chaque DO permet de spécifier le "découpage interne d'exécution" de l'application ainsi que le comportement relatif du processeur par rapport aux autres unités de commande réparties de l'installation globale [CAL 90]. L'hypothèse adoptée pour les temps d'exécution

des fonctions est ici caduque, la durée d'exécution des opérations pour un processeur **n'est pas nulle mais finie** et sera traduite par un coefficient relatif par rapport aux autres processeurs de commande (c'est à dire par un niveau de priorité pour l'ordonnanceur de la simulation).

L'objectif de cette modélisation est de permettre une vérification par simulation de l'ensemble du système automatisé en tenant compte, le plus précisément possible, des spécificités d'exécution de chaque support de commande. Nous pouvons entrevoir, sans être exhaustif, deux types de paramètres devant être pris en compte par la simulation :

- les informations de configuration propres au processeur :
  * temps de cycle relatif d'exécution,
  * signaux de type interruption,
  * existence d'entrées rapides pour la commande de procédés à fort degré de réactivité ;

- les possibilités de découpage interne des programmes de commande :
  * tâches traitements séquentiels,
  * tâches traitements synchrones,
  * tâches périodiques,
  * traitements des interruptions (internes ou externes),
  * maîtrise de l'ordre d'interprétation des traitements séquentiels.

Chaque tâche peut être composée d'un certain nombre de fonctions et se voit donc affectée de paramètres d'exécution.

Le contrôleur d'exécution serait donc composé de 4 sous-ensembles :

- **Le "préliminaire" séquentiel** : cet ensemble regroupe la liste des fonctions à exécuter de manière séquentielle dans l'ordre d'apparition dans la liste. Les résultats calculés dans une de ces fonctions peuvent être utilisés dans celles plus en aval.

- **Le traitement à réalisation synchrone** : cet partie décrit l'ensemble des fonctions dont l'exécution s'effectue par une évolution à réalisation synchrone par rapport aux entrées et sorties. Dans l'esprit des automates programmables industriels [GRE 85], l'ensemble des entrées de toutes les entités est figé en début de cycle de scrutation, puis après l'exécution de l'ensemble, les sorties sont mises à jour. L'ordre d'exécution de chaque fonction n'influe donc pas sur l'ensemble du comportement.

- **Le "postérieur" séquentiel** : Il est identique au "préliminaire" séquentiel mais est exécuté après le traitement à réalisation synchrone.

- **Le traitement asynchrone** : chaque fonction affectée ici sera traitée de façon totalement asynchrone suivant son type :

  . *les fonctions cycliques* : elles sont en attente de tops d'horloge dont la fréquence est un de leurs paramètres. Une fois réveillées, elles exécutent leur comportement puis se remettent en attente.

  . *les fonctions sous interruption* : elles ont un mode de fonctionnement identique aux fonctions cycliques mais sont en attente de signaux internes ou externes au processeur.

Il est à noter que, pour éviter tout aléa dans l'exécution de l'ensemble, une fonction appartenant au traitement asynchrone ne peut pas être interrompue pas une autre fonction du même groupe ; les traitements préliminaires, à réalisation synchrone ou postérieurs sont interrompus, avec une sauvegarde de leur contexte, dès le réveil d'un traitement asynchrone.

Nous voyons ici que la diversité des possibilités d'implémentation d'une commande dans un système hôte implique des comportements diamétralement opposés suivant les choix de découpage internes et le type de système de commande retenu. Notre but n'est pas de montrer que telle ou telle solution est la

plus adéquate au problème posé mais d'offrir au concepteur et réalisateur de l'automatisme, la possibilité de décrire **sa solution** et de pouvoir vérifier, par simulation, le comportement de son application en fonction de l'architecture matérielle choisie pour éventuellement la remettre en cause avant la réalisation finale.

2.5. L'Application : un modèle du "réel"

Une Application est donc caractérisée par un ensemble de DO en relations par l'intermédiaire de DC. Chaque DO modélise un comportement sur un processeur de commande donné, chaque DC implémente un protocole de communication entre des sorties d'un DO vers des entrées d'un autre DO. L'Application représente donc un **modèle de l'installation réelle** avec son architecture de commande et ses interconnexions sous réserve d'hypothèses de modélisation d'éxécution d'un processeur par rapport à un autre. La simulation d'une telle application doit permettre **d'essayer** des répartitions de commande sur des sites différents et ainsi valider une architecture de commande particulière et ses implications d'un point de vue communication par une mesure de charges et performances attendues plus précise qu'une évaluation empirique.

2.6. Interprétation du modèle organisationnel

1. Hyphothèses retenues

Plusieurs hypothèses ont été retenues pour la simulation d'une application :

- Chaque Diagramme Opératif correspond à une "tâche" asynchrone par rapport aux autres DO. Il ne communique avec d'autres processeurs que par l'intermédiaire de Diagrammes de Communication.

- Le contrôle de l'éxécution d'un DO est décrit dans son modèle d'éxécution et peut être totalement synchrone ou partiellement asynchrone.

- La durée d'activation d'un DO est "**relative**" à celle des autres DO.

2. Principes de simulation

Algorithme d'interprétation d'un DO

Un Diagramme Opératif est une représentation "abstraite" d'un processeur de commande. Il est constitué d'un ensemble de BFE et/ou DFE, en relation mutuelle, participant à la réalisation d'une ou plusieurs fonctions. Ces composants font partie d'un des quatres groupes distincts définis dans le contrôleur d'exécution associé (cf ce Chapître, 2.5). Pour respecter les principes de simulation de SPEX, calqués sur le principe d'interprétation commun à bon nombre d'Automates Programmables Industriels et permettant une évolution et une réalisation synchrone de l'application, un DO considèrera les BFE décrits dans sa structure comme les **unités élémentaires** de simulation. L'algorithme d'interprétation proposé par l'AFCET [AFC 83] et utilisé par SPEX sera donc adapté pour l'interprétation d'un DO comme présenté Figure III.23.

Il est à noter que cet algorithme correspond à l'exécution des parties préliminaire, synchrone et postérieur décrites dans le contrôleur d'exécution. Un DO pouvant, de plus, implémenter des tâches (périodiques ou sous interruption), ces dernières seront **prioritaires** à l'ensemble du cycle principal qui sera interrompu en cours d'exécution, le temps de l'activation de la tâche concernée.

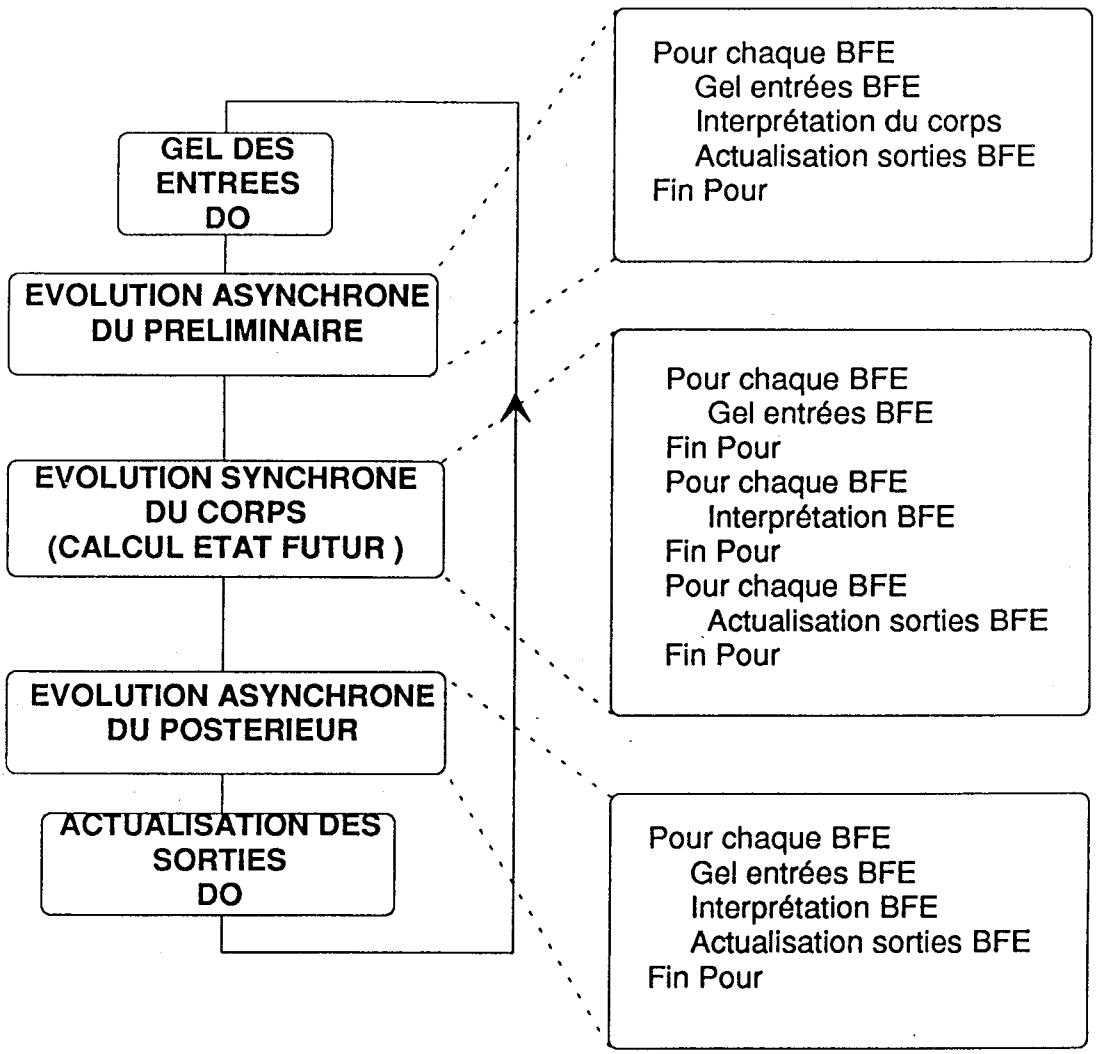


Figure III.23. : Algorithme d'interprétation d'un Diagramme Opératif

Algorithme d'interprétation d'une application

Une application est constituée d'un ensemble de Diagrammes Opératifs, chacun modélisant le comportement d'un processeur de commande. Chaque processeur a son propre cycle d'interprétation indépendamment des autres

processeurs avec qui il doit se **synchroniser**. La simulation d'une application étant réalisée sur une machine mono-processur, il est nécessaire de simuler, le plus exactement possible, un comportement multi-processeur asynchrone sur un processeur unique.

[LON 88] propose de décrire ce **niveau physique** à l'aide de ESTELLE [COU 87], un langage adapté à une exécution asynchrone de processus coopérant uniquement par échange de messages. Dans l'optique de **réutilisation** de SPEX et de son environnement pour étudier la faisabilité d'un outil support d'une **conception organique**, nous utiliserons une technique de simulation, certes moins précise et moins proche du système réel mais suffisante, dans une première phase, à la réalisation de notre problème et ne nécessitant que peu d'investissement logiciel.

L'interprétation d'un Grafcet prend comme hypothèse une discrétisation du temps en "*instants aussi petit que l'on veut*" pendant lesquels aucune variation des entrées n'est susceptible d'intervenir [AFC 87]. Nous reprenons ce même type de raisonnement pour la simulation d'une application en considérant qu'un DO sera exécuté, durant **un** cycle d'interprétation, **au moins une** fois et **au plus (n)** fois en fonction du temps relatif par rapport aux autres DO que nous aurons affecté à son contrôleur d'exécution.

Chaque contrôleur d'exécution se verra donc attribuer un paramètre d'exécution, Tc, représentant le temps, en pourcentage du cycle d'interprétation de l'application, pendant lequel le DO concerné sera susceptible d'attendre son activation, par exemple si Tc égale 50% du cycle, alors il sera exécuté deux fois au cours d'un cycle.

L'activation d'un DO est **prioritaire** à sa désactivation ce qui induit qu'il pourra être *interrompu* en cours d'exécution pour permettre l'évolution d'un autre DO mais cette interruption ne pourra excéder la valeur du paramètre Tc de son contrôleur. Une fois réactivé, il reprendra son évolution, là où il a été interrompu.

Prenons pour exemple celui représenté Figure III.24 : DO1 (ou plutôt son contrôleur d'exécution) a un paramètre Tc1 valant 33%, ce qui implique qu'il sera activé trois fois au cours d'un cycle. DO2 (son contrôleur) a un paramètre Tc2 valant 80%, ce qui indique qu'il pourra attendre 80% du temps de cycle entre deux activations consécutives. d1 et d2 sont les durées d'évolution concernant respectivement DO1 et DO2. Nous remarquons que DO2 ne peut évoluer en un seul cycle car il est entrecoupé, après un temps

d2', par une activation de DO1 après laquelle il peut terminer son évolution (d2").

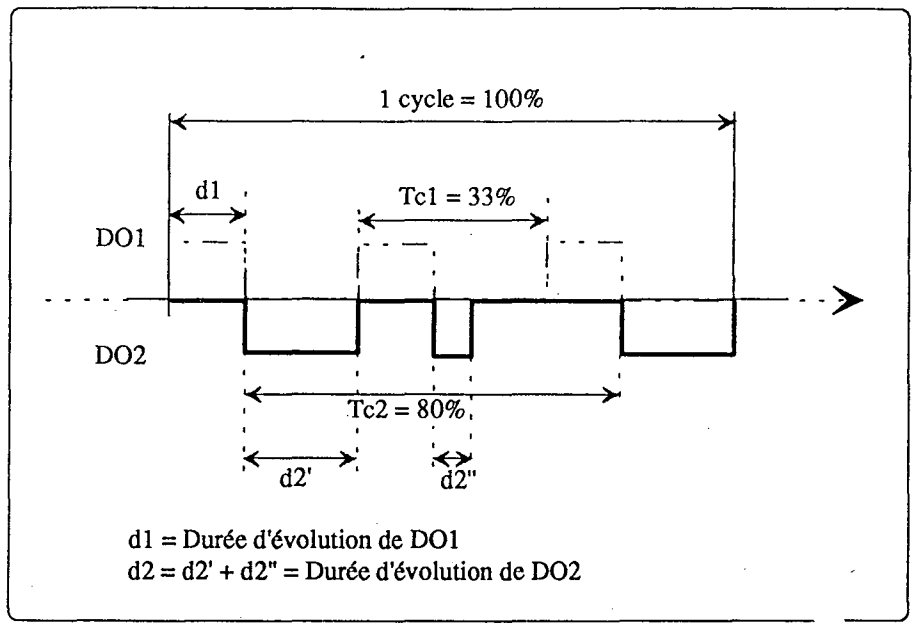


Figure III.24. : Découpage du temps

2.7. Etude de faisabilité de l'outil associé sur les bases de l'outil SPEX

1. Les apports de SPEX

La conception détaillée de l'installation dans l'outil SPEX présuppose l'exécution de la commande sur un processeur unique et impose une évolution de l'ensemble par une réalisation synchrone par rapport aux entrées/sorties de l'application globale. Le principe de simulation retenu par SPEX suit un enchaînement séquentiel composé de 3 étapes :

- Figeage des entrées de chaque exemplaire élémentaire de modèle de comportement
- Calcul de l'état futur de chaque exemplaire
- Affectation de toutes les sorties

Une extension future de l'outil SPEX devrait permettre d'associer un contrôleur d'exécution à l'application mais, pour des raisons d'indépendance de l'architecture d'implantation choisie lors des phases de codage, l'unicité du processeur de commande sera préservée.

2. L'outil de conception organique

Une application décrite dans SPEX nous founit l'ensemble des fonctions devant être assurées par l'automatisme. Ces fonctions sont affectées à un ou plusieurs systèmes de commande. L'outil devra donc nous permettre la définition d'une architecture de DO en 2 étapes :

1ère étape :

Après nous avoir proposer l'ensemble des composants "**système**" de l'application, nous pourrons regrouper un sous-ensemble de BFE et/ou DFE sur un même support de commande (Figure III.25). Le DFE étant, par définition, un ensemble de BFE, il pourra être nécessaire de "casser" ce DFE pour en extraire un certain nombre de BFE que l'on désire dissocier.

L'architecture de DO ainsi définie induit des communications. En effet, là où il y avait une **relation fonctionnelle** et où il y a maintenant une **séparation organique** impose une communication.

L'ensemble des communications élémentaires identifiées peuvent maintenant être regroupées sur des supports physiques (**Diagrammes de Communication**).

Chaque diagramme de communication doit maintenant être, soit affecté à un **Diagramme Opératif** particulier s'il est supporté par le même processeur, soit être affectée à un **Diagramme Opératif particulier** qu'il est nécessaire de créer et qui représente le support matériel de mise en oeuvre d'un DC "externe".

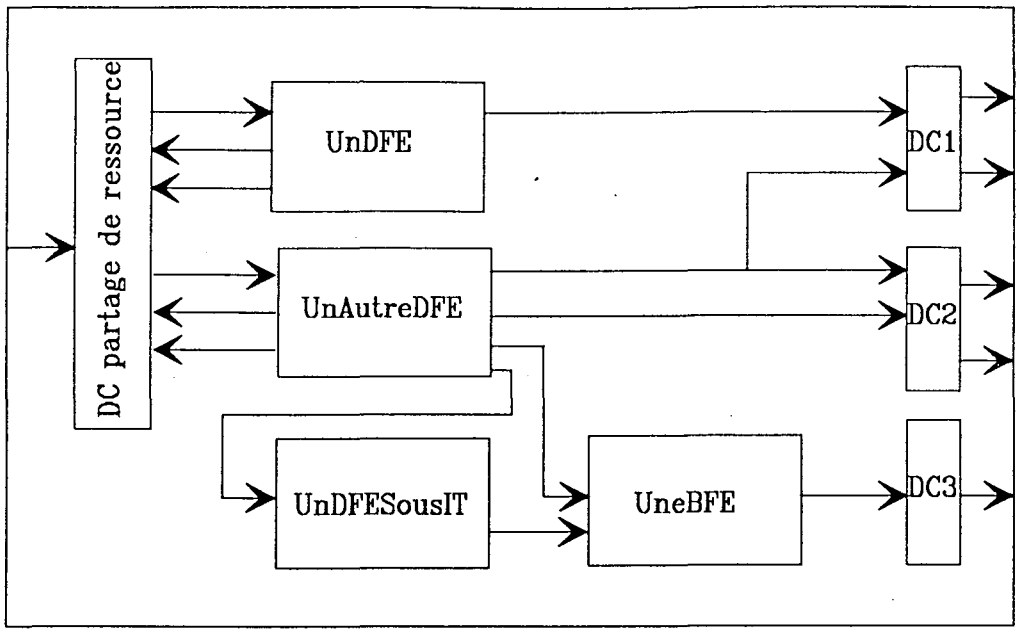


Figure III.25. : Vue synthétique d'un Diagramme Opératif

2ème étape :

Il est maintenant nécessaire de configurer le **contrôleur d'exécution** en fonction du type de découpage choisi pour permettre une simulation de l'ensemble. (Figure III.26)

Cette paramétrisation nous permettra de définir les fonctions à exécution séquentielle, synchrone, cyclique ou à l'apparition d' évènements (interruptions). L'ensemble modélisera le comportement **relatif** d'un système de commande.

CONTROLEUR D'EXECUTION	
PRELIMINAIRE SEQUENTIEL	
REALISATION SYNCHRONE	UnDFE UnAutreDFE
POSTERIEUR SEQUENTIEL	UneBFE
IT1 (* 100 ms)	UnDFESousIT

Figure III.26. : Vue synthétique d'une contrôleur d'exécution

Une fois toutes les affectations réalisées, un éditeur graphique proche de l'éditeur d'application de SPEX (Figure III.27) nous permettra la construction de notre application par la définition du comportement des DC déduits des relations entre les différents DO. Un DC sera , en fait, un exemplaire de modèle de comportement prédéfini dans la bibliothèque de SPEX.

La simulation de l'ensemble de l'application réutilisera les tableaux de bord et utilitaires de simulation déjà présents dans l'outil SPEX.

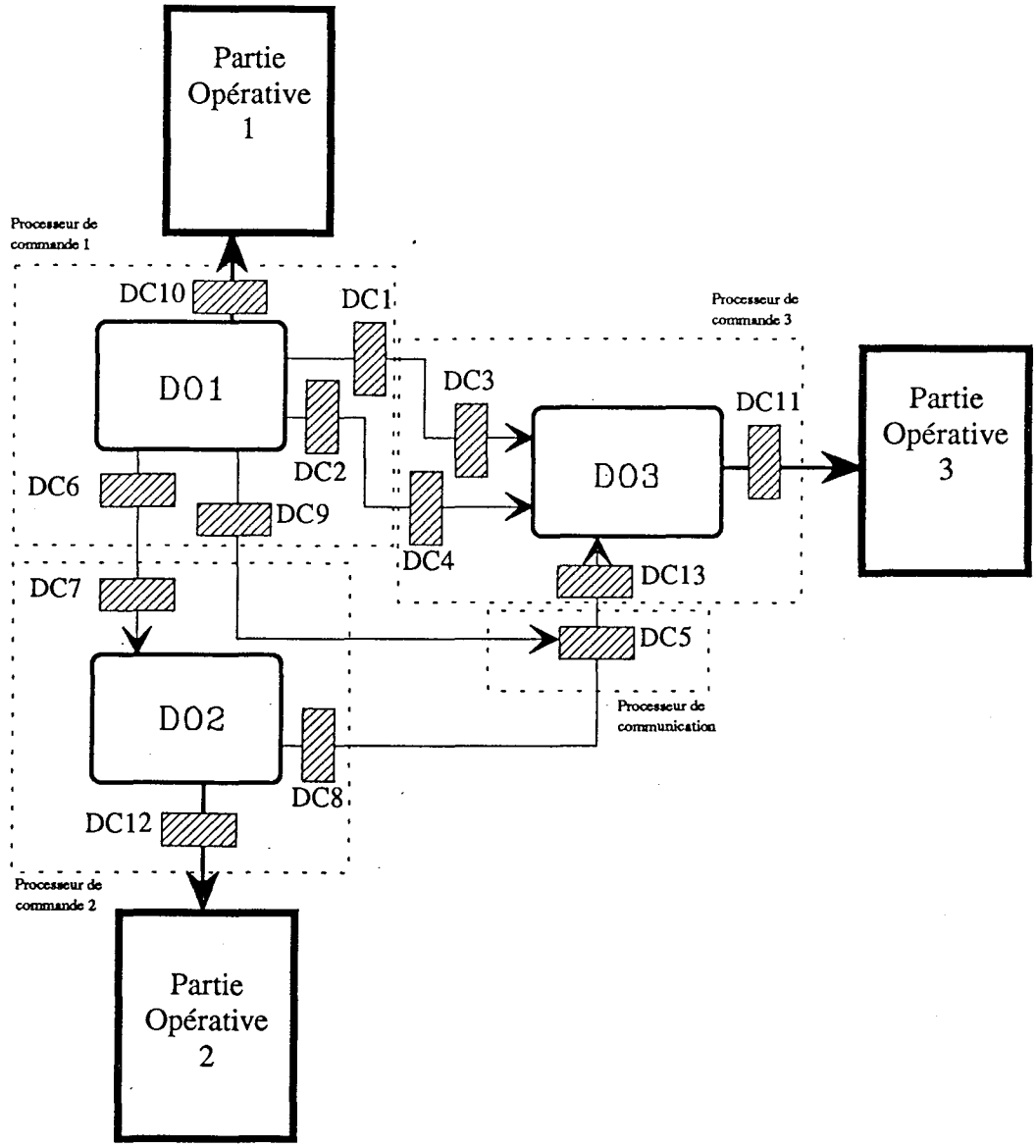


Figure III.27. : Vue d'une application

3. Emulateur de Partie Opérative

L'activité de test représente environ 50% du coût du développement d'un automatisme. Toute amélioration de la productivité passe donc par la mise en oeuvre d'outils permettant d'améliorer l'efficacité et le reproductibilité des opérations de test.

Les tests en plateforme sont une nécessité pour assurer la qualité d'un automatisme. Cependant, les tests appliqués sont, trop souvent, totalement indépendants des tests de conception. Il est donc nécessaire d'imposer, au moment des intégrations successives des réalisations, **les mêmes scénarios** que ceux ayant permis de valider les spécifications de ces réalisations.

La connexion de SPEX à l'émulateur de Partie Opérative MAXSIM [MAX 91] (Figure III.28), outil développé sur la base de SPEX, nouvelle génération de l'outil ADELAIDE [COR 85] [COR 89] offrirait cette possibilité et, de plus, éviterait de décrire (de manière non continue et multiple) les mêmes modèles à des stades différents d'avancement de l'automatisation .

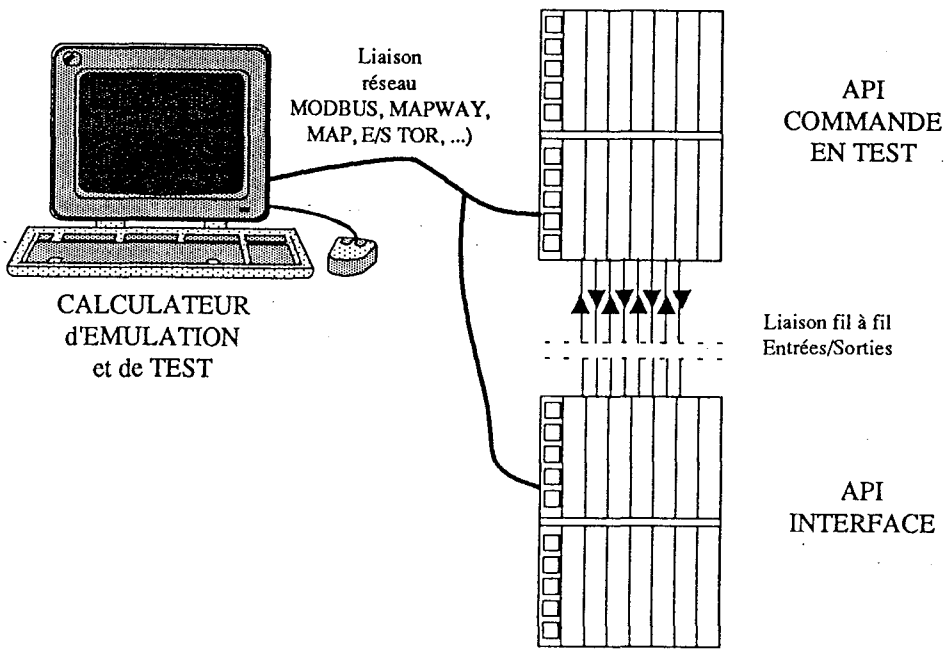
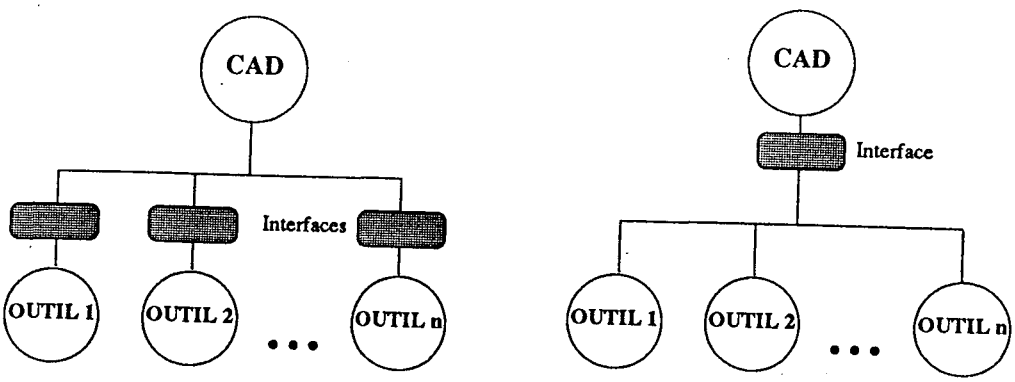


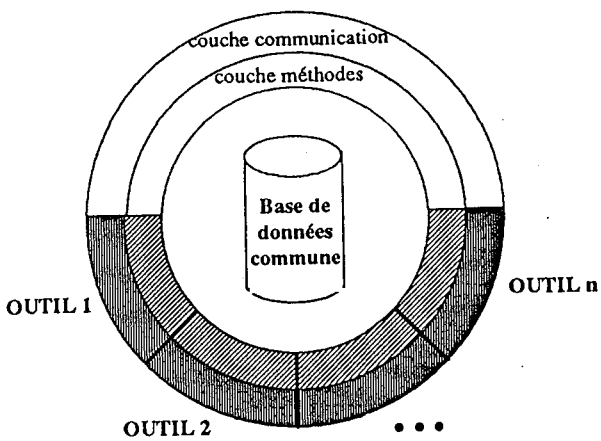
Figure III.28. : MAXSIM

II. CONCLUSIONS DU CHAPITRE III

La "connexion" d'outils proposée n'est pas, dans un premier temps, une intégration au sens propre, elle ne réalise en effet qu'une interconnexion nécessitant autant d'interfaces que de couples d'outils connectables (Figure III.29). Cependant, c'est un premier pas vers l'intégration, montrant sa faisabilité qu'il faut ensuite généraliser en terme de *structure conceptuelle* d'un **Atelier de Génie Automatique**.



(a) Connexions



(b) Intégration

Figure III.29. : Différences entre connexion et intégration [GRA 90]

SOMMAIRE
Chapitre IV

I. ... A LA SPECIFICATION D'UN ATELIER DE GENIE AUTOMATIQUE PARTICULIER

Nous débuterons ce chapitre par un bilan des **Ateliers de Génie Logiciel** et des travaux en cours pour la définition d'un **Atelier de Génie Automatique.**

L'intégration des outils dans un Atelier de Génie Automatique devrait, à terme, permettre une standardisation des échanges d'informations entre outils ainsi qu'une adaptation de ces outils à des méthodologies spécifiques à l'entreprise utilisatrice ou pour un domaine d'application particulier. L'**outil logiciel** pourrait ainsi être considéré comme un "**objet**" manipulé et configuré par une structure d'accueil, l'Atelier. Nous définirons les principes de base d'intégration d'outils au sein d'un **Atelier de Génie Automatique**, en particulier par notre approche pour la définition d'un modèle conceptuel "objet" d'un **Atelier de Génie X particulier** sur la base des définitions de tels modèles en **Génie Logiciel.**

Nous terminerons ce chapitre par nos perspectives d'intégration d'outils et leur mise en oeuvre dans le cadre d'un projet de recherche, en collaboration avec la société SGN Lorraine et le Laboratoire d'Automatique et de Micro-électronique de Montpellier.

1. Bilan des Ateliers de Génie Logiciel

Depuis les débuts de l'informatique, on s'est préoccupé de produire et de maintenir des programmes. La croissance exponentielle en volume et complexité des applications informatiques a fait naître une discipline baptisée "**software engineering**" à l'origine des concepts d'"atelier logiciel", **ensemble de matériels et logiciels censés regrouper harmonieusement les méthodes, les techniques et les outils du Génie Logiciel** [GIR 89]. Nous pouvons rajouter à cette définition qu'un **Atelier de Génie Logiciel** (AGL) doit être organisé pour la production de logiciels de qualité. [THE 88] précise, en outre, qu'un AGL est une structure (d'accueil) et non un logiciel. En effet, comme une méthodologie est un ensemble de méthodes, un AGL est un ensemble d'outils couplés entre-eux et organisés en vue d'améliorer la qualité et la productivité d'un développement logiciel. Cette intégration doit permettre la manipulation d'"objets" représentant du texte, du code ou du graphisme, quelle que soit la phase du cycle de vie ou le niveau

d'abstraction des outils. Pour remplir ce but, l' AGL doit être cohérent et organisé et l'on parle alors d' **Atelier Intégré de Génie Logiciel** (AIGL). Un AIGL doit comporter [GIR 89]:

- une structure interne unique des objets manipulés, chaque outil participant au développement pouvant en exploiter une partie,
- des mécanismes cohérents de gestion des objets,
- une base de donnée (ou d'objets) gérant les versions et configurations de chaque objet,
- et doit supporter la totalité du cycle de développement logiciel.

Les ateliers de **première génération** sont des "Boîtes à outils" style Unix visant à regrouper sur un même support ou un même système d'exploitation, un ensemble d'outils d'aide à la réalisation de logiciels sans aucune inter-connexion.

Ceux de **deuxième génération** tendent à accroître leur intégration en permettant à des outils, souvent hétérogènes, de communiquer via des passerelles et autres dictionnaires de données (EXCELERATOR en 1984 , ...).

Actuellement, apparaîssent des AIGL de **troisième génération** qui prennent en charge ce travail d'équipe et la notion de projet par une gestion centralisée de l'ensemble des objets intervenant dans tout le cycle de développement [BEN 89] (STP [STP 88], EXCELERATOR [EXC 89], MEGA [MEG 89]). Ces ateliers devraient permettre l'automatisation de la production de code à partir de spécifications écrites dans un langage formel en réutilisant un ensemble de composants logiciels standards [THE 88], supporter tout un ensemble de méthodes, aptes à exprimer les besoins des concepteurs en fonction du niveau d'abstraction choisi et de leur avancement dans le projet logiciel.

C'est ce type d'atelier que l'on peut caractériser de "**méta-atelier**" [GIR 89], c'est à dire offrant la possibilité de paramétrer les méthodes utilisées, les langages de spécification, de conception ou de programmation en fonction des caractéristiques du projet ou de l'entreprise utilisatrice. Cette notion concerne surtout le producteur de d'atelier ou "méta-concepteur" car l'utilisateur final attend sutout une facilité d'emploi et une réponse concrète à ses besoins particuliers.

Dans un futur proche, des ateliers de **quatrième génération** [THE 88] devraient intégrer l'intelligence artificielle et permettre une automatisation du Génie Logiciel en fournissant une aide à la conception par une déduction automatique de modèles à partir d'une base de connaissance.

2. Etat de l'art des Ateliers de Génie Automatique

Dans l'ensemble, et assez paradoxalement, le **Génie Automatique** est assez peu *automatisé*, d'une part parce que toutes les phases d'un cycle de vie d'un **Système Automatisé de Production** (SAP) ne sont pas supportées par des outils informatisés et, d'autre part, parce que les outils des automatiseurs sont la plupart du temps complètement **déconnectés** et n'échangent donc aucune information de manière automatique et continue [MOR 89].

Les travaux de spécification d'un **P**oste de **T**ravail pour l'Automaticien [PTA 87] et, par la suite, la définition d'un **modèle conceptuel des données** relatives à un SAP [BAS 88] ont permis de mettre en évidence les principes généraux d'un **Atelier de Génie Automatique** par :

* une pré-étude de faisabilité d'un *Atelier Logiciel pour le Génie Automatique* (ALGA) [3IP 88] :

Cette pré-étude avait pour objectifs de déterminer les nécessités et les possibilités d'**inter-connexions** des outils de l'Automatiseur entre eux après enquête auprès d'utilisateurs potentiels d'un ALGA (Figure IV.1). Elle a permis, de plus, de dresser un état de l'art des méthodes et outils employés par les concepteurs d'installations automatisées. Il en ressort plusieurs points :

- un ALGA doit permettre d'articuler un ensemble d'outils logiciels d'aide à la conception des installations automatisées sur la base d'une structure d'accueil gérant les données communes aux outils ;
- un ALGA doit être un atelier ouvert et configurable pour des classes d'utilisateurs et des classes d'applications. La *méthodologie* générale de l'atelier (choix des outils et méthodes) doit pouvoir être particularisée par rapport à ces classes et pour des besoins spécifiques ;
- les données manipulées doivent rester cohérentes entre outils par une mise en commun dans un format standard ;
- le manque d'outils d'aide à la conception dans les phases hautes du cycle de vie et surtout leur connexion avec des outils "avals" est clairement ressenti ;

- les outils doivent, à chaque stade du développement, permettre la **validation** des résultats par des procédures de tests cohérentes avec celles employées dans les phases amont d'analyse.

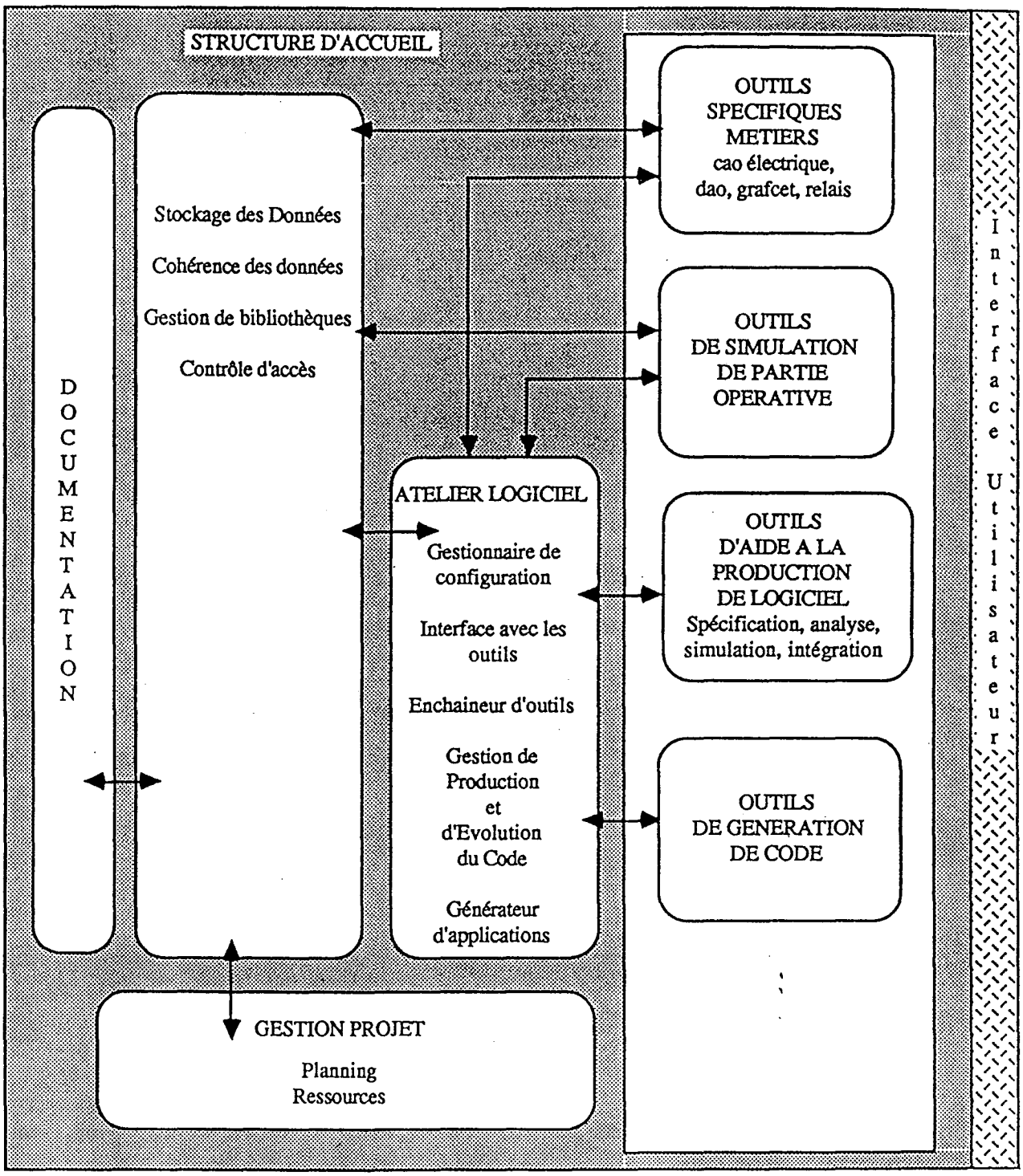


Figure IV.1. : Structure d'un Atelier pour le Génie Automatique d'après [3IP 88]

* ELEGA-O (Environnement Logiciel Expérimental pour le Génie Automatique : "Intégration des Outils"), prototype réalisant une **communication** entre outils de CAO hétérogènes par l'intermédiaire de **standards d'échanges** [FRA 90] [KOE 90] :

ELEGA-O est une **Chaîne Intégrée** de trois outils logiciels utilisant deux standards d'échanges pour leur inter-connexion (Figure IV.2). Ces trois outils sont P4, console de développement et de programmation d'automates CEGELEC, XELEC, outil de conception en schématique électrique et GRAL, outil de conception et codage multi-cible d'applications d'automatismes. Les outils communiquent par "*import/export*" d'informations dans un fichier "**neutre**" dans un format respectant la normalisation SET (Standard d'Echange et de Transfert). XELEC fournit les interfaces d'entrées/sorties du système à GRAL, qui décrit les comportements de commande avant de passer à la programmation et aux tests sur la console P4.

ELEGA-O doit être complété, à terme, par ELEGA-B qui intègrera l'ensemble des informations dans une base de donnée commune et reprenant le schéma conceptuel BASE-PTA. Un dernier niveau, ELEGA-G, devrait permettre ensuite de gérer les outils et les conflits éventuels, en fonction des choix méthodologiques des utilisateurs.

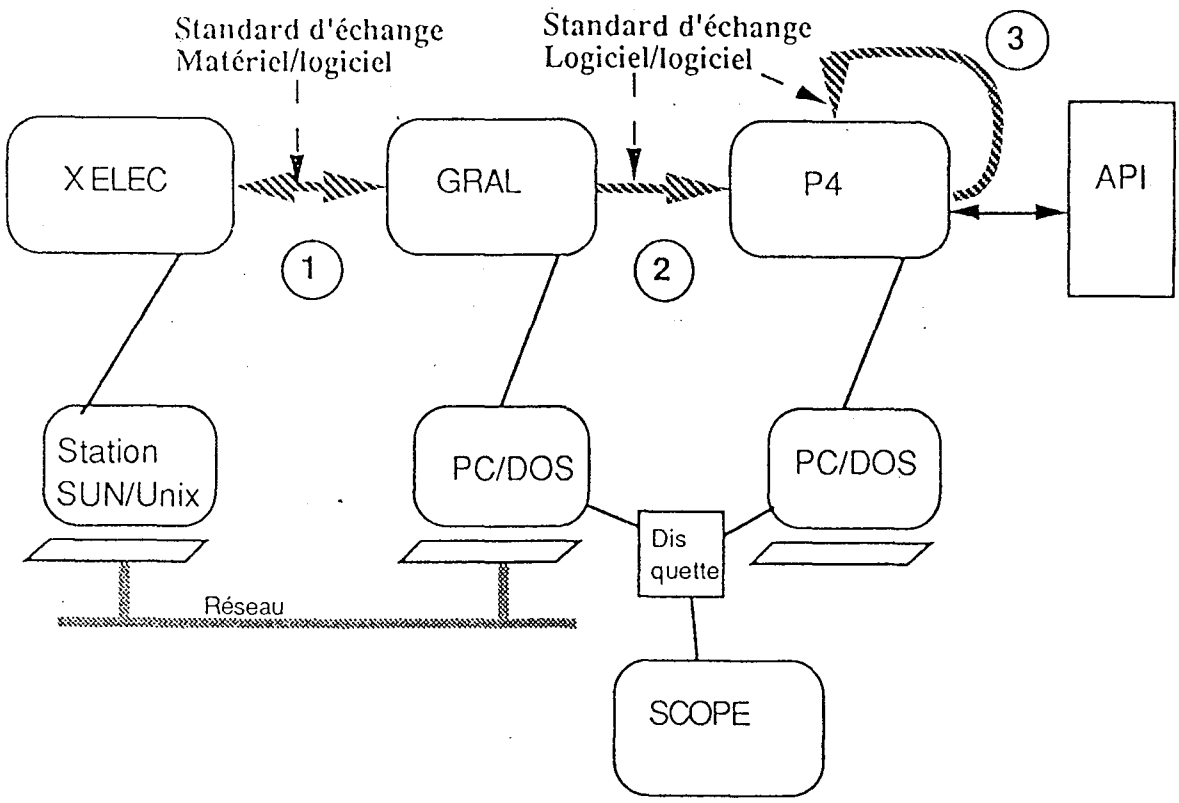


Figure IV.2. : Configuration matérielle pour ELEGA-O [KOE 90]

3. Notre approche pour un Atelier de Génie Automatique Particulier

3.2. Perspectives d'intégrations

3.2.1. Intégration "faible" des outils

La première étape pour une intégration est d'offrir des outils **faiblement intégrés**, c'est à dire ayant chacun leur propre système d'information et communiquant entre eux par l'intermédiaire d'un **"fichier neutre" commun**. Ce type d'intégration, si elle est préférable à des outils totalement déconnectés, duplique les données et laisse aux outils, la charge de la cohérence des informations qu'ils génèrent (Figure IV.3). A l'image du Génie Logiciel [GIR 90], ce type d'échange s'avère très coûteux du fait de la multiplicité d'interfaces nécessaires en fonction du nombre et du type d'outils inter-connectés. Ces interfaces, spécifiques deux à deux, offrent très peu de possibilité de réutilisation et croissent avec le nombre d'outils à faire communiquer [FRA 90].

Dans le cas de la connexion entre les outils GRILLE et SPEX qui nous concerne, le **fichier de données** généré par l'outil GRILLE met à la disposition de SPEX l'ensemble des labels et variables associées d'entrées et sorties utilisées pour la définition des connexions entre exemplaires de comportements génériques. En fait, nous n'effectuons ici qu'un **interfaçage** homogène entre deux outils et non une intégration "faible" car les données créées par l'un des outils et consommées par l'autre sont de nature fugitives, en ce sens où elles sont stockées sur un fichier propre mais ne sont pas centralisées dans un fichier "neutre" commun. Cet interfaçage s'est avéréer réalisable par l'utilisation, pour les deux outils cités, d'un même support physique et d'un même environnement de développement.

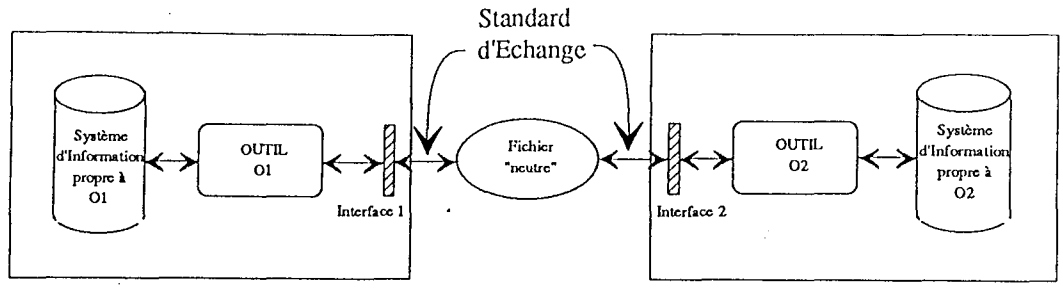


Figure IV.3. : Echange entre deux outils O1 et O2 par fichier "neutre"

3.2.2. Intégration "forte" des outils

La démarche naturelle consiste à "**regrouper**" les outils nécessaires de manière cohérente par rapport aux objectifs d'une démarche d'automatisation. Nous avons vus qu'une première intégration, de type "**Boîte à Outils**", consiste à offrir une **chaîne intégrée d'outils**, sur un même support mais cela nécessite une réalisation de multiples **passerelles** dépendantes des outils connectés. Cette solution est envisageable mais doit, à terme, permettre l'**intégration** d'un ensemble d'outils autour d'une base de données commune telle que le prouvent les travaux sur les **Ateliers de Génie Logiciel** et les travaux actuels du **Centre Coopératif pour le Génie Automatique (CCGA)** [KOE 90] [FRA 90] (Figure IV.4). Cette intégration doit éviter cette multitude d'interfaces mais nécessite, pour être efficace, que les outils soient implantés sur une même structure d'accueil informatique. Les outils intégrés doivent être conçus, dès l'origine, de manière à pouvoir **communiquer** d'une manière **concurrentielle et simultanée**, avec tous les outils concernés par les données communes. Il est nécessaire, pour réaliser cette intégration, d'extraire de chaque outil son **méta-modèle** pour définir, au sein d'un **Atelier de Génie Automatique**, un **Système d'Information ou d'Objets** commun.

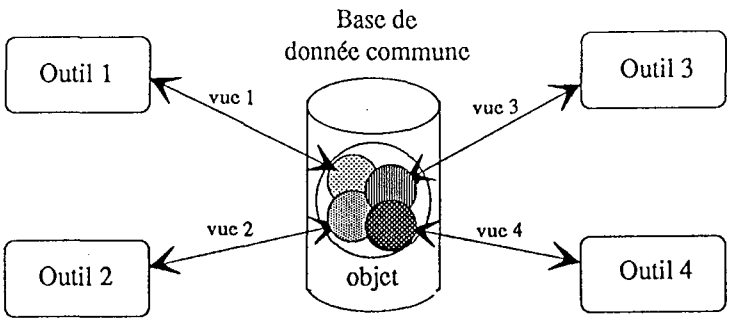


Figure IV.4. : Partage des données entre plusieurs outils par l'utilisation d'une base de donnée

L'évolution des systèmes automatisés nécessite un développement en équipe où chaque intervenant apporte sa connaissance du problème suivant un point de vue donné. Il est utopique d'espérer que chaque personne puisse parler le même langage (malgré le fait qu'ils parlent la même langue) et ainsi fournir en amont toutes les connaissances issues de son expertise sans omission ni ambigüité.

L'intégration des outils, supports des différentes méthodes et méthodologies doit permettre de concentrer l'information provenant des différents intervenants, chacun ayant son formalisme particulier de description, dans une **Base de Données (ou d'Objets)** commune à l'ensemble des outils. Ces objets seront "vus" et exploités par chaque outil en fonction de leur point de vue propre de modélisation.

1. Le modèle conceptuel "objet" d'un Atelier de Génie X

1.1. Les différents niveaux sémantiques de modèles

Le modèle conceptuel "**objet**" est un modèle **réflexif** [AND 88], c'est à dire qu'il se décrit par lui-même à partir de ses propres concepts s'appliquant à tous les

objets qu'il manipule. Il est architecturé selon quatre niveaux en fonction du niveau sémantique des objets manipulés [MEG 89] :

- le niveau "**occurences**" : De façon générale, une occurence (ou instance) identifie un élément dans un ensemble. La relation entre une occurence et son ensemble est caractérisé par une particularisation de cet ensemble. Cette relation est appelée "instanciation" dans certains Langages Orientés Objets (LOO). Elle diffère du processus de "duplication" en ce sens où chaque occurence doit être identifiée et caractérisée par rapport à sa classe d'appartenance indépendamment des autres instances présentes dans le système. Des occurences existent à chaque niveau sémantique du modèle conceptuel mais ne représenteront, pour notre part, que les entités élémentaires manipulées par l'application et ne pouvant plus être instanciées.

- le niveau "**modèle**" : Un modèle est un "moule" à partir duquel nous pourrons créer des occurences. C'est donc un ensemble d'occurences ayant la même structure mais dont les caractéristiques sont différemment valuées. Un modèle est nommé différemment selon les domaines d'application le mettant en oeuvre : "**classe**" dans certains LOO [MAS 89], "**individu**" dans l'outil MEGA [MEG 89], ...
  Le niveau "modèle" ne représentera, dans notre approche, que le niveau immédiatement supérieur du niveau "occurences", c'est à dire les ensembles d'occurences élémentaires du système.

- le niveau "**méta-modèle**" : Le niveau "méta-modèle" est aux modèles ce que le niveau modèle est aux occurences. Un méta-modèle représente la structure générique d'un modèle et permet, par instanciation, de créer des modèles particuliers. Il décrit, de plus, les liens existants entre plusieurs modèles dont il est le "moule". Il caractérise, pour les occurences terminales, le support "sémantique" de leurs relations mutuelles alors que le modèle exprime leurs relations "mécaniques". Il correspond à la notion de "**métaclasse**" dans certains LOO et d' "**individu type**" dans MEGA.

- le niveau "**méta-méta-modèle**" : Le méta-méta-modèle décrit la structure minimale des méta-modèles ainsi que leurs relations et permet la

construction de méta-modèles adaptés à des besoins particuliers. Il est la base de tout le modèle conceptuel. Il est nommé "**classe Métaclass**" dans les LOO et "**segment**" dans MEGA.

Nous pouvons faire cette analogie d'après le modèle chimique:

Le **méta-méta-modèle** identifie les "éléments physiques de base (protons, neutrons, électrons, ...)" ainsi que les lois physiques permettant de construire des "atomes" (**les méta-modèles**) qui, en relations, forment des "molécules" (**les modèles**) avec lesquelles nous fabriquons de la "matière" (**les occurences**).

En poussant plus loin cette analogie, nous pouvons remarquer que le méta-méta-modèle est caractérisé par la nature même des objets manipulés, chimiques ou organiques, pour lesquels il est totalement différent et dépendant. Le "niveau modèle" peut, éventuellement, définir un **comportement dynamique** pour chacune de ses occurences représentant la "vie" et la "mort" de chacune d'elles, ce qui est le cas, dans la nature, pour les composants organiques évolués.

1.2. Le modèle conceptuel objet d'un langage orienté objet

Nous prendrons pour exemple Smalltalk80 [GOL 83], le "père" de tous les Langages Orientés Objets, qui est de plus, celui qui formalise le mieux les concepts attachés à la notion d'objet, c'est à dire les notions de classe, d'instanciation et d'héritage ainsi que le concept de **métacircularité** qui en fait un modèle réflexif où tout est objet.

Smalltalk80 est basé sur les concepts de classes et de métaclasses. Il est construit autour de deux arbres :

- l'arbre d'héritage définit la hiérarchie des classes et des sous-classes héritant de leur classe mère, la classe "**Object**" étant la racine de l'arbre;
- l'arbre d'instanciation décrit la relation d'appartenance de chaque objet à une classe, la classe "**Métaclass**" étant la racine de l'arbre.

La figure IV.5 décrit une partie de l'arbre d'héritage de Smalltalk80. La relation "*est sous classe de*" est représentée par l'indentation. Les métaclasses sont nommées par le nom de la classe suivi du terme "class", par exemple la métaclasse de la classe "Boolean" s'appelle "Boolean class".

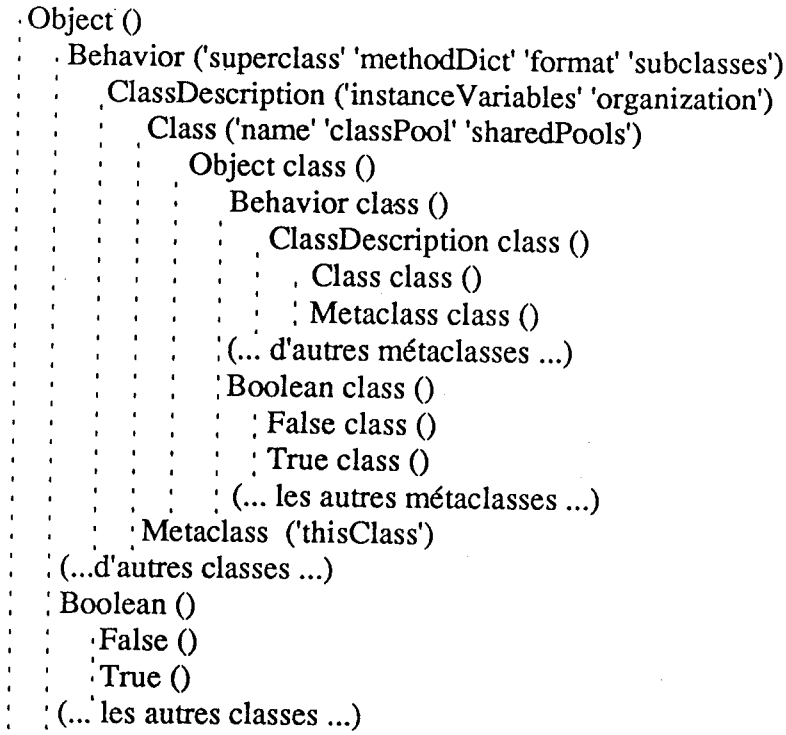


Figure IV.5. : L'arbre d'héritage de Smalltalk80

La figure IV.6 représente l'arbre d'instanciation de Smalltalk80 dans laquelle la relation "*est instance de*" est matérialisée par une flèche orientée. Nous remarquons que la notion de métacircularité implique que la classe "Métaclass", racine de l'arbre, est instance de la métaclasse "*Métaclass class*" et inversement (flèche bidirectionnelle).

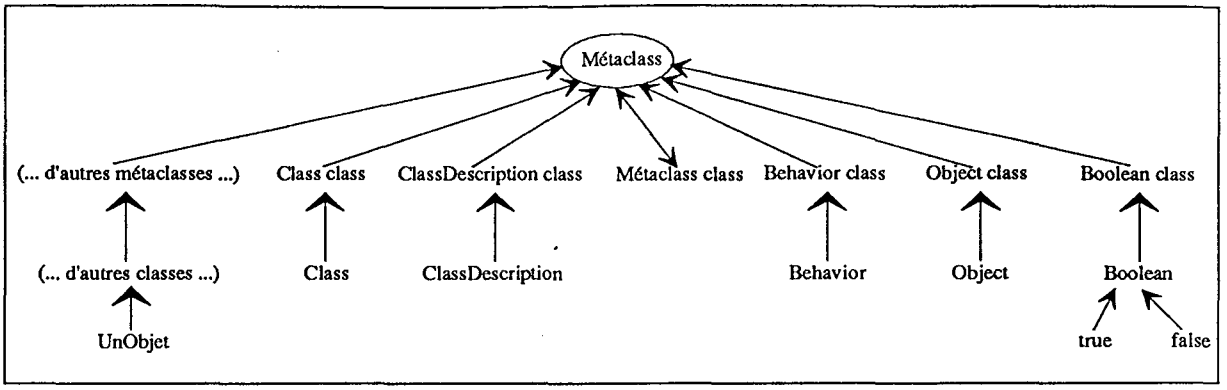


Figure IV.6. : L'arbre d'instanciation de Smalltalk80

Smalltalk80 est un modèle objet réflexif. Il se décrit par lui même à partir de 5 classes principales et leur métaclasses associées :

- la classe "Object", racine de l'arbre d'héritage, décrit le "comportement" minimal nécessaire à toutes les classes du système,
- la classe "Behavior" contient, pour chaque classe, son comportement privé ainsi que sa localisation dans la hiérarchie du système (sa classe mère et l'ensemble de ses classes filles),
- la classe "ClassDescription" décrit, pour chaque classe, la structure de données privées, c'est à dire les attributs caractéristiques de chaque instance,
- la classe "Class" décrit la structure de données globales, c'est à dire les attributs dont la valeur est identique pour toutes les instances d'une même classe.
- la classe "Métaclass" implémente les primitives système nécessaires à la création d'une instance.

Une modification apportée à l'une de ces classes permet la reconfiguration, voire la destruction, du système complet. Ces classes décrivent le **méta-méta-modèle** de Smalltalk80 et ne doivent être modifiées que par une personne avisée que nous appellerons le "méta-concepteur" ou concepteur du langage.

Toute classe du système est une instance d'une métaclasse décrivant son comportement, par exemple l'initialisation de ses variables. L'ensemble des métaclasses décrivent le **méta-modèle** de Smalltalk80.

La création d'une application dans l'environnement objet consiste à définir un ensemble de classes représentant les objets manipulés. Cet ensemble décrit un **modèle** relatif aux besoins de l'utilisateur. Du point de vue de ce dernier, ce modèle est **l'applicatif** lui permettant de concevoir sa propre application par la création d'instances (**occurences**) des classes définies par le concepteur applicatif.

Nous retrouvons bien, comme présentés sur la figure IV.7, les quatre niveaux **sémantiques** du modèle Objet ainsi que les intervenants concernés par les différents domaines d'utilisation du système.

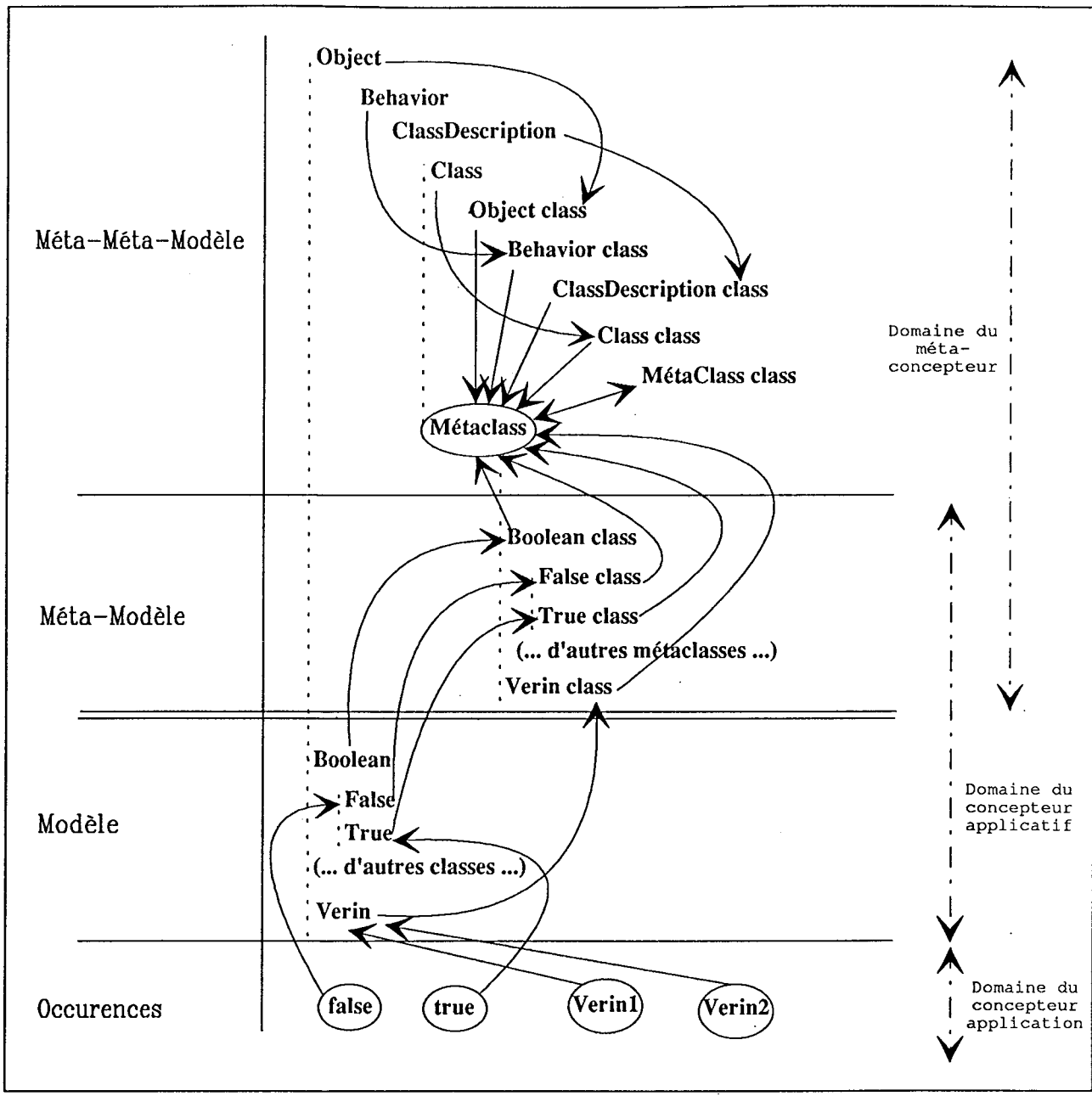


Figure IV.7. : Le modèle conceptuel Objet de Smalltalk80

1.3. Le modèle conceptuel objet de MEGA

MEGA [MEG 89] est un atelier de conception et de prototypage des Systèmes d'Information (SI). Il est bâti autour de la méthode "MERISE étendue" et permet la modélisation d'un Système d'Information suivant trois niveaux :

- le niveau conceptuel
- le niveau organisationnel
- le niveau logique

A chaque niveau correspondent des modèles décrivant des points de vue différents (activités, communications, données, traitements, ...).

Chaque modèle est associé à un ou plusieurs "segments" de la base de donnée MEGA. Un segment est une structure de donnée composée d'une liste d'informations et en relation avec d'autres segments, par exemple individu, outil, opération, ...

Le modèle conceptuel de MEGA (figure IV.8) est un modèle réflexif architecturé autour de quatres niveaux :

- le niveau **méta-méta-modèle** est composé de trois segments "spéciaux" appelés ".SEGMENT", ".LIEN", ".ATTRIBUT" utilisés pour décrire la structure même de la base de donnée MEGA. Ces segments sont "au-dessus" des autres segments et liens de la base, c'est à dire chaque segment particulier de la base est un exemplaire de l'un d'eux. Ils constituent la structure physique du SGBD MEGA et permettent sa modification.

- le niveau **méta-modèle** est le domaine du concepteur applicatif. Il est constitué de l'ensemble des "segments types" (individu type, message type, ...) occurences du segment ".SEGMENT" et décrit les propriétés (attributs) des différentes entités que peut manipuler l'utilisateur final pour la définition de son propre modèle (son application). L'ensemble des "types" peut être architecturé selon un arbre de **sous-typage** réalisant une relation "est sous ensemble de" entre un sous-type et son type père. Un type décrit, en outre, les relations (liens) entre plusieurs occurences issues de lui-même. Les liens et les attributs sont des occurences respectivement des segments ".LIEN" et ".ATTRIBUT" du méta-méta-modèle.

Ce niveau décrit un **Système d'Information de Référence** pour un **domaine d'application particulier.**

- le niveau **modèle** représente la structure de l'application réalisée par un utilisateur particulier, le concepteur application. Il est constitué de l'ensemble des **classes d'entités** nécessaires à l'application, par exemple "Outil", "Pièce", "Programme". Les relations entre ces différentes entités sont décrites au niveau méta-modèle.
Ce niveau particularise le Système d'Information de Référence du domaine pour une **application particulière.**

- le niveau **occurences** est constitué de l'ensemble des occurences de classes d'entités modélisant un Système d'Information **particulier** à une entreprise.

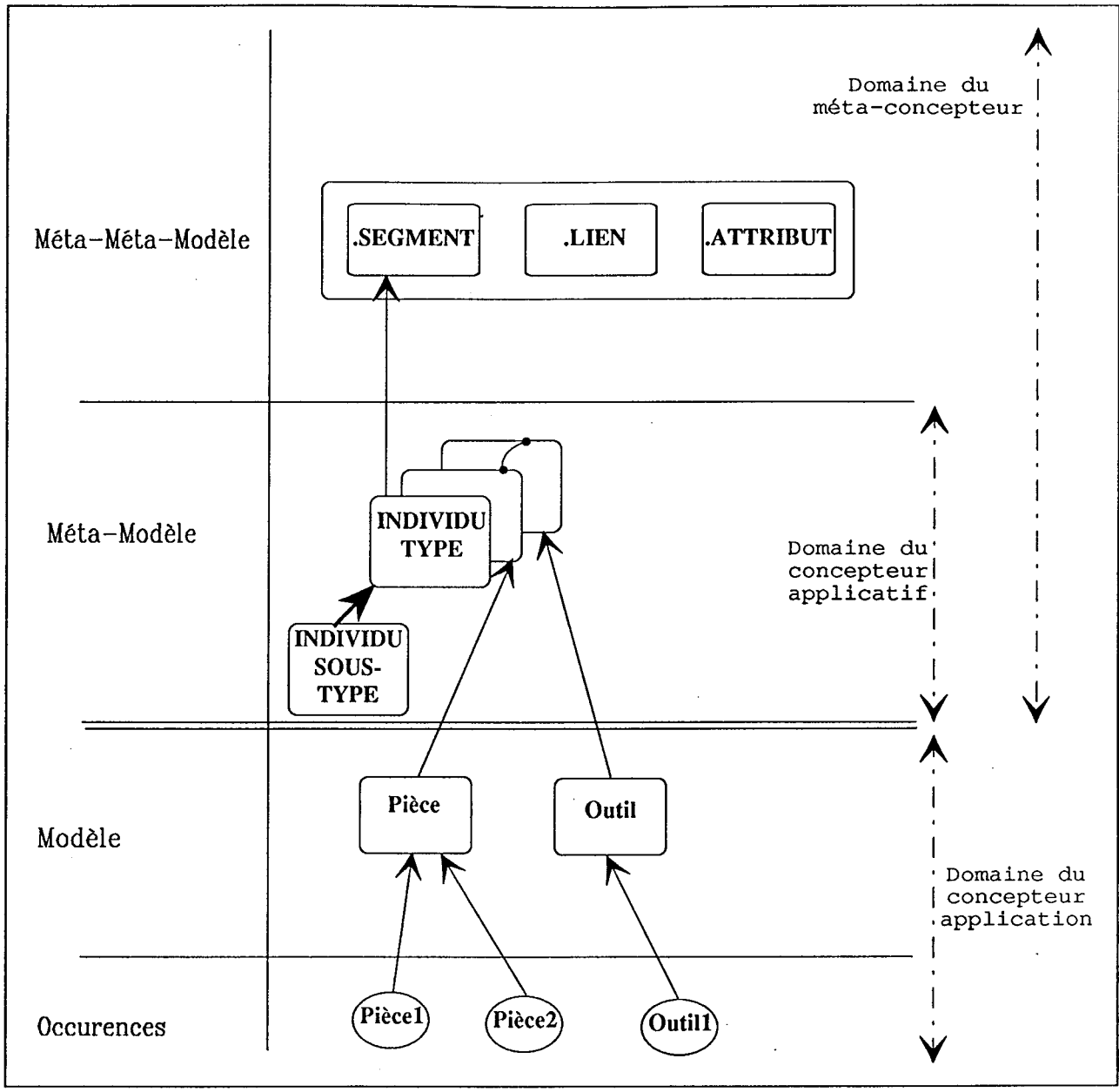


Figure IV.8. : Le modèle conceptuel Objet de MEGA

1.4. Vers un modèle conceptuel d'un Atelier pour le Génie Automatique

Définitions préliminaires

Nous allons, dans un premier temps, donner nos définitions concernant les termes de **formalisme, méthode** et **méthodologie,** termes souvent utilisés mais dont l'interprétation dépend du domaine d'application [ALA 88b].

Un **formalisme** est une représentation, de préférence graphique, exprimant un point de vue sur la solution d'un problème donné. Il utilise un certain nombre de notations, de préférence proches des représentations mentales et métiers des utilisateurs, identifiant des entités conceptuelles, logiques ou physiques, statiques ou dynamiques ainsi que les relations entre les entités. Un formalisme est associé à un modèle syntaxique définissant des règles d'utilisation et d'assemblage.

Une **méthode** est un ensemble de règles à appliquer pour traîter un problème. Elle se concrétise par un ensemble de techniques d'utilisation de notations ainsi que par une **interprétation sémantique** des objets manipulés. Elle peut être automatisée par un **outil** informatisé évitant les écueils couramment révélés par les "méthodes papier".

Une **méthodologie** est **une** méthode d'établissement de nouvelles méthodes ou **une** méthode de choix des "bonnes" méthodes à utiliser pour traîter un problème donné. Elle exprime aussi les liens entre plusieurs méthodes, ainsi que les points de vue associés et les informations échangées. Elle peut être supportée par un "**atelier**" informatisé mettant en oeuvre un ensemble d'outils-méthodes et architecturé autour d'un **Système d'Informations** commun.

Le modèle conceptuel proposé

La Figure IV.9 représente une proposition pour un **modèle conceptuel d'atelier de Génie X** dont l'adaptation au Génie Automatique dépend d'une part du choix des outils, supports des différentes méthodes utilisées par le concepteur automatiseur, et d'autre part, de la méthodologie appropriée décrite dans le méta-méta-modèle.

Le **méta-méta-modèle** de l'atelier définit en outre le protocole d'accès à la **Base de Donnée ou d'Objets** pour tous les outils qu'il intègre. Il représente les liens relationnels entre les outils-méthodes logiciels correspondant aux méta-modèles.

Le **méta-modèle** établit, pour chaque outil, les règles d'utilisation des différentes méthodes associées ainsi que le protocole d'échange et la cohérence globale des informations manipulées entre les différents points de vue modélisés au sein d'un outil particulier, ceci à l'aide d'un formalisme adapté.

Les **modèles** permettent au concepteur application de manipuler des formalismes appropriés pour l'expression d'une solution particulière de son problème suivant plusieurs **points de vue** et des degrés de précision variés.

Pour une **application particulière** d'Ingénierie Productique, il est nécessaire d'extraire de chaque outil son "**méta-modèle**" pour un déduire le **Système de Gestion d'Objets** de l'Atelier le plus adéquat. Il est aussi nécessaire de permette l'accès au **méta-méta-modèle** de l'atelier pour permettre la modification du **méta-modèle des outils** afin d'adapter la méthode supportée à la sémantique des objets manipulés et à la méthodologie d'utilisation des outils [MOR 88].

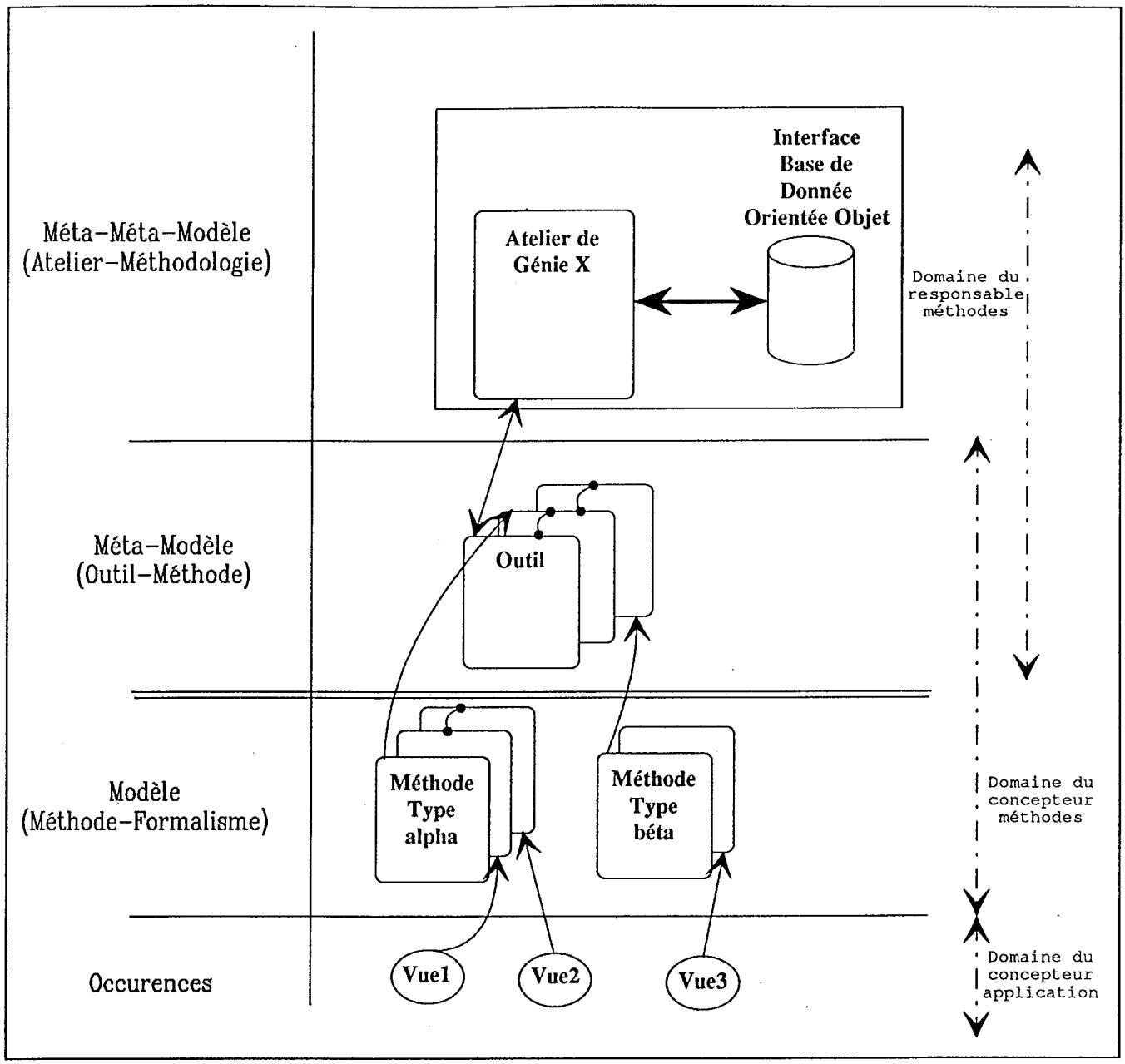


Figure IV.9. : Le modèle conceptuel d'un Atelier de Génie X

3.2.3. Le projet d'intégration

En conception de la commande sur l'outil SPEX, la vérification des comportements est réalisée par une simulation et une animation des graphes de commande. La validation de la conduite nécessite une interprétation des différents états des comportements et serait plus "visuelle" s'il était possible d'animer, en parallèle, une vue technologique des éléments commandés. Deux possibilités sont envisagées pour offrir cette fonctionnalité à SPEX :

- la première possibilité consiste à développer une fonction d'édition et d'animation de synoptiques sur l'outil SPEX et fournir, ensuite, des post-processeurs pour le transfert des graphiques et des données vers des superviseurs particuliers en évitant ainsi toute resaisie des mêmes graphiques sur les dits superviseurs. Cette solution est technologiquement réalisable mais nécessite un investissement logiciel important ;

- la deuxième possibilité tend à réutiliser l'existant, c'est à dire à utiliser, en conception, les fonctionnalités d'édition et d'animation d'un superviseur industriel particulier. SPEX ne fournirait donc, en cours de simulation, qu'un ensemble de données permettant l'animation des synoptiques édités sur le superviseur.

C'est cette deuxième possibilité qui sera retenue pour une connexion avec le superviseur CIMVUE [CIM 91].

CIMVUE est un superviseur de procédés industriels permettant l'édition et l'animation de synoptiques. L'animation est utile pour la conduite de l'installation mais elle l'est aussi en phase de maintenance pour l'identification des défaillances éventuelles.

En phase de maintenance, l'identification des blocages éventuels de la commande suite à des défaillances est très importante mais elle n'est exploitable de façon pratique que si, en parallèle d'une visualisation de l'application d'un point de vue technologique, il soit possible d'animer les graphes de commande qui ont servis à la conception générale. Dans cette optique, et pour maintenir une cohérence entre les modèles visualisés et ceux créés dans SPEX, nous transfèrerons les graphes de SPEX vers le superviseur (Figure IV.10) pour qu'il puisse les utiliser sans avoir donc à les redécrire.

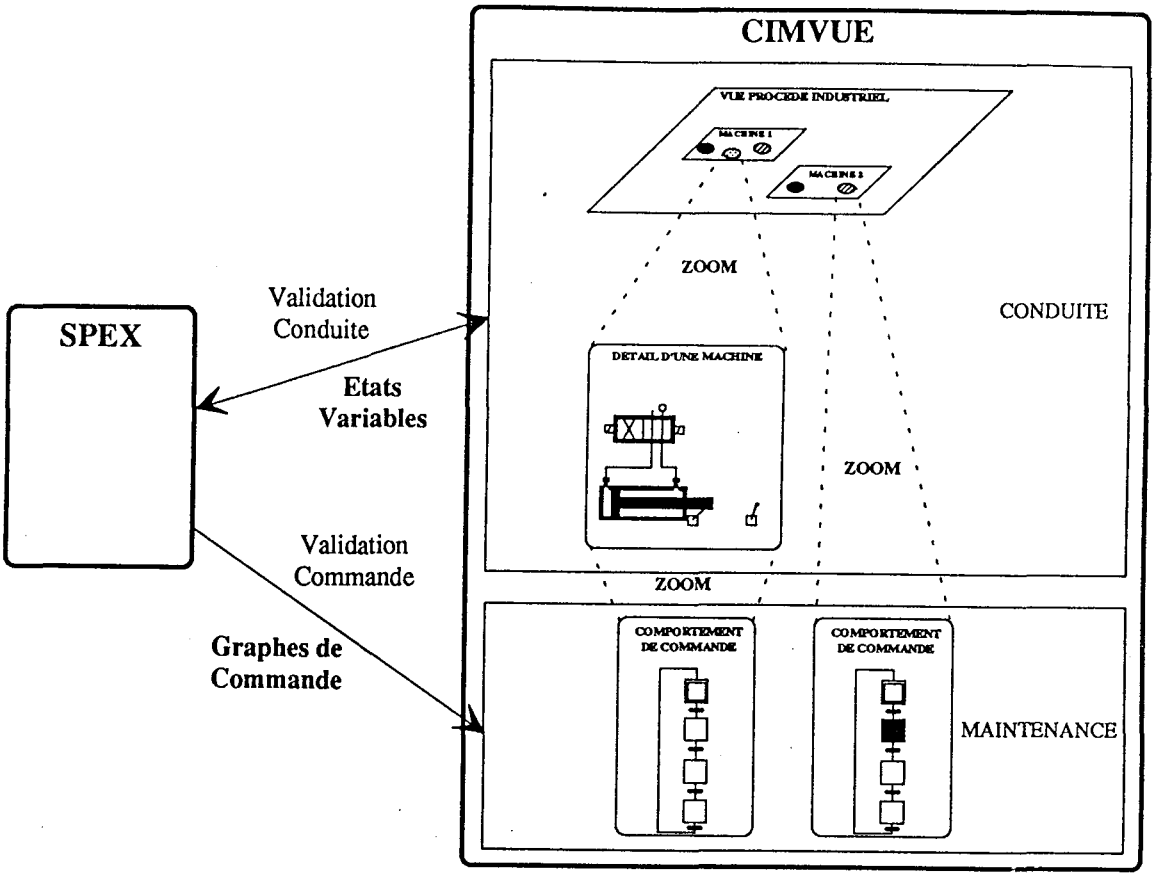


Figure IV.10. : Les échanges d'informations entre SPEX et CIMVUE

Notre objectif d'intégration doit se concrétiser, dans le cadre d'une collaboration avec la société SGN Lorraine et le Laboratoire d'Automatique et de Micro-électronique de Montpellier, par une intégration de trois outils :

- SPEX , présenté au chapitre I, pour la conception générale et détaillée,
- CADEPA [SGN 90] pour la génération de code multi-automate,
- CIMVUE [ARC 91] pour la supervision de procédés industriels.

Nous avons montré, précédemment l'utilité et la faisabilité d'une connexion entre SPEX et CADEPA. Nous effectuerons, dans un premier temps, une **connexion** entre ces outils avec une ouverture possible pour d'autres outils (Figure IV.11) tels que, par exemple, Orchis [TNI 89] en amont, MAXSIM [MAX 91] en aval et la nouvelle chaîne PIASTRE en cours de réécriture avec le langage Smalltalk80, pour ensuite les **intégrer** autour d'une base de donnée commune.

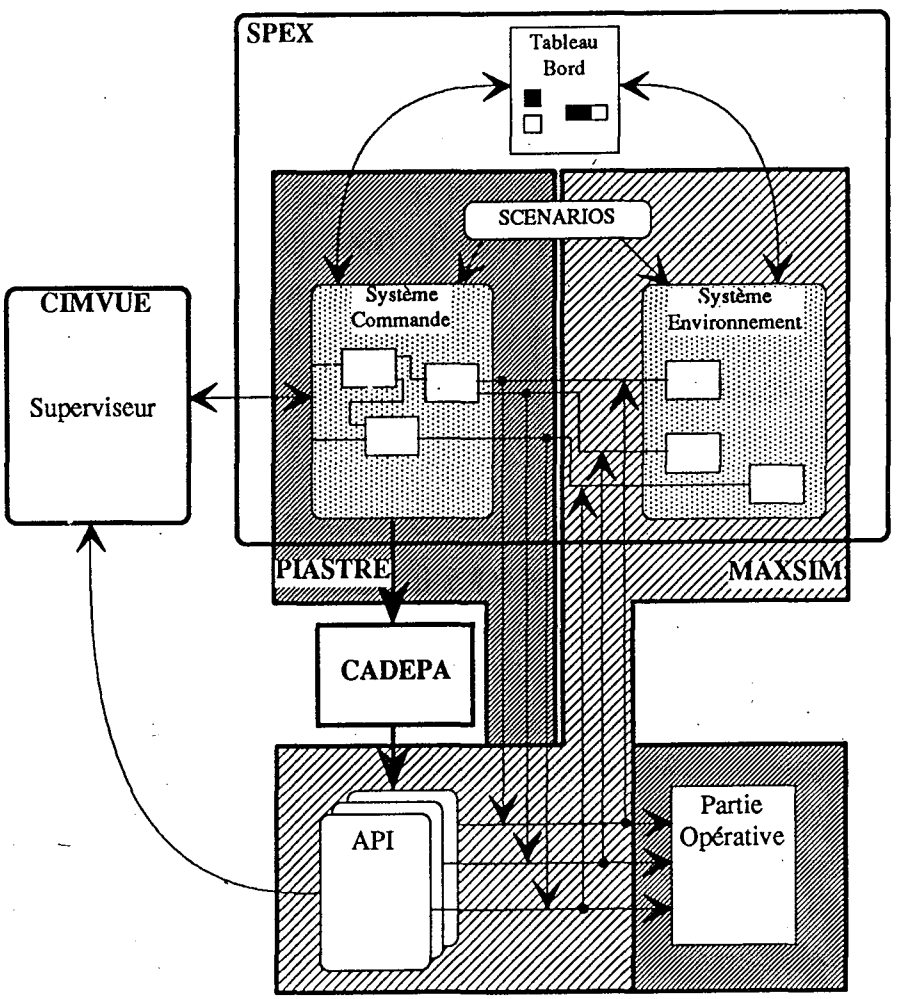


Figure IV.11. : Perspective d'intégration faible d'outils

Dans une première phase, une connexion par un transfert des graphes sera réalisée entre les outils CADEPA et CIMVUE. Par la suite, lorsque SPEX (qui est encore un prototype) deviendra un produit, les connexions sus-citées seront mises en oeuvre.

Par la suite, pour permettre l'intégration de tous ces outils autour d'une base de donnée et ainsi offrir un Atelier de Génie Automatique particulier, il sera nécessaire d'extraire le méta-modèle de chacun d'eux pour déterminer le type et la structure de la base de donnée (ou d'objets) ainsi qu'une caractérisation des "objets" qu'ils manipulent.

Le choix d'une base de donnée (ou d'objets) adaptée à nos besoins s'avère un point crucial. Nous avons donc fait l'acquisition de quatre Systèmes de **Gestion de Base de Données** (SGBD) ou d'**Objets** (SGBO) différents, GEMSTONE [GEM 89], EMERAUDE [EME 90], ORACLE [ORA 90] et SQL-SERVER [SQL 89] ainsi que deux ateliers de conception de Systèmes d'Informations, PC-IAST [PCI 90] et MEGA [MEG 39], que nous allons pouvoir évaluer et comparer en fonction de nos besoins. Ces choix ne sont pas le fait du hasard pour les raisons suivantes :

- GEMSTONE, développé par la société Servio Corporation (USA) est un SGBO qui implémente tous les concepts propres aux approches objets (encapsulation, classe, héritage, instanciation, envois de messages) ainsi que les mécanismes de base d'un SGBD classique multi-utilisateur (protection, concurence d'accès, ...). Notre choix s'est porté sur ce SGBO car il s'interface naturellement avec le langage Smalltalk80, langage de développement du prototype SPEX, ce qui nécessite ainsi relativement peu de modification à SPEX pour pouvoir l'intégrer. GEMSTONE s'interface aussi avec le langage C, langage cible pour le produit final SPEX.

  GEMSTONE peut aussi s'interfacer avec le système expert orienté objet NEXPERT OBJECT [NEX 88] qui pourrait être un autre outil de l'atelier offrant une aide aux choix des méthodes et des modèles utilisés par le concepteur.

- EMERAUDE, développé par le GIE EMERAUDE, est une implémentation de la structure d'accueil PCTE (Portable Common Tools Environment), normalisée au niveau européen [PCT 88] et conçue pour servir de base à la construction d'Ateliers Intégrés de Génie Logiciel. la base de PCTE est son **Système de Gestion d'Objets**, entité sur laquelle peuvent travailler les utilisateurs et les programmes. Son modèle sous-jacent est dérivé du modèle Entité-Association de Chen [CHE 76] et permet la définition de types d'objets, de types de liens et de types d'attributs.

  La réalisation d'un **Atelier de Génie Automatique** sur la base de cette structure d'accueil nous paraît appropriée d'autant que les résutats d'un projet Esprit nommé PACT (PCTE Added Common Tools) [PAC 86] apporte un ensemble d'outils de développement et de services communs (gestion de versions, gestion de dialogues, manipulation de données, ...) aux utilisateurs de PCTE.

- ORACLE est un ensemble d'outils de conception et de développement de Systèmes d'Information architecturé autour d'une Base de Donnée Relationnelle à **accès répartie**, c'est à dire dont les données sont accessibles par un ensemble hétérogène de stations via un réseau.

  Il peut s'interfacer avec un grand nombre de langages de programmation pour la création d'applications utilisants ses données.

- SQL-SERVER est un SGBD relationnel multi-utilisateurs développé par Microsoft et Ashton-Tate et conçu pour le traitement transactionnel, garantissant la cohérence et la recouvrabilité de la base de données. Il est basé sur une architecture client-serveur fonctionnant sur réseaux et permettant la distribution des données sur plusieurs serveurs indépendants, et ceci de façon transparente pour l'utilisateur. Un grand nombre d'applications existantes peuvent s'interfacer avec la base ainsi que toutes applications utilisant ses bibliothèques C. Une version fonctionnant sur matériel SUN, support matériel actuel de SPEX, existe, et permettrait ainsi l'intégration de ce dernier avec des applications compatibles SQL-SERVER.

- PC-IAST, développé par CONTROL DATA est un logiciel d'aide à l'analyse des informations et à la conception de Systèmes d'Information par la méthode NIAM [NIJ 80] générant un modèle neutre directement assimilable par de nombreux SGBD (ORACLE, INGRES, ...). Cet outil a servi de support à la définition du système d'information de CIMVUE et peut donc nous fournir une aide à l'identification du méta-modèle du superviseur pour son intégration dans l'Atelier de Génie Automatique particulier.

- MEGA est un Atelier de Conception des Systèmes d'Informations utilisant la méthode "MERISE étendue" [TAB 88]. Il est entièrement paramétrable par la notion de méta-méta-modèle et peut générer un prototype de l'application pour vérifier ses spécifications.

II. CONCLUSIONS DU CHAPITRE IV

"**Intégrer**" est le maître-mot des années 90 et les progrès rapides de l'informatique devraient permettre rapidement la création d'un Atelier de Génie Automatique. Mais encore faut-il savoir **quoi intégrer** et **comment intégrer**. En effet, nous sommes à l'aube de la banalisation d'outils logiciels dans les entreprises mais les utilisateurs, de par leur manque de formation et de formalisation, sont encore à la préhistoire de l'utilisation de tels ateliers. Il reste peut-être beaucoup plus à faire dans le domaine de leur formation méthodologique que dans la mise à leur disposition d'outils d'aide au développement de Systèmes Automatisés qui, créés individuellement ou n'admettant pas d'ouverture à d'autres outils, contribuent à la désintégration plutôt qu'à **l'intégration**.

A terme, la maxime "**Zéro-Papier**" devrait être une réalité et on ne devrait plus communiquer que par l'intermédiaire de dossiers informatisés incluant, non seulement les documents écrits de spécification et de conception pour rassurer, mais aussi et surtout les maquettes et prototypes exécutables ayant permis leur validation. Par exemple, e client pourraient ainsi comparer, en dynamique, les résultats de la conception avec ses besoins réels et cela avant codage.

A terme, il est certain que l'apport d'outils comme SPEX et leur intégration au sein d'un Atelier de Génie Automatique tendra à inverser la répartition des efforts de conception des Machines et Systèmes Automatisés de Production au profit des phases d'analyse.

Il est aussi indéniable que la concentration du **savoir-faire de l'entreprise**, tant d'un point de vue informationnel que des démarches méthodologiques, au sein d'une base commune et sa mise à disposition tout au long du cycle de vie d'un système automatisé, depuis sa spécification jusqu'à son exploitation et sa maintenance, doit permettre une amélioration des gains en productivité, ainsi que de la qualité et de la maîtrise des Systèmes Automatisés de Production Manufacturière.

Conclusions

CONCLUSIONS

Les apports des concepts liés au **Génie Logiciel** sont indéniables mais ne suffisent pas pour résoudre tous les problèmes du **Génie Automatique**. Il ne faut pas oublier que le domaine de l'automatisation manipule des objets logiciels ayant un lien **indissociable** avec des objets matériels et qu'une démarche d'**Automatisation Intégrée** doit être supportée par des outils-méthodes riches d'un point de vue "**sémantique**". L'apport de SPEX pour les aspects syntaxiques des objets d'automatisation permet de régler le problème le plus "**mécanique**" de la manipulation de ces objets. Ainsi, la possibilité d'exprimer "la **forme**" des objets d'automatisation et le fait de les utiliser dans un grand nombre d'applications aident à communiquer les **idées** et à présiser la ou les sémantiques de ces objets. L'"**idée**" ne se communique que par sa" **forme**" [CLA 79].

> [BEN 66] "*Le langage a cette faculté de représenter le réel par un «signe» et de comprendre «le signe» comme représentant le réel, donc d'établir un rapport de signification entre quelque chose et quelque chose d'autre*"

D'autre part, s'il est nécessaire d'avoir accès au **méta-méta-modèle** d'un Atelier particulier pour permettre la modification du **méta-modèle** des outils et ainsi adapter les méthodes supportées à la sémantique des objets manipulés, il conviendrait, dans une première phase, d'appliquer ces concepts aux outils eux-mêmes pour l'adaptation des formalismes aux utilisateurs selon un point de vue **pragmatique**.

L'intégration de tels outils au sein d'un **Atelier de Génie Automatique** n'est pas un simple problème technique, c'est aussi un enjeu économique et surtout social qu'il ne faut pas négliger. L'automatisation de l'automatisation va bouleverser le mode de travail des bureaux d'études concernés et nécessitera, probablement, un temps d'adaptation non négligeable de la part des concepteurs-utilisateurs.

[ALA 88] "*Si les concepteurs de CAO ont un rôle à jouer, adapter ceux-ci à un métier pour les rendre «cerveau-compatible», afin de les intégrer dans un système plus vaste, celui des responsables des bureaux d'études d'automatisation est encore plus crucial : ce sont eux qui devront gérer le changement de technologie*"

En effet, si les techniques actuelles fournissent des **interfaces homme-machine** de plus en plus conviviales et ne nécessitant que peu de connaissances pour une utilisation optimale, il n'en est pas de même pour les démarches supportées par ces outils. Celles-ci, du fait de la multiplicité des notations et des méthodes, nécessitent une formation de plus en plus pointue, ce que les automatiseurs actuels ne sont pas toujours prêts à remettre en cause.

Face aux besoins de maîtriser la **qualité des productions**, la **gestion des versions** et la **documentation**, il devient nécessaire de mettre en oeuvre les techniques informatiques les plus avancées. Il est, pour cela, nécessaire de disposer, pour tous les outils, d'informations intègres et cohérentes centralisées.

Une solution envisageable pour résoudre le problème de la documentation est d'employer les techniques basées sur les **hypertextes**. Ces derniers permettraient de "naviguer" au travers des différents modèles et formalismes, de façon transversale pour tous les outils, et représenteraient les informations selon le point de vue et la notation désirée (et surtout compréhensible) par l'utilisateur. Si cette solution semble intéressante, il subsiste tout de même une difficulté quant à une **bijection totale**, aussi bien au niveau structurel que sémantique et de l'*interprétation* des différents modèles et/ou formalismes employés [DOU 90] [AFC 89].

Au delà de cette ouverture nécessaire des outils, il paraît tout aussi important, tant d'un point de vue "intégration" que de celui d'"utilisation", de leur offrir un **comportement intelligent** [DAV 90].

A l'image des travaux prometteurs sur les futurs **Ateliers de Génie Logiciel** de 4ème génération [THE 88] intégrant l'"intelligence artificielle", les **Ateliers de Génie Automatique** doivent, par la définition d'un méta-méta-modèle "intelligent", prendre en compte des **traitements experts** (systèmes experts) et permettre "*l'élaboration de modèles de données intelligents capables de déduire de nouvelles informations _SGBD déductifs_ , de lancer des traitements, d'assurer l'évolutivité du modèle*" [MOR 90] et ainsi apporter une **aide sémantique adaptée** aux concepteurs des Machines et Systèmes Automatisés de Production.

Devant l'ampleur du travail qu'il reste à faire, nous pouvons dire que le **Génie Automatique** a encore de "beaux jours" devant lui...

Références Bibliographiques

Scientifiques

ABB 86
R.J.ABBOTT
An integrated approach to SOFTWARE DEVELOPMENT
WILEY - interscience publication- JOHN WILEY&SONS 1986

ADE 84
ADEPA
"Apports de la modélisation de la Partie Opérative aux différentes étapes de la vie d'un automatisme
Synthèse des Travaux de la Commission - 1984

AFC 83
AFCET
Rapport du Groupe "Systèmes Logiques" de l'AFCET
Le Nouvel Automatisme - Septembre 1983

AFC 87
AFCET Groupe Systèmes logiques
Formalisation d'extensions du Grafcet (Macro-étape et Forçage)
Document de synthèse - 19 Janvier 1987

AFC 89
AFCET "Collège Automatique et Productique"
Les Langages Pivots Publics pour le Génie Automatique
Compte-rendu de la réunion - 28-29 Septembre 1989 - PARIS

ALA 84
P.ALANCHE, X. DE ROQUEMAUREL, D.MORIN
Le Poste de Travail pour l'Automatisation
Textes des communications du colloque International "Automatique Appliquée"
SEE - NICE - 1984

ALA 86
ALANCHE P., LHOSTE P., MOREL G., ROESCH M., SALIM M., SALVI Ph.
Application de la Modélisation de Partie Opérative à la structuration de la commande
Congrès AFCET "Méthodes et outils modernes de conception et d'exploitation de la commande de procédés discontinus complexes" - Montpellier - 1986

ALA 88a
P.ALANCHE
Automatiser l'automatisation
Revue Automatique et Productique Appliquées - Vol. 1 - N° 2 - 7/14 - 1988

ALA 88b
P.ALANCHE
Les automatismes et leur CAO
Collection Technologies de pointe - Editions Hermès - N° 12 - 1988

ALD 90
M.ALDAMONDO, D.NOYES
Aide à l'intégration des systèmes de production : exploitation d'invariants structurels et formels
Actes des journées CIM 90 - 459/466 - BORDEAUX - 1990

AMA 90
S.AMAR, E.CASTELAIN, J.C.GENTINA
Modélisation des moyens de production par langages orientés objet en vue de la conception de la commande d'un système de production flexible
Actes des joournées CIM 90 - 323/331 - BORDEAUX - 1990

AND 88
J.ANDRE, P.PERRIN
Le Modèle Conceptuel Objet
AFCET - Projet de papier transmis au Groupe de Travail 135 - Avril 1988

BAI 89
S.C.BAILIN
An object-oriented requirements specification method
Communications of the ACM, Vol. 32, n° 5, 608/623 - Mai 1989

BAR 89
F.BARBIER, A.HAURAT
Glossaire des systèmes à objets : application à l'informatisation de la fonction production d'une entreprise manufacturière
Génie Logiciel & Systèmes Experts - N°17 - 88/111 - Décembre 1989

BAS 88
BASE-PTA
Modèle conceptuel des données en automatisation : résultat des travaux du groupe BASE-PTA
Congrès AFCET - GRENOBLE - 1988

BEL 89
A. BELHIMEUR
Contribution à l'étude d'une méthode de conception des automatismes des systèmes de conduite des processus industriels
Thèse de l'Université des sciences et techniques de LILLE FLANDRES ARTOIS Spécialité "Automatique" - 17 Mai 1989

BEL 90
J.BELLARA
Contribution à la gestion de ressources dans un système de production
Thèse de doctorat de l'Université de Montpellier II "Composants, signaux, systèmes"
USTL - MONTPELLIER - 1990

BEN 66
BENVENISTE
Problèmes de linguistique générale
Edition Gallimard - 1966

BEN 89
K.BENALI
Assistance et Pilotage dans le développement de logiciel : Vers un modèle de description
Thèse de l'Université de NANCY I en Informatique - 1989

BOE 78
B.W. BOEHM, J.R.BROWN, M.LIPOW, G.J.MAC-LEOD, M.J.MERRIT
Characteristics of Software Quality
Ed. North Holland Publishing Company - 1978

BOO 86
G.BOOCH
Object-Oriented Development
IEEE Transactions on Software Engineering, Vol. 12, 211/221, Février 1986

BOO 88
G. BOOCH
Traduction de J.P.ROSEN
Ingénierie du logiciel avec ADA
InterEditions - 1988

BOU 90
G.BOULAYE
Le prototypage de logiciels
AFCET/INTERFACES - N° 91/92 - 14 /22 - Mai/juin 1990

BOUa 90
J.P.BOURRIERES, F.LHOTE
Modélisation hiérarchisée des tâches d'assemblage
Actes des journées CIM 90 - 505/511 - BORDEAUX - 1990

BRO 88
J.BRODE, G.H.MALKARY
A research programming support environment for design ans study of distributed control systems
IMACS - PARIS - Juillet 1988

CAL 90
J.P.CALVEZ
Spécification et conception des systèmes. Une méthodologie
Editions Masson - Paris - 1990

CAZ 83
C.CAZALOT
LEDA-MOD : spécification de commande d'automatismes complexes par association de Grafcet, langage et chaîne de CAO associée
Thèse de 3° cycle - MONTPELLIER - 1983

CHE 76
P.P. CHEN
The Entity Relationship Model: toward an unify view of data
ACM/TODS - 1,1 - 1976

CHO 88
C.CHOPPY
Maquettage et prototypage : panorama des outils et des techniques
Génie Logiciel & Systèmes Experts - N°11 - 6 /13 - Mars 1988

CLA 79
J.CLARET
L'idée et la forme
Collection Que sais-je ?
Presses Universitaires de France - 1979

COR 85
CORBIER F.
Simulation pour validation des processus discontinus
Contrat CIFRE, LACN-CRAN/SPIE TRINDEL n° 313/85

COR 88
CORBIER F., EDLINGER A., MOREL G., ROESCH M., ROESCH P.
Procédé pour valider le fonctionnement d'un automatisme et dispositif pour sa mise en oeuvre
Demande de brevet européen - SPIE TRINDEL/LACN - CRAN n° 88400550.5

COU 76
J.COURTES
Intoduction à la Sémiotique narrative et discursive
Collection Langue, Linguistique, Communication
Editions Hachette

COU 87
COURTIAT, DEMBINSKI, GROZ, JARD
ESTELLE, un langage ISO pour les algorithmes distribués et les protocoles
TSI - Vol 6 - N°2 - 89-103 - 1987

COX 86
B.J.COX
Object Oriented Programming, An Evolutionnary Approch
ADDISON-WESLEY - 1986

DAV 90
B.T.DAVID, S.SABOU
Un modèle d'architecture pour les systèmes CAO intelligents
Actes de la 9ème Conférence Internationale MICAD 90 - Vol. 1, 166/181
PARIS - 1990

DEF 86a
J.DEFRENNE
Modélisation de la Partie Opérative; impact sur la sécurité et la maintenance des systèmes à évolution séquentielle
Thèse de Docteur-ès-Sciences - LILLE - 1986

DEF 86b
J.DEFRENNE, J.M.TOULOTTE, R.HACHEMAN
Validation des logiciels de commande des systèmes industriels à evolutions simultanées
Actes de la CONVENTION AUTOMATIQUE PRODUCTIQUE
"Moyens d'automatisation dans les industries manufacturières" - PARIS - 1986

DEM 78
T.DEMARCO
Structured Analysis and System Spécification
Yourdon Press - 1978

DER 86
X. DE ROQUEMAUREL
Expression des besoins
Simulateur de machine commandée par un automate programmable
Citroën Industries DMI/RTN/COR/86 - 1986

DOU 90
G.DOUMEINGTS & al.
Méthodes pour concevoir et spécifier les systèmes de production, ...
Actes de CIM 90 - 89/103 - BORDEAUX - 1990

DUR 90
D.DURAND
La systémique
Collection Que sais-je ?
PRESSES UNIVERSITAIRES DE FRANCE - 1990

EDL 89
EDLINGER A.
SPIE-TRINDEL (Uckange)
Vers l'Atelier Logiciel de Génie Automatique : une nécessité pour un Intégrateur
Conférence Automation 89 - 17 Mai 1989

FOR 89
Thomas FORSE (collectif de C.GEORGES, P.BROSSAIS, M.ROMAIN, I.MIALON, F.FICHOT, B.SCARDIA, P.DUCH)
Qualimétrie des systèmes complexes
Mesure de la qualité du logiciel
Les éditions d'Organisation - 1989

FRA 87
J.P.FRACHET
Une introduction au Génie Automatique: faisabilité d'une chaine d'outils CAO pour la conception et l'exploitation des machines automatiques industrielles
Thèse d'Etat - Université de Nancy I - 1987

FRA 90
J.P.FRACHET
Le concept PTA : première mise en oeuvre
Actes des jounées AUTOMATION 90 - 31/48 - PARIS - 1990

GAC 90
R.GACHES, B.QUERENET, P.VIOLET, F.B.VERNADAT
CIM-OSA : une architecture ouverte pour la productique
Actes des journées CIM 90 - BORDEAUX - 1990

GIR 89
E.GIRARD
La notion actuelle d'atelier de Génie Logiciel
Génie Logiciel et Systèmes Experts, N° 14, Mars 1989

GOL 83
A.GOLDBERG
Smalltalk80 : The langage and its implementation
Addison-Wesley Publishing Company, Reading, MA - 1983

GRA 90
H.GRABOWSKI, H.SCHÄFER, S.BRIDGE .
CAD/CAM Integration
Conférence invitée à CIM90 - BORDEAUX - 1990

GRE 85
GREPA
Le grafcet, de nouveaux concepts
CEPADUES EDITIONS - PARIS - 1985

HAL 79
M.L.HALSTEAD
Elements of Software Science
New York
Elsevier 1979

HAT 87
D.J.HATLEY, I.A.PIRBHAI
Stategies for Real-Time System Spécification
Dorset House - 1987

HAU 87
A.HAURAT, J.L.PARRARD
Spécification d'interconnexion d'équipements, analyse préliminaire
Rapport Interne Equipe Logiciel pour la Productique
Institut de Productique / ENSMM - BESABCON - 6 Août 1987

HOD 89
R.HODSON
Methodes for Object-Oriented Software Engineering
Tutorial N°5 - Deuxièmes journées internationales "Le Génie Logiciel et ses Applications"
TOULOUSE - 1989

IGL 89
I.G.L. Technologie
SADT : Un langage pour communiquer
Editions Eyrolles - 1989

ING 87
F.F.INGRAND
Inférences de formes à partir de fonctions , Application à la conception de montage d'usinage
Thèse de 3ème cycle - 6 Février 1987

ISM 83
ISMCM - VALORIS
Modélisation, simulation et évaluation de la partie opérative d'un système automatisé
Rapport scientifique de fin de contrat
Agence de l'Informatique - Convention de Recherche n° 81/545 - Mai 1983

IUN 91
B.IUNG, G.MOREL; P.LHOSTE, M.ROESCH
Functional modelling of an Intelligent Actuator : Aplicable to an ON/OFF or modulating electrical actuator
ESPRIT Workshop "CIM in the Process Industry"
14-15 Mars 1991 - ATHENES (GRECE)

KLO 90
E.KLOTZ
Qualité en Génie Automatique : Approche qualitative à l'image du Génie Logiciel
Rapport bibliographie de DEA Production Automatisée
Université de NANCY I - 1990

KOE 90
J.B.KOECHLIN
Logiciel ELEGA : première réalisation mettant en oeuvre les travaux BASE-PTA
Actes des fournées CIM 90 - BORDEAUX - 1990

LAG 87
Laboratoire d'Automatique de Grenoble - P. LADET
Proposition de Langage Textuel pour la Description des Commandes d'Automatismes
Rapport Final PTA - 1987

LEM 84
J.L.LE MOIGNE
La théorie du système général, théorie de la modélisation
Deuxième édition - PRESSE UNIVERSITAIRE DE FRANCE - 1984

LEP 88
F.LE, C.PEUGEOT
Quelques problèmes liés à l'introduction du concept de généralisation/spécialisation dans le modèle ENTITE-RELATION
Modèles et Bases de Données - N°10 - Juillet 1988

LHO 84
P. LHOSTE
EXAO ou "EXploitation Assistée par Ordinateur des Systèmes Automatisés de Fabrication"
Journées du G.A.M.I. "Les outils de la Productique"
Ecole Centrale de Paris, 5-6 Décembre 1984

LHO 85
P.LHOSTE
Exploitation Assistée par Ordinateur, EX.A.O. : Proposition d'une Approche Méthodologique et d'Outils d'Assistance
Thèse de l'Université de Nancy I Option "Génie Electrique" - 18 Décembre 1985

LHO 88
P.LHOSTE, J.M.TIXADOR, G.MOREL, M.ROESCH
Outils de Conception des Systèmes Automatisés de Production
Congrès AFCET Automatique 88 - 331/339 - GRENOBLE

LLO 87
P.LLORCA
Méthodologie et CAO d'Automatisme à partir du Grafcet : définition, implantation et utilisation de bibliothèques de comportement
Thèse de 3° cycle, MONTPELLIER - 1987

LON 87
J.LONCHAMP
Conception des applications informatiques réparties en commande de procédés industriels : une démarche, des outils
Thèse de Doctorat-ès-Sciences Mathématiques Mention : Informatique
Université de NANCY I - 1987

LON 88
J.LONCHAMP
Méthodes et outils pour la conception d'applications industrielles réparties
Actes du 4ème colloque GENIE LOGICIEL - PARIS - Octobre 1988

MAR 90
H.MARTI, A.SFALCIN
Une aide à la Spécification
Rapport Interne LACN/AIP - 1990

MAS 89
G.MASINI, A.NAPOLI, D.COLNET, D.LEONARD, K.TOMBRE
Les Langages à Objets
InterEdition - 1989

MCC 76
J.A.MACCALL, P.K.RICHARDS, G.F.WALTERS
Factors in Software Quality
Reports NTIS ADIA-049014,015,055 - Vol I, II, II - National Technical Information Services - US
Rome Air Development center
US Departement - 1977

MEV 87
A.MEVEL, T.GUEGEN
Smalltalk-80
Editions Eyrolles - 1987

MEY 88
B.MEYER
Object-oriented Software Construction
Prentice-Hall - 1988

MOR 88
G.MOREL, M.ROESCH, M.VERON
Génie Productique, Génie X
Actes du congrés AFCET Automatique - 320/330 - GRENOBLE - 1988

MOR 89
G.MOREL, M.ROESCH, B.IUNG
Les Outils de XAO dans le cycle de vie de l'automatisation
Revue Générale d'électricité - N° 9 - Octobre 1989

MOR 90
G.MOREL, P.LHOSTE, M.ROESCH
Automatisation intégrée d'un Îlot de Fabrication manufacturière
Actes de CIM90 - 345/354 - BORDEAUX - 1990

MORa 89
J.MOREJON, R.OUDRIHIRI
Vers un processus global de «conception/rétro-conception»
Actes des journées "Le Génie Logiciel & ses Applications" - TOULOUSE - 1989

MUN 88
F.MUNERATO
Robotisation Intégrée d'un Ilot de Production Manufacturière : Aspects Contrôle-Commande et Communication
Thèse de Doctorat de l'Université de Nancy I en Génie Electrique- 1988

NIJ 80
G.M.NIJSSEN
Data base sémantic
INFOTECH - LONDRES - 1980

NIV 89
M.NIVAT
Présentation de C3 : coopération, concurence et communication
1989

PAR 90
E.PARENT
Approche semi-générative de la programmation fonctionnelle d'une cellule flexible de fabrication manufacturière
Raport de DEA en Production Automatisée - Université de Nancy I - 1990

PTA 87
PTA
Poste de travail pour l'Automatisation
Synthèse MRES, RNUR, MICHELIN, ADEPA, PSA, SGN-ISMCM, LACN, LAG, LAMM

PRU 86
F.PRUNET, C.CAZALOT, P.LLORCA, G.DECHENAUX
De l'automatisme simple à l'atelier flexible avec le grafcet
Congrès AFCET automatique - MARS 1986 - MONTPELLIER

QUA
D.QUARANTE, L.MAGNON
Grande encyclopédie ALPHA des Sciences et des Techniques
Volume Technologique V
Editions ATLAS

ROB 88
M.ROBOAM
Modèles de Référence et intégration des méthodes d'analyse pour la conception de Systèmes de Production
Thèse de l'Université de Bordeaux I - 1988

ROSS 75
D.T.ROSS, J.B.DOODENOUGH, C.A.IRVINE
Software Engineering : Process, Principles, and Goals
Computer N° 65 - Mai 1975

SFA 89
A.SFALCIN, M.ROESCH, P.LHOSTE, G.MOREL
Fiabilité, disponibilité et sécurité des installations intégrant des filtres de comportement
2ème colloque annuel du club FIABEX, Institut national des Télécommunications
EVRY - 22-23 Novembre 1989

SFA 90
A.SFALCIN, H.MARTI
Intelligence et Comportement
Rapport Interne - LACN/AIP - 1990

STA 84
T.A.STANDISH
An Essay on Software Reuse
IEEE Transansactions on Software Engineering, Vol SE-10 - N° 5 - 494/497
Septembre 1984

TAB 88
Y.TABOURIER
Du modèle ENTITE-RELATION à un véritable réseau sémantique
Modèles et Bases de Données - N°9 - Juin 1988

THE 85
J.THEVENOT
L'intégration des caractéristiques organisationnelles dans la conception du système d'information: propositions méthodologiques
Thèse de Doctorat d'Etat ès Sciences de Gestion - Université de MONTPELLIER I
30 Novembre 1985

THE 88
P.THERON
Guide pratique du Génie Logiciel
Editions Eyrolles - Paris - 1988

THO 90
J.P.THOMESSE
Les services du bus de terrain FIP et l'intégration dans les systèmes automatisés
Actes des journées CIM 90 - BORDEAUX - 1990

TIX 89
J.M. TIXADOR
Une contribution au Génie Automatique : la SPécification EXécutable des Machines et Systèmes Automatisés de Production
Thèse de l'Université de Nancy I Option "Production Automatisée" - 10 Juillet 1989

VAL 90
J. VALANCOGNE
Présentation des travaux du groupe "méthode de conception des systèmes" de l'A.F.C.I.Q.
Exposé AFCET, groupe Systèmes Logiques - 10 MAi 1990

VAN 75
M.H.VAN EMDEN
An analysis of complexity
Mathematical Center Tract
35 - Mathematisch Centrum Amsterdam - 1975

VOG 87
R.VOGRIG, P.BARACOS, P.LHOSTE, G.MOREL, B.SALZEMANN
Flexible Manufacturing Shop Opération
Proceedings of Manufacturing Systems - Vol. 16 - 43/55 - 1987

VOG 88
C.VOGEL
Génie cognitif
Sciences cognitives - Masson - 1988

WAR 89
P.T.WARD
How to integrate object orientation whith structured analysis and design
IEEE Software - Vol. 6, n° 2, 74/82 - 1989

Techniques

3IP 88
Pré-étude de faisabilité de l'Atelier Logiciel pour le Génie Automatique
Rapport Final 3IP - Décembre 1988

AIP 91
LACN - Equipe Automatisation Intégrée de Production
Rédaction d'un cahier des charges d'automatisation pour la rénovation d'un tour à commande numérique
Rapport Interne LACN/AIP - 1991

ARC 91
ARC INFORMATIQUE
CIMVUE : nouvelle version de PCVUE
Présentation à Automation 91

ASA 88
Atelier de Spécification et de tests de Systèmes Informatiques
Document de Présentation de l'outil ASA, VERILOG - 1988

CCG 87
CCGA : Centre Coopératif de Génie Automatique
carte des activités de développement d'un SAP
Publication du CCGA - ISMCM - PARIS - 1987

CEI 88
CEI 848
Norme Internationale
Etablissement des diagrammes fonctionnels pour systèmes de commande
1ère édition
Commision Electrotechnique Internationnale - 1988

CIA 87
CIAME - AFCET
"Les capteurs intelligents", réflexions des utilisateurs
note de synthèse du livre blanc
CIAME - 1987

CIA 88
CIAME - AFCET
"Les actionneurs intelligents", réflexions des utilisateurs
note de synthèse du livre blanc
CIAME - 1988

CMM 90
CMMS General Concept
Workpakage 0
Projet ESPRIT/DIAS 2172 - 1990

COR 89
CORBIER F.
Modélisation et Emulation de la Partie Opérative pour la recette en plateforme d'équipements automatisés
Thèse de l'Université de Nancy I Option MAE - 30 Juin 1989

DIAS 89
Projet ESPRIT/DIAS 2172
Distributed Intelligent Actuators and Sensors
1989 - 1992

EDF 90
B.IUNG, D.GALARA, G.MOREL
Spécification d'un actionneur intelligent
Groupe EDF Actionneurs
Protocole de Collaboration CRAN/LACN - EDF N° UNI 1990

EME 90
GIE EMERAUDE
Documentation Emeraude V 12.1.2
Paris - 1990

EXC 89
EXCELERATOR
Reference Guide - Application Guide - Release 1.9
CAO SIGETI - 1989

FOR 90
FORESIGHT
Prototypage systèmes avec ESML+ sur FORESIGHT
Documentation Microtools - Courbevoie - 1990

GEM 89
GEMSTONE
Object-Oriented Database Management System
Documentation Servio Logic Development Corporation
Alameda (California) - 1989

ICA 81
ICAM
Integated Computer-Aided Manufacturing
IDEF0: function modeling manual
IDEF1: information modeling manual
IDEF2: dynamics modeling manual
Air Force Aeronautical Laboratories - Wright-Patterson Air Force Base
Ohio 45433 - June 1981

ISD 90
Integrated Software Development Methodology Software Tools
DESIGN-IDEF : Atelier de spécification de SADT/IDEF0 et E/A
DESIGN/CPN : Simulation et validation de spécifications fonctionnelles
V.O.PINCI, R.M.SHAPIRO : *Development and implementation of strategy for electronic funds transfer by means of hierarchical colored Petri nets*
R.M.SHAPIRO : *IDEF/CPN : an extension to SADT to support Behavioral Modeling*
Documents Meta Software Corporation et IGL Technology - PARIS - 1990

LHO 88a
P. LHOSTE, F. MUNERATO
Application d'une méthode de structuration des parties commndes d'automatismes
Rapport de fin de contrat RNUR/LACN
Université de NANCY I N° UNI 88.256, 1988

LHO 89
P.LHOSTE
Automatisation d'un montage embarqué
rapport Interne LACN/AIP - 1989

MAX 91
MAXSIM
Emulateur de Partie Opérative
SPIE AUTOMATION - 1991

MEG 89
MERISE-MEGA
MEGA - Manuel de référence - 1989
ESPACE-MICRO - Manuel utilisateur - 1989
MEGA-REALISATION - Manuel de référence - 1989
GAMMA International - PARIS

MIP 87
MIPTT-SERICS
Projet PTA
contrat n° 87.2.4.191 - 25 Décembre 1987
Convention n° UNI 88.113 entre SPIE TRINDEL, TNI et le CRAN/LACN

NEX 88
NEXPERT OBJECT
Documentation NEURON DATA
NEURON DATA INC FRANCE - PARIS

NFC 82
NFC 03-190
Norme Française AFNOR
Diagramme fonctionnel "GRAFCET" pour la description des systèmes logiques de commande
1982

ORA 90
ORACLE
Manuels utilisateurs ORACLE V 6.
ORACLE FRANCE
PARIS - 1990

PAC 86
PACT Consortium
PACT general description
Programme ESPRIT, Septembre 1986

PAN 90
H. PANETTO
Automatisation d'une Machine Transfert
Rapport Interne LACN/AIP - 1990

PCI 90
PRECISE * PC-IAST
Documentation PC-IAST
CONTROL DATA FRANCE - PARIS - 1990

PCT 88
PCTE Consortium
Functionnal specification Vol 1 et 2
Version 1.5 - Novembre 1988

PRO 88
PROSYST
Simulateur de machine commandée par automate programmable
Documentation société PROSYST - VALENCIENNES - 1988

PRO 90
PROPEL
Manuel utilisateur
ITMI - GRENOBLE - 1990

SGN 90
CADEPA V. 5
Manuel Utilisateur
SGN Lorraine - 1990

SQL 89
SQL SERVER
Manuels utilisateur
Microsoft et Ashton-Tate - 1989

STP 88
STP (Software Through Pictures)
Reference Guide Tomes 1, 2, 3 - Release 3.0
IGL Technologie - 1988

TNI 89
Documentation de Présentation de l'Outil ORCHIS
TNI - 1989

TSA 89
TSA PREMOTECH
Collection A.CAPLIEZ
EDUCALIVRE
par J.P.CHASSAING, J.C.LABARDIN, P.MERIGAUD

Liste des Figures

Introduction - Problématique

Chapitre I : SPEX : un outil, deux métiers

Chapitre II : Sémantique des objets d'automatisation

Chapitre III : De l'interconnexion d'outils ...

Chapitre IV : ... Vers la spécification d'un Atelier de génie Automatique

Annexes
Conception de la Partie Commande d'une Machine Transfert

Annexe A : Spécification de Conception
Annexe B : Grilles de Description des Missions de Contrôle-Commande
Annexe C : Application SPEX
Annexe D : Simulation de l'Application

Exemple d'application de l'interconnexion ORCHIS-GRILLE-SPEX

Conception de la Partie Commande d'une Machine Transfert

Nous présentons dans ces annexes un extrait du dossier de spécification et de conception de la Partie Commande d'une Machine Transfert, de son Robot de chargement/déchargement et de son Convoyeur de palettes.

Le cadre de l'étude

L'installation est composée (Figure 1) :

- d'un convoyeur possédant deux tapis de transfert de palettes, l'un déservant une machine en amont de la machine transfert, l'autre une machine en aval. Nous avons choisi un point de vue de conception supposant une gestion des palettes en "flux tiré", en ce sens qu'une machine est prioritaire à une machine plus en amont en ce qui concerne les demandes de palettes. Ce concoyeur est piloté par un Automate Programmable Insdustriel (API) ;

- d'une machine transfert comprenant quatre postes (Chargement/Déchargement, Perçage, Fraisage, Taraudage) ainsi qu'un système de bridage commun et un plateau tournant pour le transfert des pièces entre postes. Elle est commandée par un API, en liaison avec l'API du convoyeur par un réseau JBUS, ce dernier étant esclave de la machine ;

- d'un Robot 6 axes AID piloté par une commande numérique. L'armoire de commande numérique possède des entrées/sorties TOR connectées à l'automate de la machine transfert pour la synchronisation avec cette dernière.

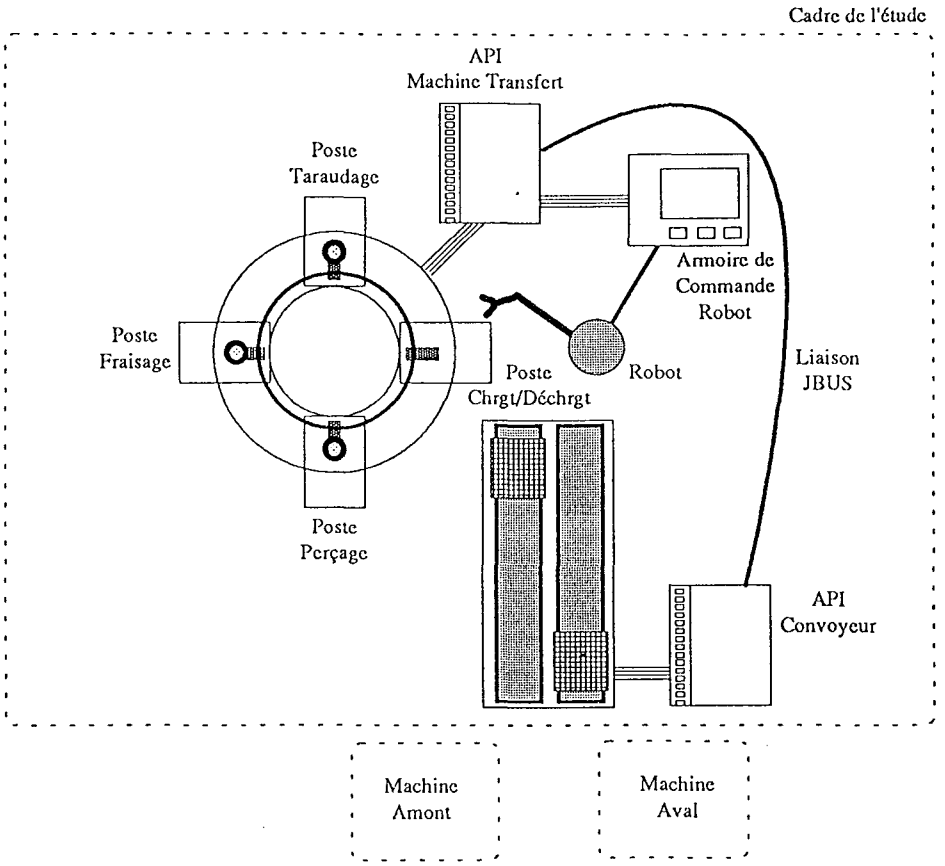


Figure 1. : L'installation étudiée

Technologie d'actionnement

Nous ne présenterons pas la technologie d'actionnement du convoyeur et du robot, ces annexes se limitant principalement à la description de Partie Commande de la Machine Transfert.

Machine Transfert

Les postes de Perçage et Fraisage sont chacun constitués d'un moteur triphasé à un sens et une vitesse et d'un vérin pneumatique double effet muni de deux

contacts de fin de course et préactionné par un distributeur bistable 5/2 à commande électropneumatique.

Le poste de Taraudage est mis en oeuvre par un moteur triphasé à deux sens et une vitesse mettant en rotation une vis sans fin. La vis est munie de deux contacts de fin de course et d'un limiteur de pression.

Tous ces moteurs renvoient une information de retour contacteur.

Le bridage des pièces est actionné par la sortie d'un vérin pneumatique simple effet en position repos rentrée muni d'un pressostat et préactionné par un distributeur bistable 5/2 à commande électropneumatique.

Le plateau tournant est actionné par un vérin pneumatique double effet associé à une crémaillère et des crabots, et préactionné par un distributeur monostable 5/2 à commande électropneumatique. Le plateau est équipé de deux capteurs indiquant, pour l'un le retour arrière de la crémaillère, pour l'autre une rotation de 360° à partir d'une position d'origine.

Le conditionnement en air comprimée est assuré par un distributeur monostable 3/2 à commande électropneumatique muni d'un manocontact indiquant la présence de la pression dans la circuiterie.

Démarche méthodologique

L'**annexe A** montre un extrait de la spécification de conception, éditée sur l'outil Orchis, concernant les niveaux :

- A-1 qui montre la place de la Machine Transfert au sein d'un ilôt de production,
- A-0 qui représente ses échanges d'informations et de matière avec l'environnement,
- A0 qui décrit le premier niveau de décomposition faisant apparaître la machine proprement dite et le convoyeur.
- A2 qui décompose le processus associé au convoyage en deux sous-processus gérant chacun un tapis.
- A3 qui décrit le premier niveau de décomposition de la Machine Transfert.
- A34 qui fait apparaître les différents postes d'usinage.
- A343 qui indique les relations entre les processus de rotation et de translation de la broche perçage.

- A3432 et A3433 qui représentent les mécanigrammes associées respectivement au moteur de rotation et au vérin de translation de la broche perçage.

L'**annexe B** présente les différentes grilles de définition des fonctions de contrôle-commande précédemment identifiées ainsi que l'extrait du glossaire et du dictionnaire de données correspondant, tels que générés par l'outil GRILLE.

L'**annexe C** est composée d'une partie des exemplaires de Boîtes Fonctionnelles et/ou Diagrammes Fonctionnels utilisés pour cette application et montre le détail de quelques Boîtes Fonctionnelles Génériques.

Enfin, l'**annexe D** décrit quelques "vues" représentatives de la simulation de l'application ainsi qu'un scénario de test et un extrait de la trace textuelle associée.

Cahier des Charges

La Machine Transfert peut fonctionner selon un mode automatique ou un mode manuel par sélection respective des commutateurs CAUTO ou CMANU. Deux arrêts d'urgence sont prévus, l'un dans l'armoire de commande (AUARM), l'autre sur le pupitre de conduite (AUPUP). La mise sous tension électrique de l'installation est effectuée par l'appui du bouton poussoir BST, l'alimentation pneumatique est assurée après appui sur BSP. L'arrêt de la tension électrique nécessite un appui sur le bouton poussoir BAT.

Mode Automatique (Mode signalé par LAUTO)

Le cycle d'usinage d'une pièce suit la séquence : Chargement, Perçage, Fraisage, Taraudage, Déchargement.

Les palettes sont constituées de seize pièces identiques. Nous nous sommes limités à des palettes de quatres pièces pour des raisons de simplification.

Le cycle automatique peut être de deux types exclusifs :

- un "**cycle une pièce**" sélectionnable par le commutateur C1PIC et signalé par L1PIC, effectuant un usinage pièce par pièce, le robot ne chargeant une nouvelle pièce que lorsque la pièce précédente a été complètement usinée.

- un "**cycle quatre pièces**" sélectionnable par le commutateur C4PIC et pour lequel les pièces sont usinées dès leur arrivée à chaque poste d'usinage.

Mode Manuel

Le mode manuel permet la commande de tous les postes d'usinage à partir du pupitre de conduite par sélection des boutons poussoirs correspondants. Par exemple, l'actionnement du bouton poussoir RMPS (Réglage Mouvement Perçage Sortie) met en route la rotation de la broche perçage et effectue la sortie du vérin de translation correspondant ; l'actionnement de RMPR (Réglage Mouvement Perçage Rentrée) rentre le vérin de perçage puis arrête le moteur.

Surveillance

Dans tous les modes, les éléments opératifs sont continuellement surveillés. Tout défaut constaté provoque la désactivation de la séquence en cours. L'appui sur le BP "ACQDF" provoque la réinitialisation de la séquence.

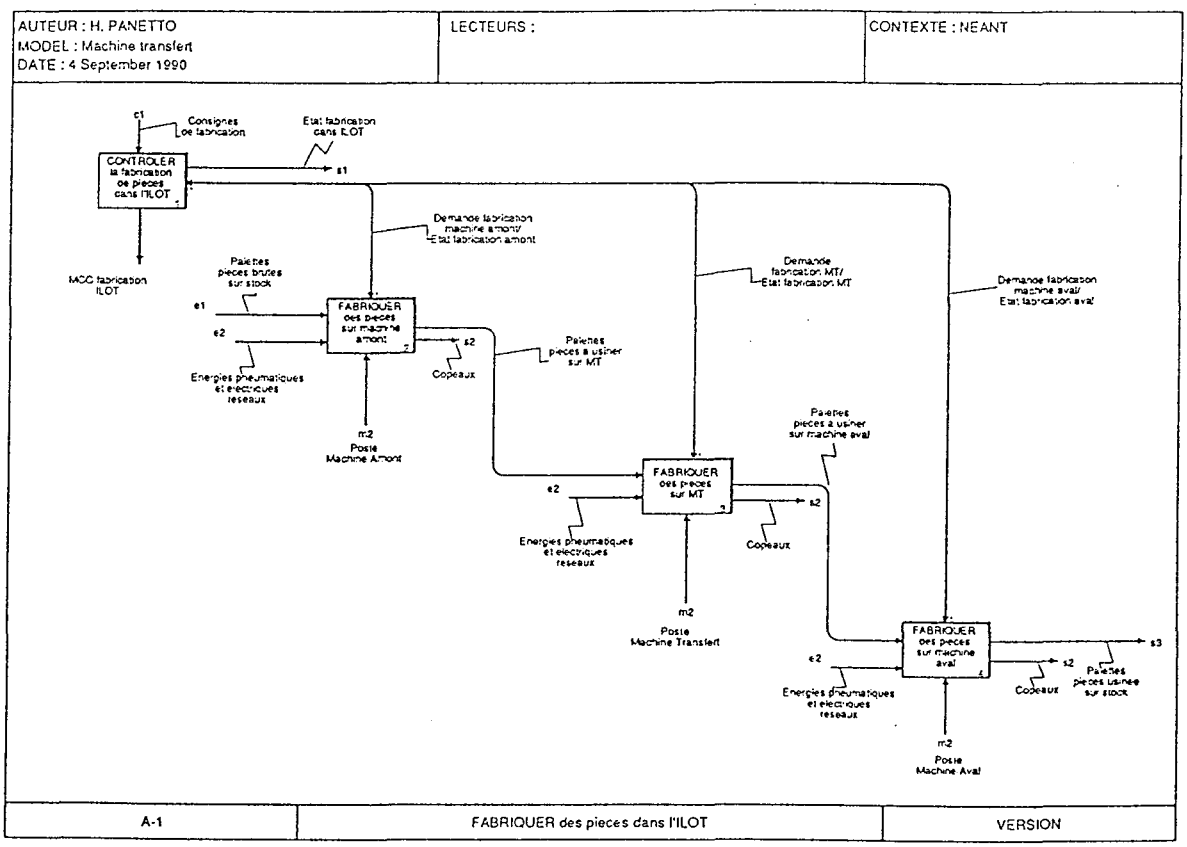
Le défaut est signalé par "LDF"

Arrêt Urgence

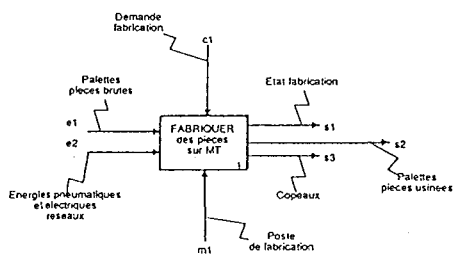
L'appui sur AUPUP ou AUARM provoque la même réaction que pour la détection d'un défaut.

Annexe A
Spécification de Conception

La démarche préconisée consiste, comme présentée au Chapitre III, §I.1.1.1.2.2.3, en une décomposition de chaque processus identifié par une fonction de Contrôle-Commande de coordination et par l'ensemble des sous-processus coordonnés. Cette décomposition est réitérée à chaque niveau jusqu'à la description d'un mécanigramme, représentatif de la technologie d'actionnement du processus considéré.

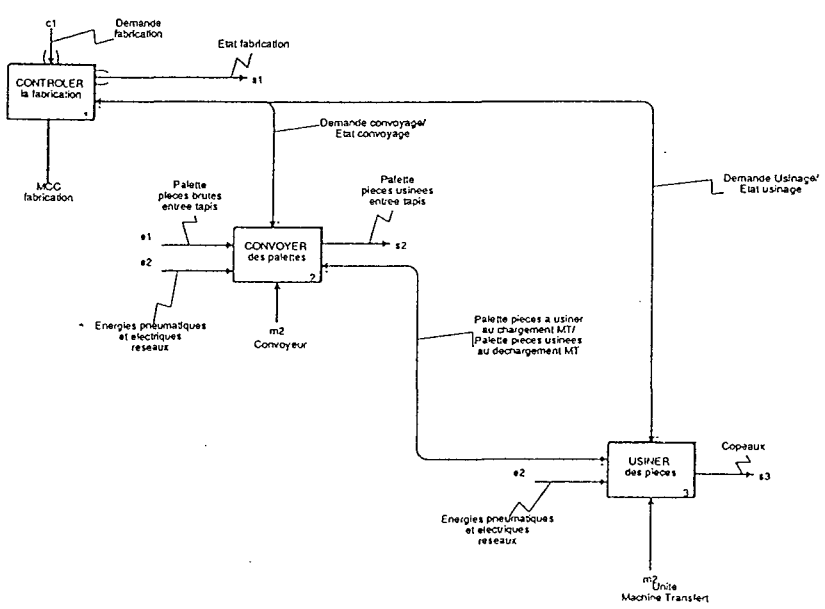


AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : FABRIQUER des pieces sur ILOT



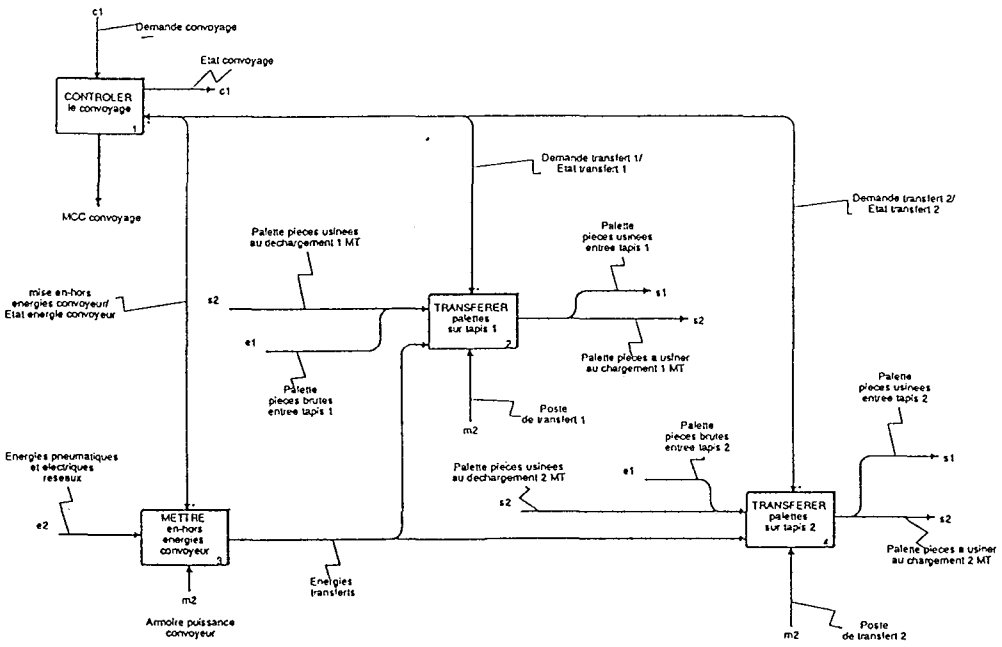
A-0	FABRIQUER des pieces sur MT	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : FABRIQUER des pieces sur MT



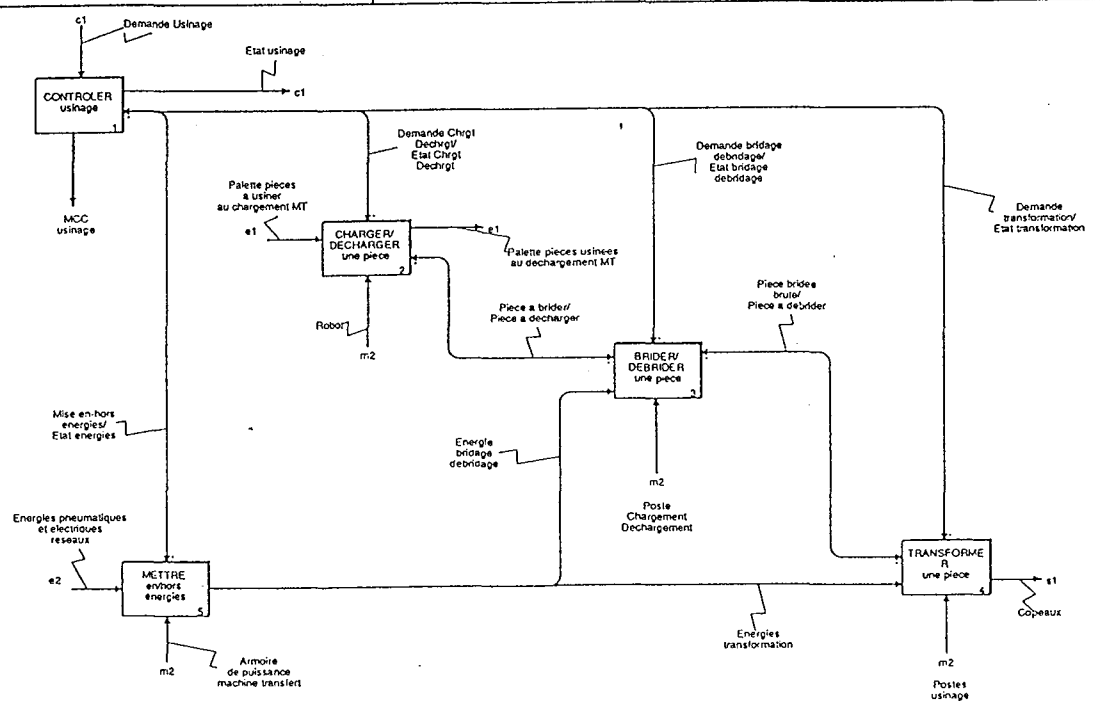
A0	FABRIQUER des pieces sur MT	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : FABRIQUER des pieces



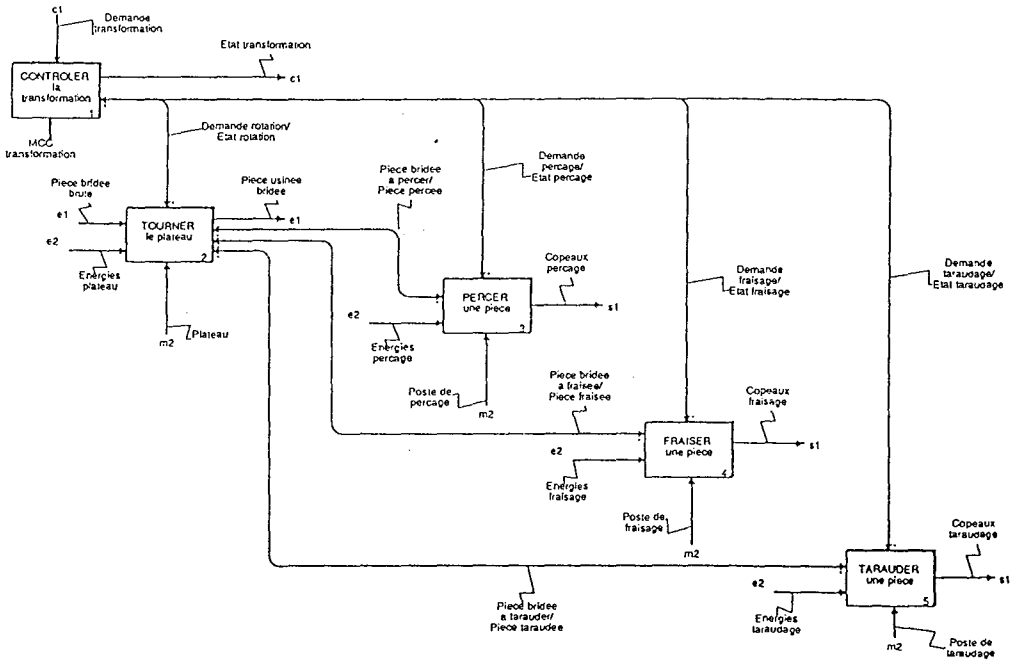
A2	CONVOYER des palettes	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : FABRIQUER des pieces



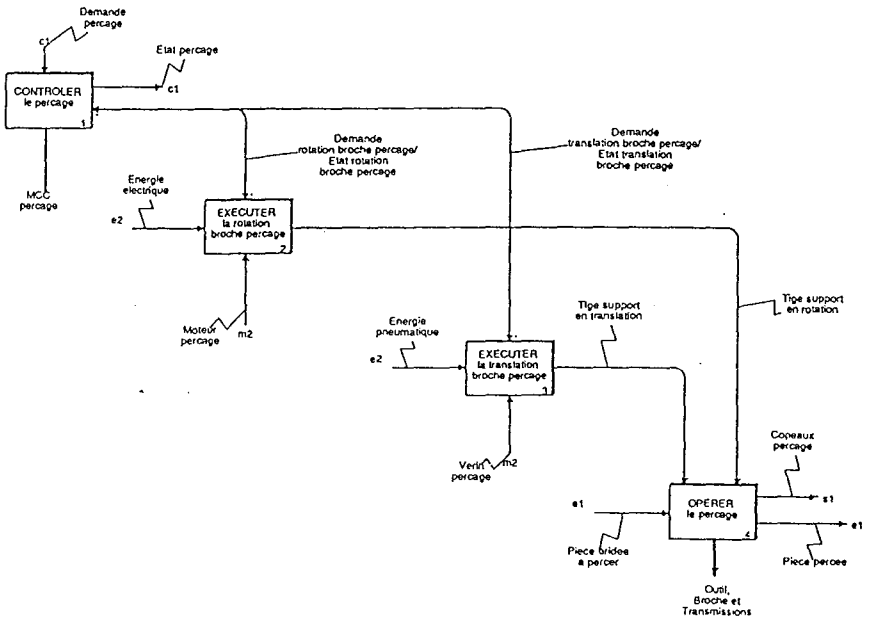
A3	USINER des pieces	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : USINER des pieces



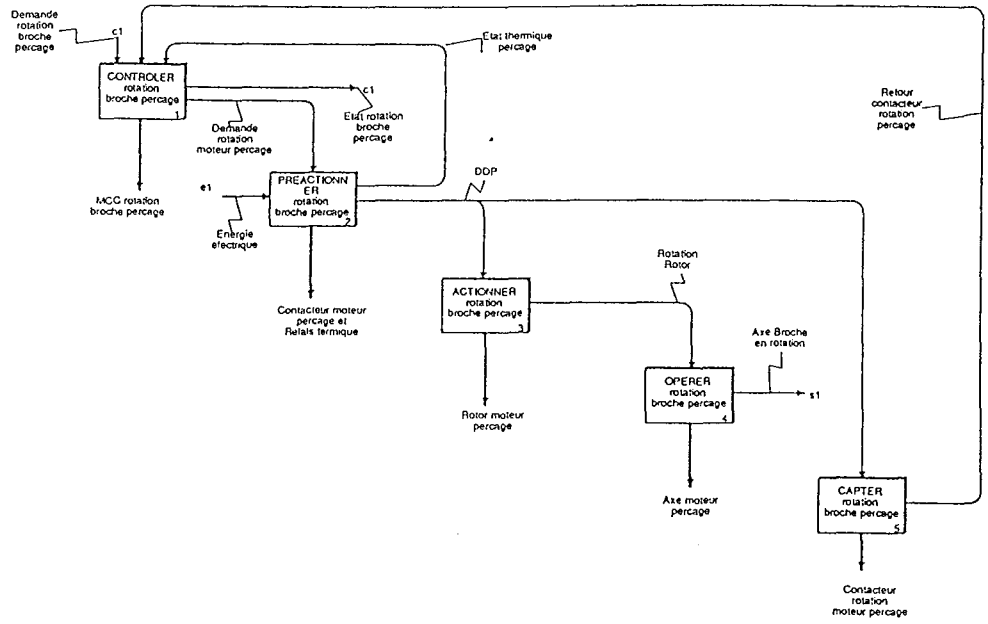
A34	TRANSFORMER une piece	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : TRANSFORMER une piece



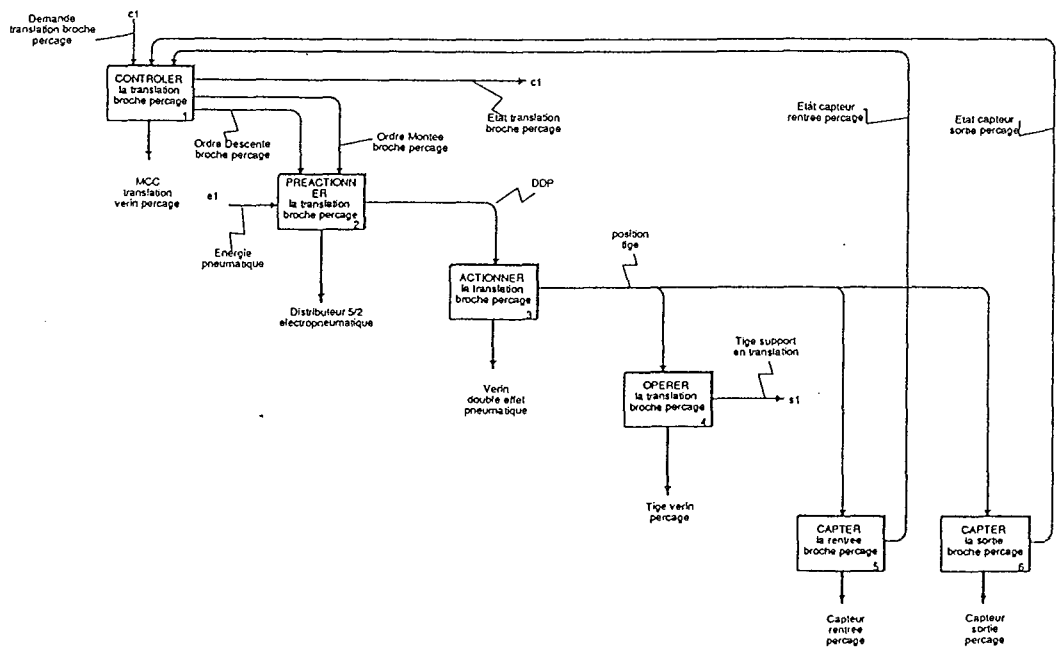
A343	PERCER une piece	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : PERCER une piece

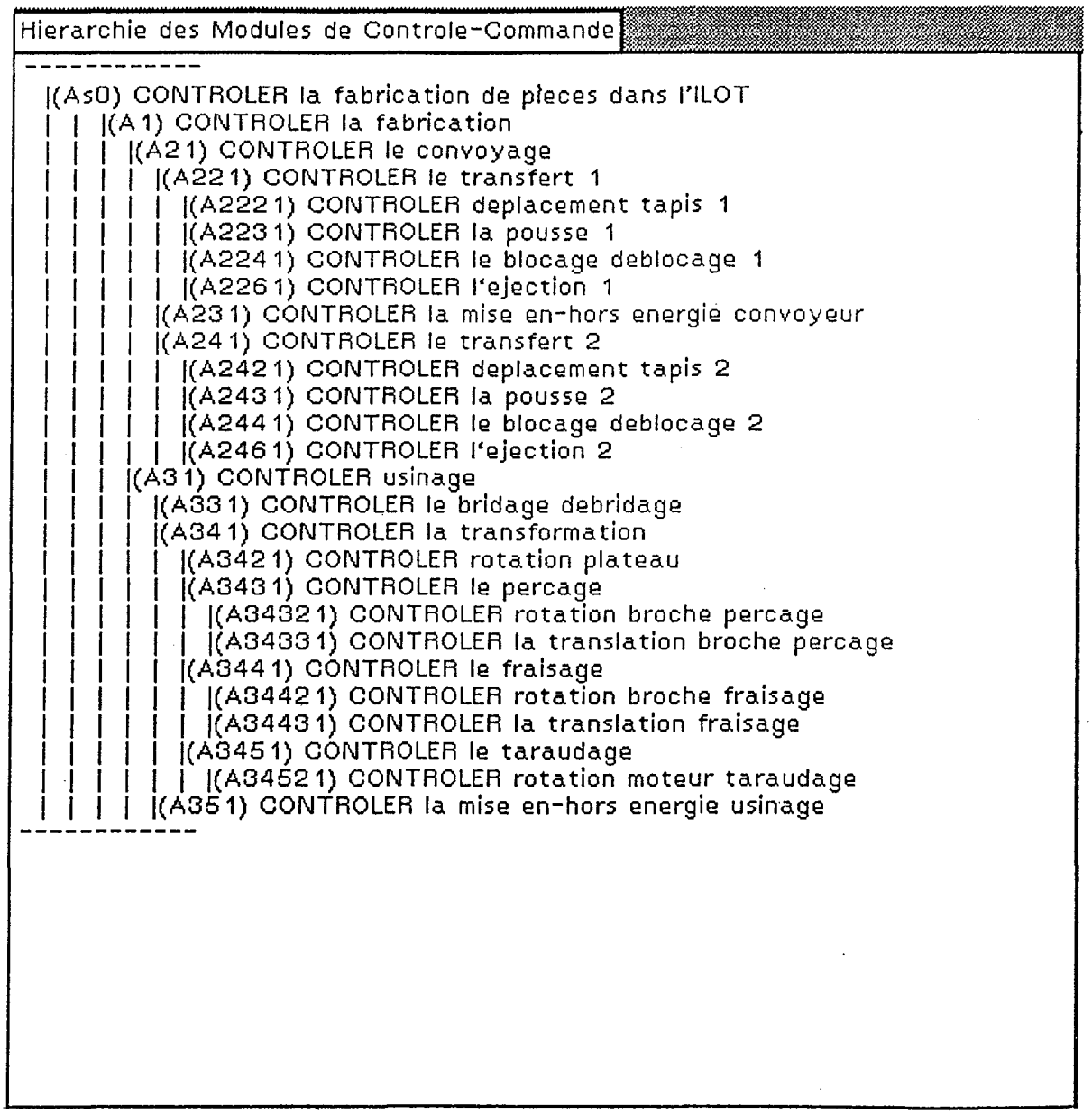


A3432	EXECUTER la rotation broche percage	VERSION

AUTEUR : H. PANETTO MODEL : Machine transfert DATE : 4 September 1990	LECTEURS :	CONTEXTE : PERCER une piece



A3433	EXECUTER la translation broche percage	VERSION



Liste Hiérarchique des Fonctions de Contrôle-Commande

Annexe B
Grilles de Description des Missions de Contrôle-Commande

Nous nous limiterons, dans cette annexe, à la description des fonctions de Contrôle-Commande correspondant à la Machine Transfert c'est à dire :

"CONTROLER usinage"

"CONTROLER la transformation"

"CONTROLER le perçage"

"CONTROLER rotation broche perçage"

"CONTROLER la translation broche perçage"

A chacune de ces fonctions correspond un Module de Contrôle-Comande (MCC) qui est détaillé à l'aide de l'outil GRILLE (cf Chapitre III §I 1.1.1.2.2.4).

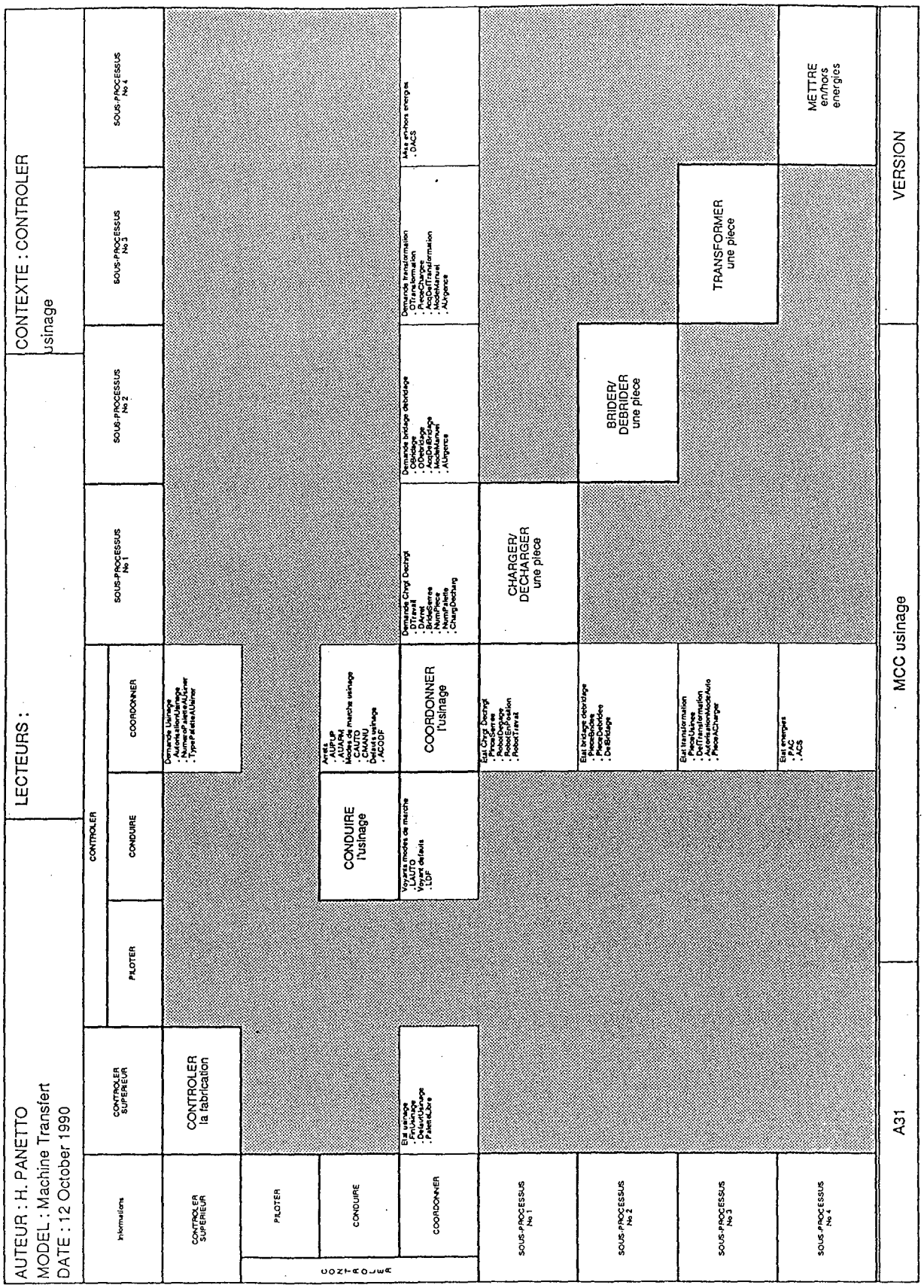
Chaque grille importe la fonction de niveau supérieur, la fonction concernée et l'ensemble des sous-processus qu'elle coordonne. L'ensemble des flux d'information est aussi importé.

L'outil propose de sélectionner les sous-fonctions de contrôle désirées parmi "PILOTER", "CONDUIRE", "COORDONNER".

Chaque "label" importé de l'analyse doit être détaillé par les données élémentaires mises en jeu. L'ensemble des relations informationnelles entre les sous-fonctions doit aussi être explicité.

Nous remarquons que les boutons poussoirs d'actionnement manuel du poste de perçage apparaissent au niveau du "MCC perçage" car il est nécessaire de commander automatiquement la marche et l'arrêt du moteur perçage, ce dernier n'ayant pas de commande manuelle.

Les MCC correspondant aux mécanigrammes montrent que la commande n'est effectuée qu'au niveau du préactionnement avec un retour d'information par le "retour contacteur" ou les capteurs.


AUTEUR : H. PANETTO
MODEL : Machine Transfert
DATE : 12 October 1990
LECTEURS :
CONTEXTE : CONTROLER
usinage
CONTROLER
PILOTER
CONDUIRE
COORDONNER
SOUS-PROCESSUS No 1
SOUS-PROCESSUS No 2
SOUS-PROCESSUS No 3
SOUS-PROCESSUS No 4
Informations
CONTROLER SUPERIEUR
CONTROLER la fabrication
CONDUIRE l'usinage
COORDONNER l'usinage
CHARGER/ DECHARGER une piece
BRIDER/ DEBRIDER une piece
TRANSFORMER une piece
METTRE en/hors energies
A31
MCC usinage
VERSION

AUTEUR : H. PANETTO
MODEL : Machine Transfert
DATE : 12 October 1990

LECTEURS :

CONTEXTE : CONTROLER
a transformation

Informations	CONTROLER SUPERIEUR	CONTROLER			SOUS-PROCESSUS No 1	SOUS-PROCESSUS No 2	SOUS-PROCESSUS No 3	SOUS-PROCESSUS No 4
		PILOTER	CONDUIRE	COORDONNER				
CONTROLER SUPERIEUR	CONTROLER usinage			Demande transformation . OTransformation . PieceChargee . AcqDefTransformation . ModeManuel . AUrgence				
CONTROLER — PILOTER								
CONTROLER — CONDUIRE			CONDUIRE la transformation	Modes cycle . C1PIC . C4PIC				
CONTROLER — COORDONNER	Etat transformation . PieceUsinee . DefTransformation . AutorisationModeAuto . PieceACharger		Voyants cycles . L1PIC	COORDONNER la transformation	Demande rotation . ORotationPlateau . AcqDefUnite . ModeReglage	Demande percage . OPercage . AcqDefUnite . AUUnite . ModeReglage	Demande fraisage . OFraisage . AcqDefUnite . AUUnite . ModeReglage	Demande taraudage . OTaraudage . AcqDefUnite . AUUnite . ModeReglage
SOUS-PROCESSUS No 1				Etat rotation . RotationQuartTour . RotationUnTour . DefRotation . PlateauPret	TOURNER le plateau			
SOUS-PROCESSUS No 2				Etat percage . FinPercage . DefPercage . PercagePret		PERCER une piece		
SOUS-PROCESSUS No 3				Etat fraisage . FinFraisage . DefFraisage . FraisagePret			FRAISER une piece	
SOUS-PROCESSUS No 4				Etat taraudage . FinTaraudage . DefTaraudage . TaraudagePret				TARAUDER une piece

A341	MCC transformation	VERSION

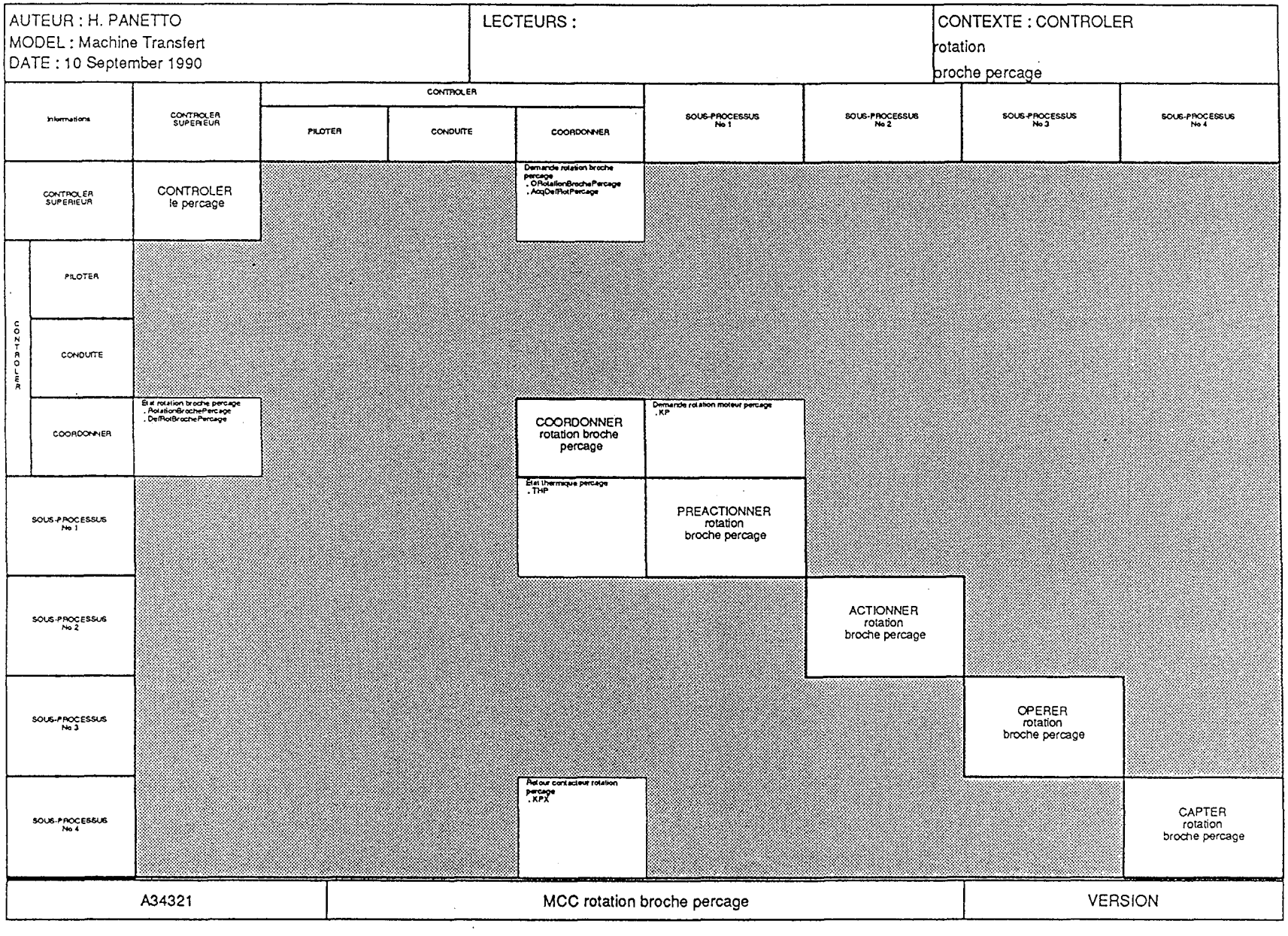
AUTEUR : H. PANETTO
MODEL : Machine Transfert
DATE : 21 September 1990

LECTEURS :

CONTEXTE : CONTROLER
le percage

Informations	CONTROLER SUPERIEUR	CONTROLER: PILOTER	CONTROLER: CONDUITE	CONTROLER: COORDONNER	SOUS-PROCESSUS No 1	SOUS-PROCESSUS No 2	SOUS-PROCESSUS No 3
CONTROLER SUPERIEUR	CONTROLER la transformation			Demande percage . OPercage . AcqDefUnite . AUUnite . ModeReglage			
CONTROLER: PILOTER							
CONTROLER: CONDUITE			CONDUIRE le percage	BPs Mode reglage percage . RMPS . RMPR			
CONTROLER: COORDONNER	Etat percage . FinPercage . DefPercage . PercagePret			COORDONNER le percage	Demande rotation broche percage . OPRotationBrochePercage . AcqDefRotPercage	Demande translation broche percage . ODescenteBrochePercage . OMonteeBrochePercage . AcqDefTransPercage	
SOUS-PROCESSUS No 1				Etat rotation broche percage . RotationBrochePercage . DefRotBrochePercage	EXECUTER la rotation broche percage		
SOUS-PROCESSUS No 2				Etat translation broche percage . BrocheHautPercage . BrocheBasPercage . DefTransBrochePercage		EXECUTER la translation broche percage	
SOUS-PROCESSUS No 3							OPERER le percage

A3431	MCC percage	VERSION


AUTEUR : H. PANETTO
MODEL : Machine Transfert
DATE : 10 September 1990
LECTEURS :
CONTEXTE : CONTROLER
rotation
broche percage
Informations
CONTROLER SUPERIEUR
CONTROLER
PILOTER
CONDUITE
COORDONNER
SOUS-PROCESSUS No 1
SOUS-PROCESSUS No 2
SOUS-PROCESSUS No 3
SOUS-PROCESSUS No 4
CONTROLER le percage
Demande rotation broche percage
. ORotationBrochePercage
. AcqDefRotPercage
Etat rotation broche percage
. RotationBrochePercage
. DefRotBrochePercage
COORDONNER rotation broche percage
Demande rotation moteur percage
. KP
Etat thermique percage
. THP
PREACTIONNER rotation broche percage
ACTIONNER rotation broche percage
OPERER rotation broche percage
Retour contacteur rotation percage
. KPX
CAPTER rotation broche percage
A34321
MCC rotation broche percage
VERSION

AUTEUR : H..PANETTO
MODEL : Machine Transfert
DATE : 10 September 1990

LECTEURS :

CONTEXTE : CONTROLER
la translation
broche percage

Informations	CONTROLER SUPERIEUR	CONTROLER: PILOTER	CONTROLER: CONDUITE	CONTROLER: COORDONNER	SOUS-PROCESSUS No 1	SOUS-PROCESSUS No 2	SOUS-PROCESSUS No 3	SOUS-PROCESSUS No 4	SOUS-PROCESSUS No 5
CONTROLER SUPERIEUR	CONTROLER le percage			Demande translation broche percage . ODescenteBrochePercage . OMonteeBrochePercage . AcqDefTransPercage					
CONTROLER: PILOTER									
CONTROLER: CONDUITE									
CONTROLER: COORDONNER	Etat translation broche percage . BrocheHautPercage . BrocheBasPercage . DefTransBrochePercage			COORDONNER translation broche percage	Ordre Montee broche percage . EVPS Ordre Descente broche percage . EVPR				
SOUS-PROCESSUS No 1					PREACTIONNER la translation broche percage				
SOUS-PROCESSUS No 2						ACTIONNER la translation broche percage			
SOUS-PROCESSUS No 3							OPERER la translation broche percage		
SOUS-PROCESSUS No 4				Etat capteur rentree percage . FCPR				CAPTER la rentree broche percage	
SOUS-PROCESSUS No 5				Etat capteur sortie percage . FCPS					CAPTER la sortie broche percage

A34331	MCC translation verin percage	VERSION

AUTEUR : H. PANETTO MODEL : Machine Transfert DATE : 1 August 1990	LECTEURS :	

DICTIONNAIRE DE DONNES

Etat thermique percage
. THP

Retour contacteur rotation percage
. KPX

Voyant defauts
. LDF

Ordre Montee broche percage
. EVPS

BPs Mode reglage percage
. RMPS
. RMPR

Ordre Descente broche percage
. EVPR

Voyants modes de marche
. LAUTO

Etat transformation
. PieceUsinee
. DefTransformation
. AutorisationModeAuto
. PieceACharger

Defauts usinage
. ACQDF

Demande transformation
. OTransformation
. PieceChargee
. AcqDefTransformation
. ModeManuel
. AUrgence

Etat usinage
. FinUsinage
. DefautUsinage
. PaletteLibre

Demande Usinage
. AutorisationUsinage
. NumeroPaletteAUsiner
. TypePaletteAUsiner

Etat percage
. FinPercage
. DefPercage
. PercagePret

Modes de marche usinage
. CAUTO
. CMANU

Demande percage
. OPercage
. AcqDefUnite
. AUUnite
. ModeReglage

Demande rotation moteur percage
. KP

Arrets
. AUPUP
. AUARM

Etat fraisage
. FinFraisage
. DefFraisage
. FraisagePret

Etat taraudage
. FinTaraudage
. DefTaraudage
. TaraudagePret

Demande fraisage
. OFraisage
. AcqDefUnite
. AUUnite
. ModeReglage

Demande taraudage
. OTaraudage
. AcqDefUnite
. AUUnite
. ModeReglage

Etat translation broche percage
. BrocheHautPercage
. BrocheBasPercage
. DefTransBrochePercage

Demande translation broche percage
. ODescenteBrochePercage
. OMonteeBrochePercage
. AcqDefTransPercage

Etat energies
. PAC
. ACS

Mise en-hors energies
. DACS

Etat rotation broche percage
. RotationBrochePercage
. DefRotBrochePercage

Demande rotation broche percage
. ORotationBrochePercage
. AcqDefRotPercage

Etat Chrgt Dechrgt
. PinceSerree
. RobotDegage
. RobotEnPosition
. RobotTravail

Demande Chrgt Dechrgt
. DTravail
. DArret
. BrideSerree
. NumPiece
. NumPalette
. ChargDecharg

Voyants cycles
. L1PIC

Etat bridage debridage
. PieceBridee
. PieceDebridee
. DefBridage

Etat rotation
. RotationQuartTour
. RotationUnTour
. DefRotation
. PlateauPret

Demande bridage debridage
. OBridage
. ODebridage
. AcqDefBridage
. ModeManuel
. AUrgence

Modes cycle
. C1PIC
. C4PIC

Demande rotation
. ORotationPlateau
. AcqDefUnite
. ModeReglage

Etat capteur rentree percage
. FCPR

Etat capteur sortie percage
. FCPS

		VERSION

AUTEUR : H.PANETTO MODEL : Machine Transfert DATE : 1 August 1990	LECTEURS :	CONTEXTE : CONTROLER usinage

GLOSSAIRE (MCC usinage)

. AutorisationUsinage
Demande usinage pieces a la MT

. NumeroPaletteAUsiner
Numero de la palette a usiner

. TypePaletteAUsiner
Type de la palette a usiner

Etat usinage
Type de la palette a usiner

. FinUsinage
Fin usinage de la palette, on peut la reprendre
mais elle n'est peut etre pas entierement chargee

. DefautUsinage
Defaut usinage MT

. PaletteLibre
Palette usinee et chargee

Demande Chrgt Dechrgt
Palette usinee et chargee

. DTravail
Demande travail Robot

. DArret
Demande arret Robot

. DefTransformation
Defaut transformation

. AutorisationModeAuto
Autorisation mode automatique

. PieceACharger
Demande chargement piece

. PAC
Pression air comprimee

. ACS
Sorties alimentees

. BrideSerree
Information bride serree

. NumPiece
Numero piece a charger

. NumPalette
Numero palette en cours de chargement

. ChargDecharg
Information cycle chargement ou dechargement

. OBridage
Ordre bridage piece

. ODebridage
Ordre debridage piece

. AcqDefBridage
Acquittement defaut bridage/debridage

. ModeManuel
Mode manuel

. AUrgence
Arret d'urgence

. OTransformation
Ordre cycle transformation

. PieceChargee
CR Piece chargee

. AcqDefTransformation
Acquittement defaut transformation

. ModeManuel
Mode manuel

. AUrgence
Arret d'urgence

. DACS
Relais desactivation des sorties

Etat Chrgt Dechrgt
Relais desactivation des sorties

. PinceSerree
Information pince serree

. RobotDegage
Robot en position replis

. RobotEnPosition
Robot en position chargement ou dechargement

. RobotTravail
Acquittement demande travail

. PieceBridee
CR Piece bridee

. PieceDebridee
CR piece debridee

. DefBridage
Defaut bridage/debridage

. PieceUsinee
Piece usinee au poste dechargement

A31	MCC usinage	VERSION

AUTEUR : H.PANETTO MODEL : Machine Transfert DATE : 1 August 1990	LECTEURS :	CONTEXTE : CONTROLER la transformation

GLOSSAIRE (MCC transformation)

. **OTransformation**
Ordre cycle transformation

. **PieceChargee**
CR Piece chargee

. **AcqDefTransformation**
Acquittement defaut transformation

. **ModeManuel**
Mode manuel

. **AUrgence**
Arret d'urgence

. **PieceUsinee**
Piece usinee au poste dechargement

. **DefTransformation**
Defaut transformation

. **AutorisationModeAuto**
Autorisation mode automatique

. **PieceACharger**
Demande chargement piece

. **ORotationPlateau**
Ordre rotation plateau 1 increment

. **AcqDefUnite**
Acquittement defaut

. **ModeReglage**
Mode manuel

. **OPercage**
Ordre cycle percage piece

. **AcqDefUnite**
Acquittement defaut

. **AUUnite**
Arret d'urgence

. **ModeReglage**
Mode manuel

. **OFraisage**
Ordre cycle fraisage piece

. **AcqDefUnite**
Acquittement defaut

. **AUUnite**
Arret d'urgence

. **ModeReglage**
Mode manuel

. **OTaraudage**
Ordre cycle taraudage piece

. **AcqDefUnite**
Acquittement defaut

. **AUUnite**
Arret d'urgence

. **RotationQuartTour**
CR rotation 1/4 tour

. **RotationUnTour**
Cr rotation 1/2 tour

. **DefRotation**
Defaut rotation plateau

. **PlateauPret**
Plateau pret

. **FinPercage**
Fin cycle percage

. **DefPercage**
Defaut percage

. **PercagePret**
Percage pret

. **FinFraisage**
Fin cycle fraisage

. **DefFraisage**
Defaut fraisage

. **FraisagePret**
Fraisage pret

. **FinTaraudage**
Fin cycle taraudage

. **DefTaraudage**
Defaut taraudage

. **TaraudagePret**
Taraudage pret

A341	MCC transformation	VERSION

AUTEUR : H.PANETTO MODEL : Machine Transfert DATE : 1 August 1990	LECTEURS :	

GLOSSAIRE (MCC translation verin percage)

. EVPS
Electrovanne verin percage sortie

. EVPR
Electrovanne verin percage rentree

. FCPS
Fin de course percage sortie

. FCPR
Fin de course percage rentree

GLOSSAIRE (MCC rotation broche percage)

. RotationBrochePercage
CR rotation broche percage

. DefRotBrochePercage
Defaut moteur percage

. ORotationBrochePercage
Ordre rotation moteur percage

. AcqDefRotPercage
Acquittement defaut moteur percage

. KP
Contacteur Moteur percage

Etat thermique percage
Contacteur Moteur percage

. THP
Thermique moteur percage disjoncte

. KPX
Retour contacteur moteur percage marche

GLOSSAIRE (MCC percage)

. FinPercage
Fin cycle percage

. DefPercage
Defaut percage

. PercagePret
Percage pret

. OPercage
Ordre cycle percage piece

. AcqDefUnite
Acquittement defaut

. AUUnite
Arret d'urgence

. ModeReglage
Mode manuel

. ORotationBrochePercage
Ordre rotation moteur percage

. AcqDefRotPercage
Acquittement defaut moteur percage

. RotationBrochePercage
CR rotation broche percage

. DefRotBrochePercage
Defaut moteur percage

. RMPS
BP reglage mouvement percage sortie

. RMPR
BP reglage mouvement percage rentree

. BrocheHautPercage
CR Broche en haut

. BrocheBasPercage
Cr broche en bas

. DefTransBrochePercage
Defaut verin percage

. ODescenteBrochePercage
Ordre descente brovhe percage

. OMonteeBrochePercage
Ordre montee broche percage

. AcqDefTransPercage
Acquittement defaut verin percage

		VERSION

Annexe C
Application SPEX

Chaque fonction "COORDONNER" est implémentée, sur l'outil SPEX, par un exemplaire de Boîte Fonctionnelle (BFE) paramétré en conséquence. L'ensemble des entrées et sorties de la fonction est importé puis utilisé pour l'instanciation d'un modèle générique de la bibliothèque.

Nous désirons, ici, créer un **prototype exécutable** de notre application par l'instanciation de modèles préexistants. Nous appliquons donc une démarche **ascendante** :

- pour chaque **mécanigramme**, nous créons un Diagramme Fonctionnel composé d'une part d'une BFE (pour la sous-fonction "COORDONNER" correspondante) implémentant un modèle "Filtre" d'Elément de Partie Opérative (cf Chapitre II §I.2.2) et, d'autre part d'un modèle d'émulation de l'élément opératif permettant, par la suite, une simulation en boucle fermée (cf Chapitre II §I.2.2) ;

- pour chaque niveau de la décomposition, en partant des niveaux les plus bas, nous créons un DF constitué d'une BFE (pour la sous-fonction "COORDONNER" correspondante) et des DFE, précédemmment créés, correspondant chacun à un sous-processus de ce niveau.

Nous réitérons cette démarche jusqu'au niveau le plus haut qui constitue alors notre prototype.

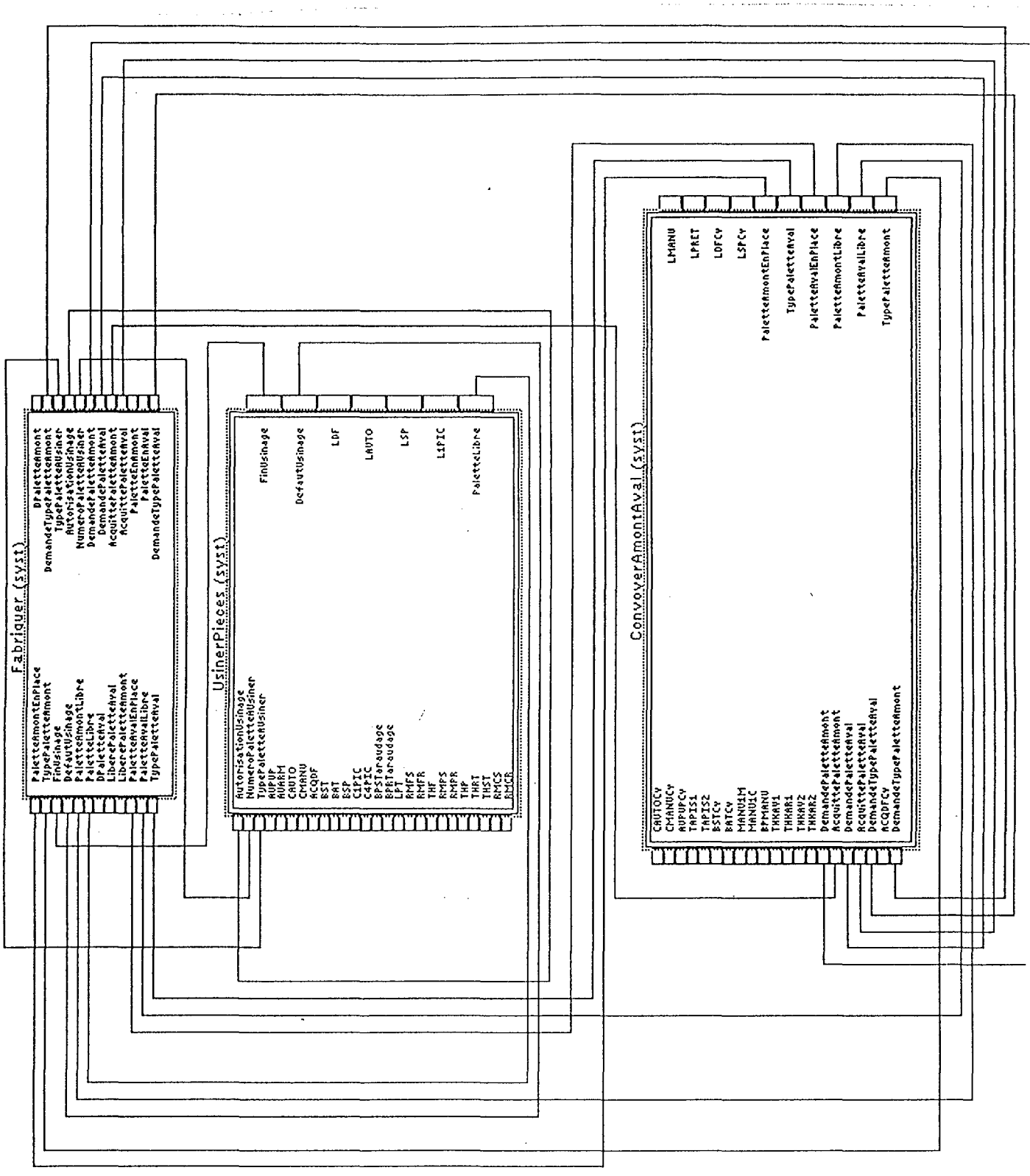


Figure C.1 : Vue synthétique de l'application

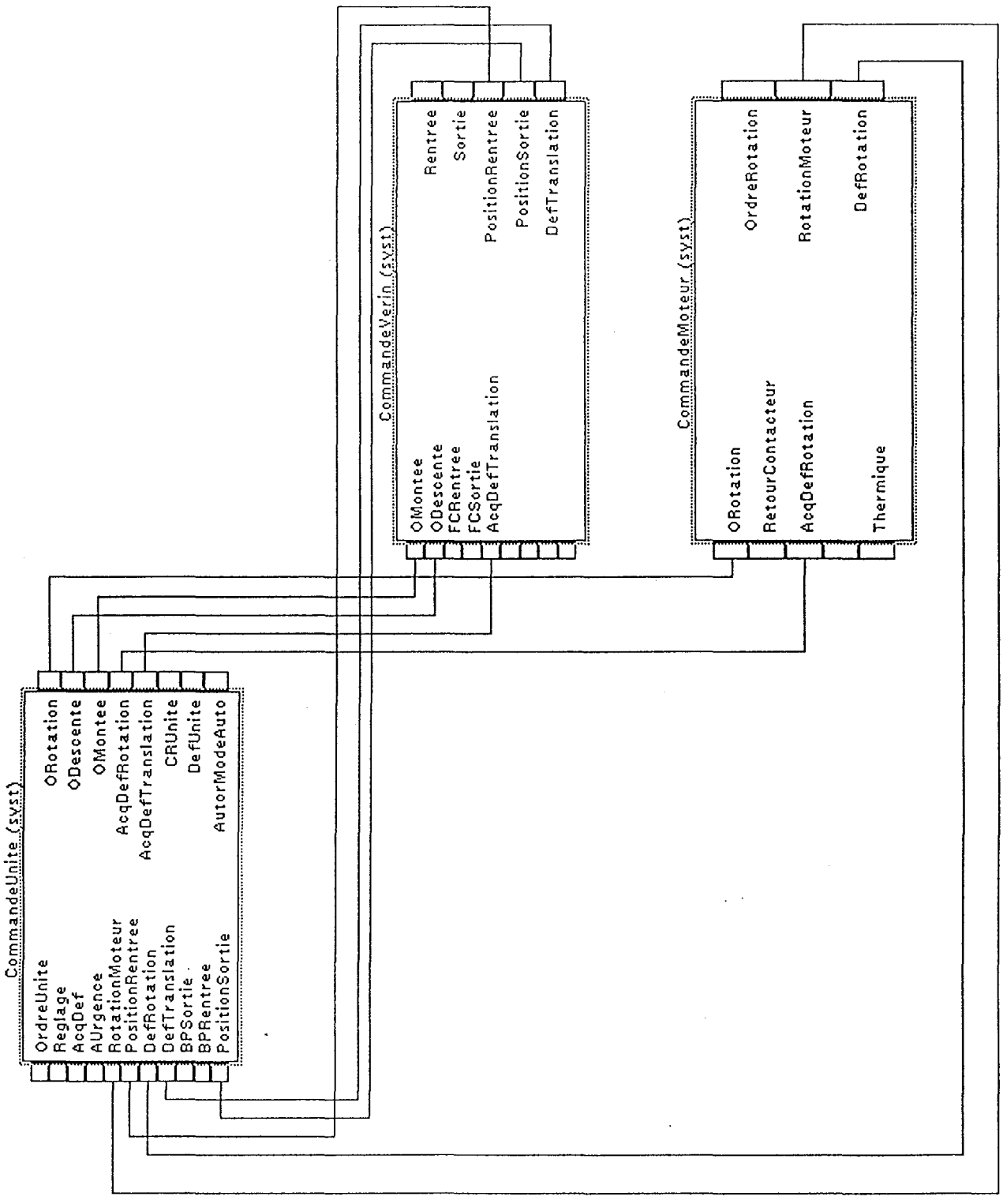


Figure C.2 : Vue synthéque d'une Unité

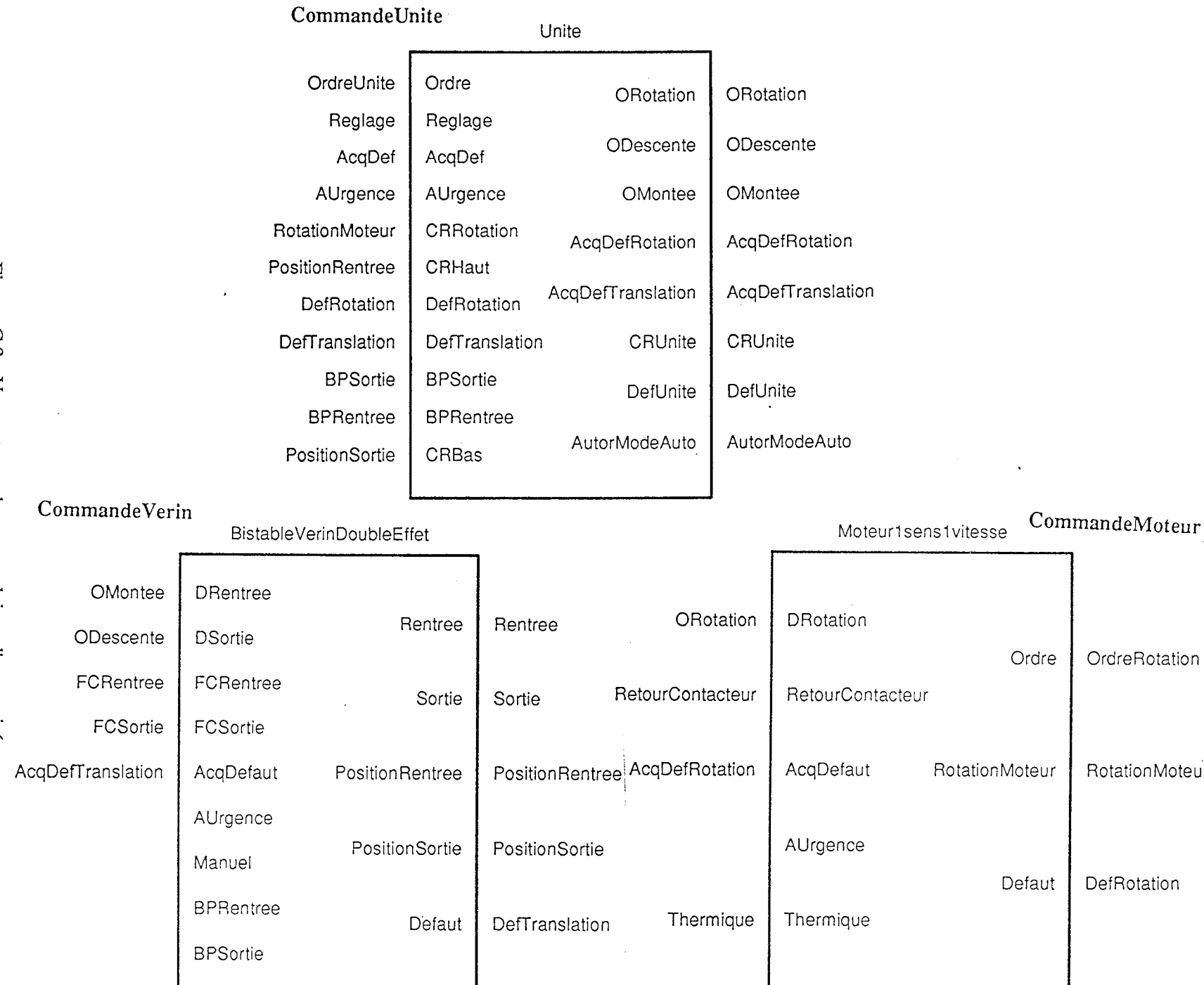
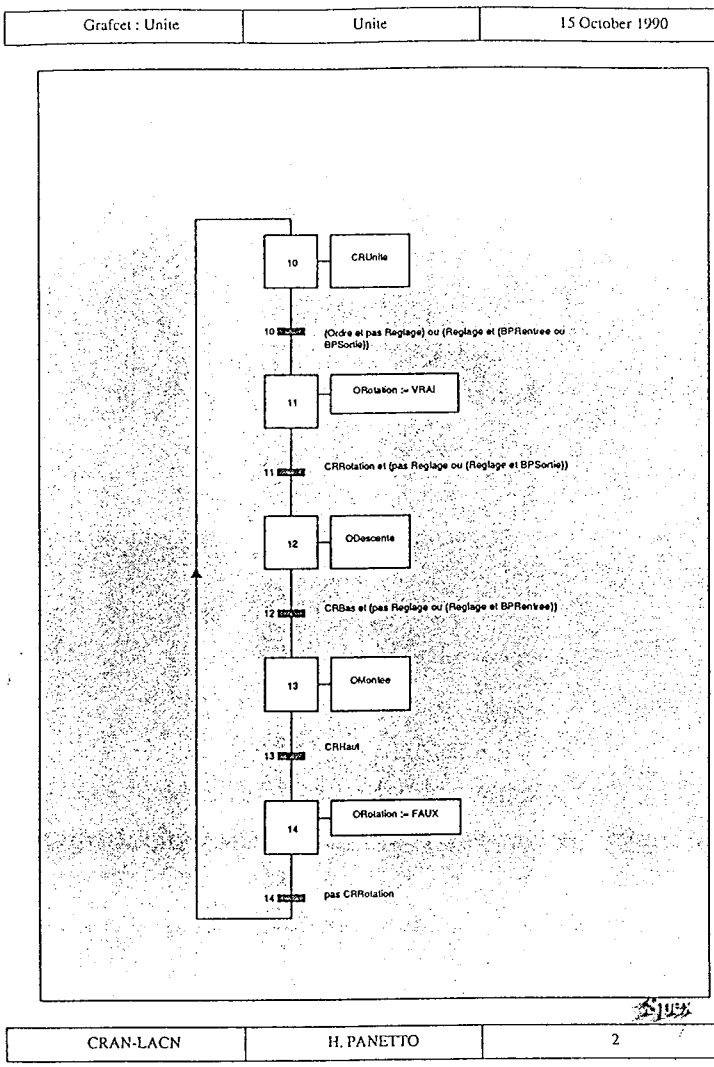
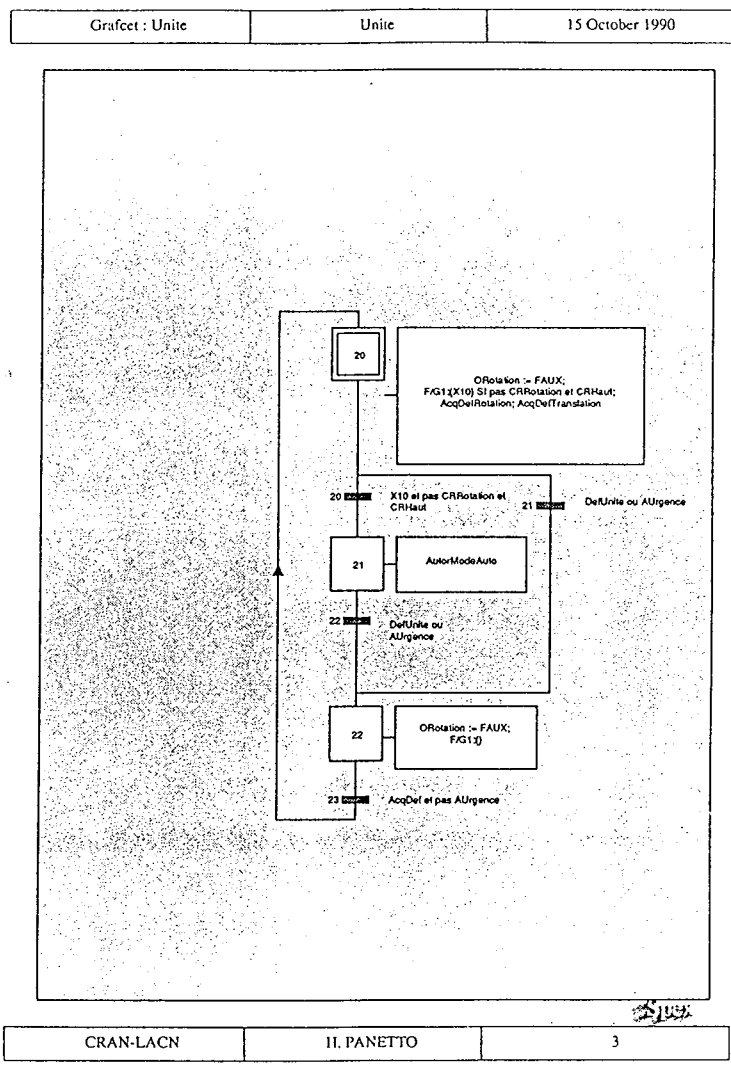


Figure C.3 : Vue externe des exemplaires d'une unité



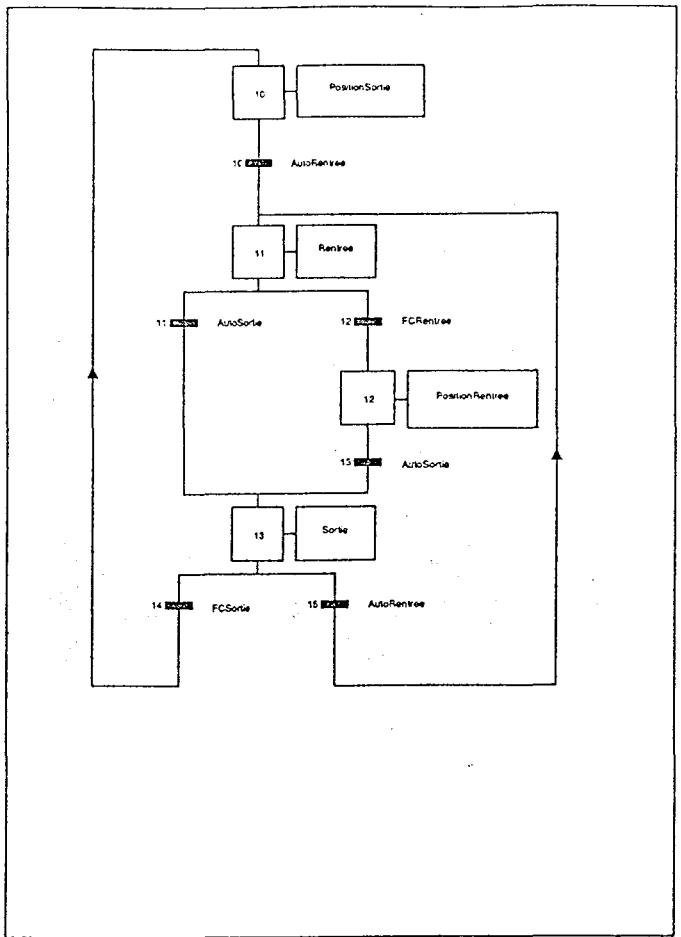


Combinatoires entree

DefUnite
Code :
DefUnite := DefRotation ou DefTranslation

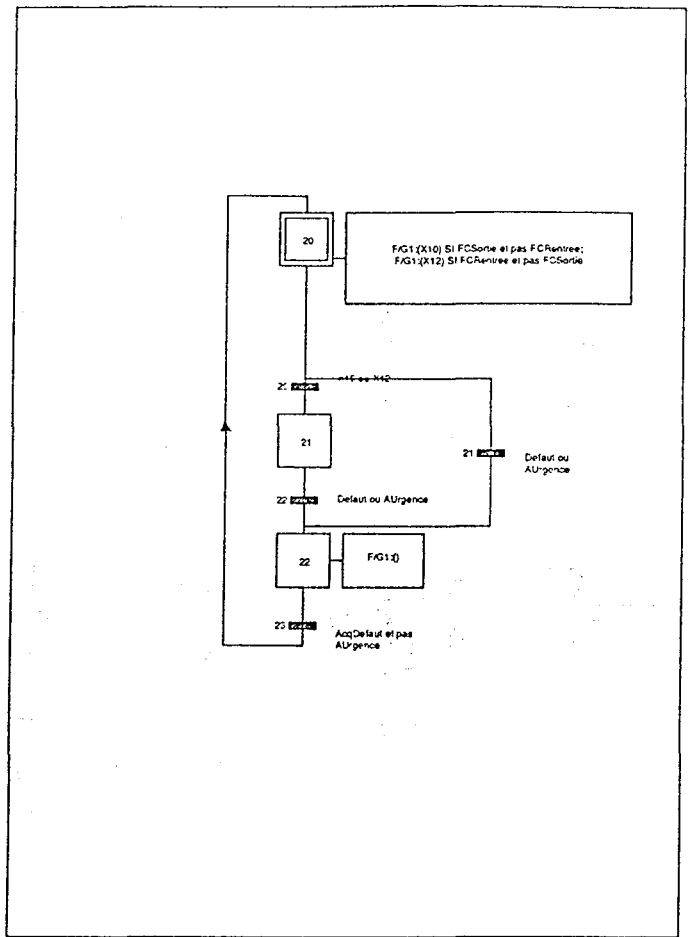
Figure C.4 : Vue interne de la BF générique "Commande Unité"

Grafcet : BistableVerinDoubleEffe	BistableVerinDoubleEffet	15 October 1990



CRAN-LACN	H.PANETTO	2

Grafcet : BistableVerinDoubleEffe	BistableVerinDoubleEffet	15 October 1990



CRAN-LACN	H.PANETTO	3

Combinatoires entree

Def1
Code :
Def1 := FAUX SI ^X20

Def2
Code :
Def2 := FAUX SI ^X20

Def3
Code :
Def3 := FAUX SI ^X20

Def4
Code :
Def4 := FAUX SI ^X20

Def5
Code :
Def5 := FAUX SI ^X20

Defaut
Code :
Defaut := FAUX SI ^X20

AutoSortie
Code :
AutoSortie := (DSortie et pas DRentree) ou (Manuel et BPSortie et pas BPRentree)

AutoRentree
Code :
AutoRentree := (DRentree et pas DSortie) ou (Manuel et BPRentree et pas BPSortie)

Combinatoires sortie

Def1
Code :
Def1 := VRAI SI X11 et T/X11/TpRentree

Def2
Code :
Def2 := VRAI SI X13 et T/X13/TpSortie

Def3
Code :
Def3 := VRAI SI FCRentree et FCSortie

Def4
Code :
Def4 := VRAI SI X10 et (pas FCSortie ou FCRentree)

Def5
Code :
Def5 := VRAI SI X12 et (pas FCRentree ou FCSortie)

Defaut
Code :
Defaut := Def1 ou Def2 ou Def3 ou Def4 ou Def5

Figure C.5 : Vue interne de la BF générique "BistableVerinDoubleEffet"

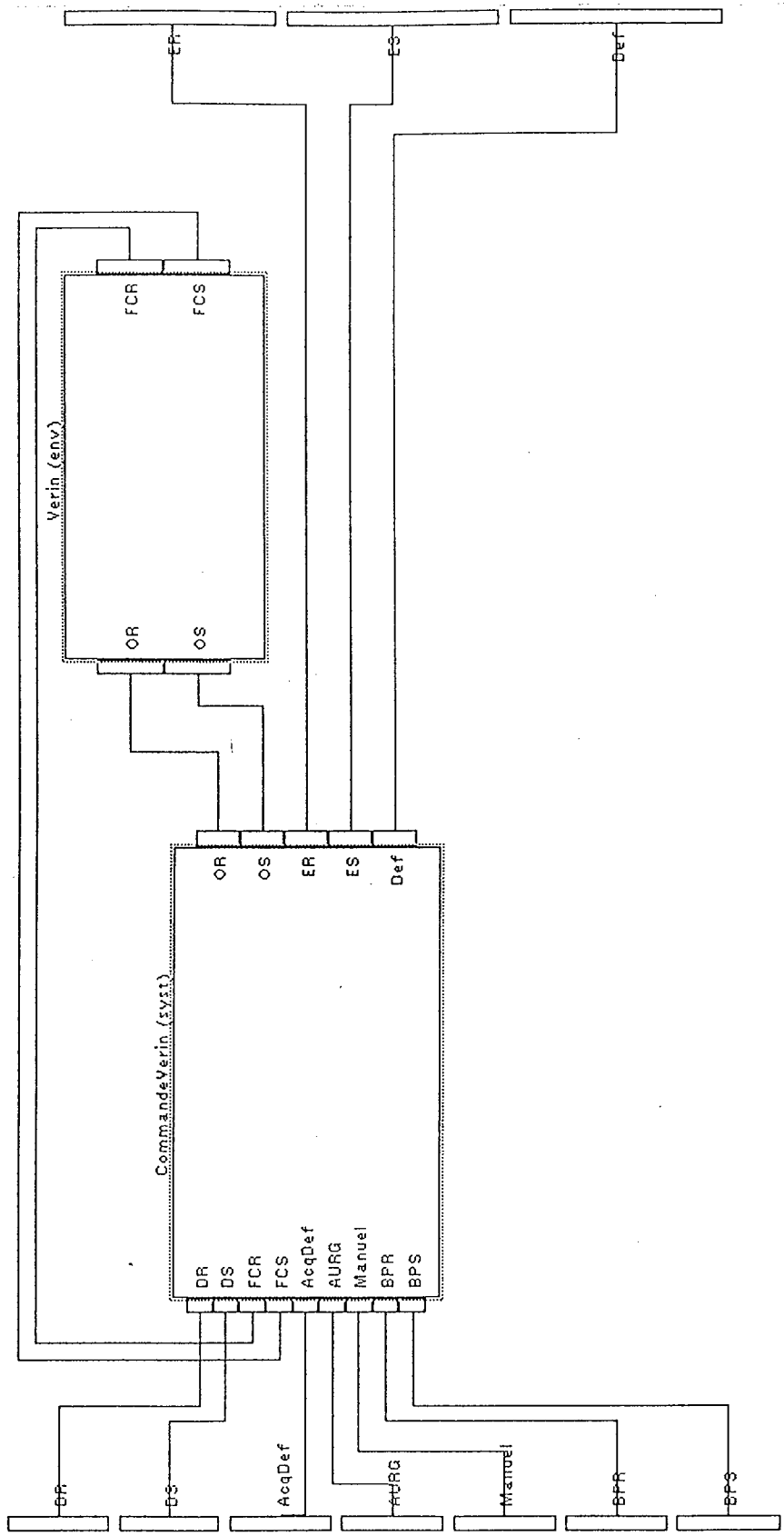
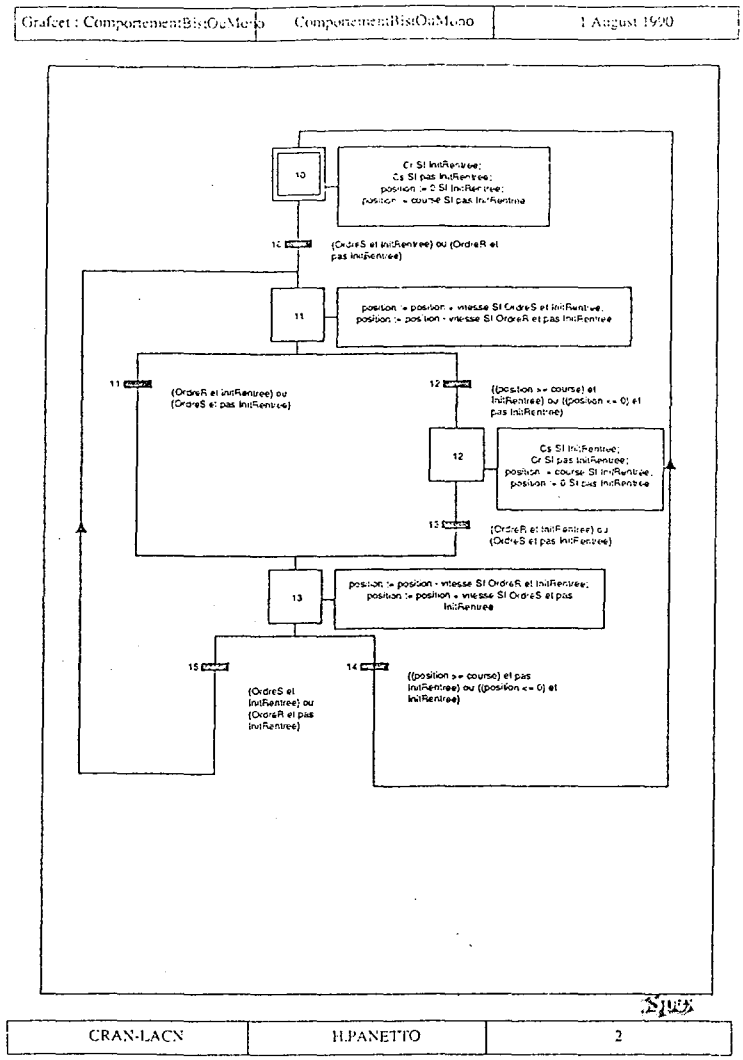


Figure C.6 : Utilisation d'un modèle d'EPO en émulation



Parametres

NOM	TYPE	VALEUR INITIALE
vitesse	Entier	2
course	Entier	10
InitRentree	Bool	VRAI
CommandeBistable	Bool	VRAI

Combinatoires entree

OrdreS
Code :
OrdreS := (Os et pas Or et CommandeBistable) ou (Os ou Or et pas CommandeBistable et InitRentree) ou (pas Os et pas Or et pas CommandeBistable et pas InitRentree)

OrdreR
Code :
OrdreR := (Or et pas Os et CommandeBistable) ou (Or ou Os et pas CommandeBistable et pas InitRentree) ou (pas Or et pas Os et pas CommandeBistable et InitRentree)

Figure C.7 : Vue interne de la BF générique
"Comportement Bistable ou Monostable" utilisée en émulation

Annexe D
Simulation de l'Application

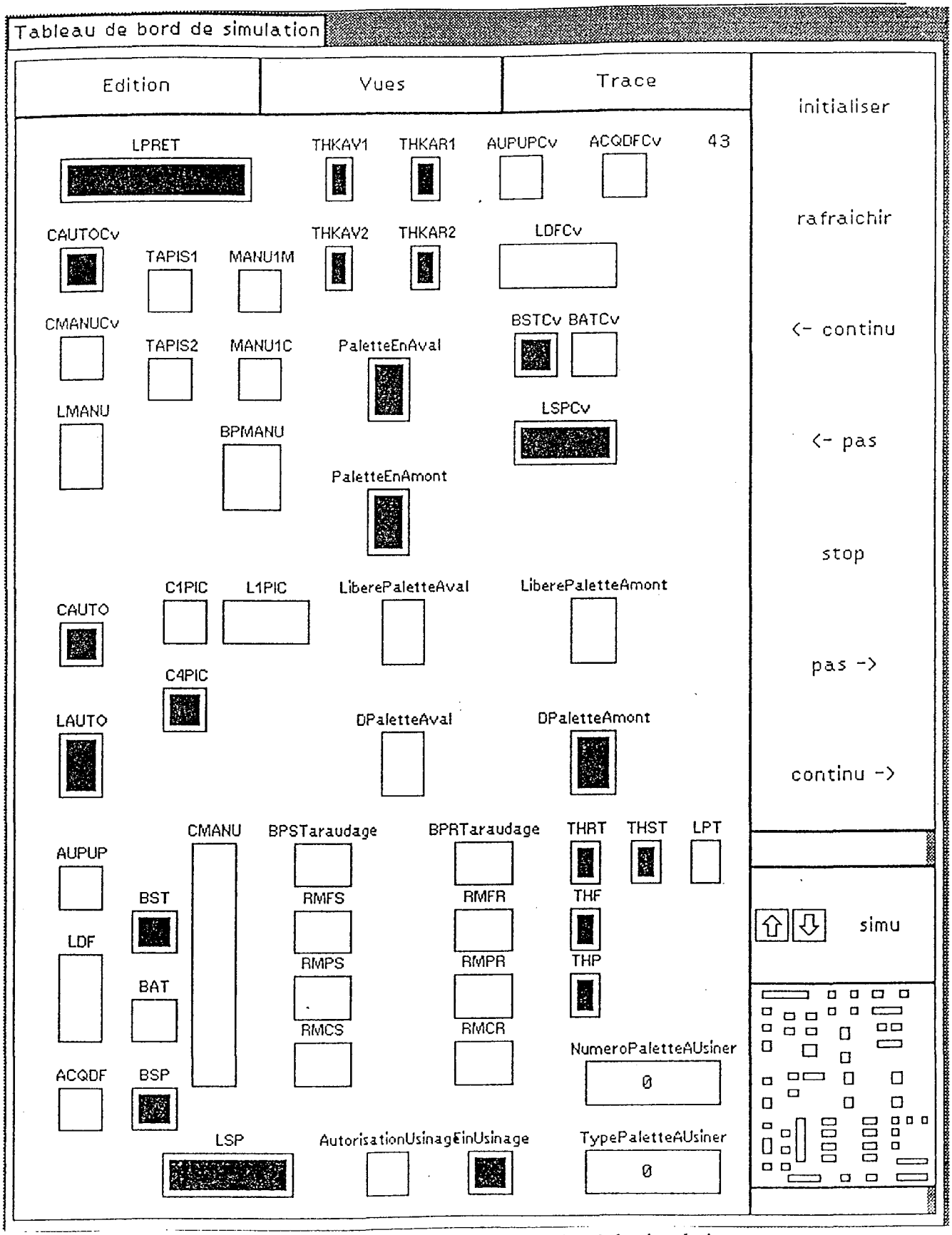


Figure D.1 : Le tableau de bord de simulation

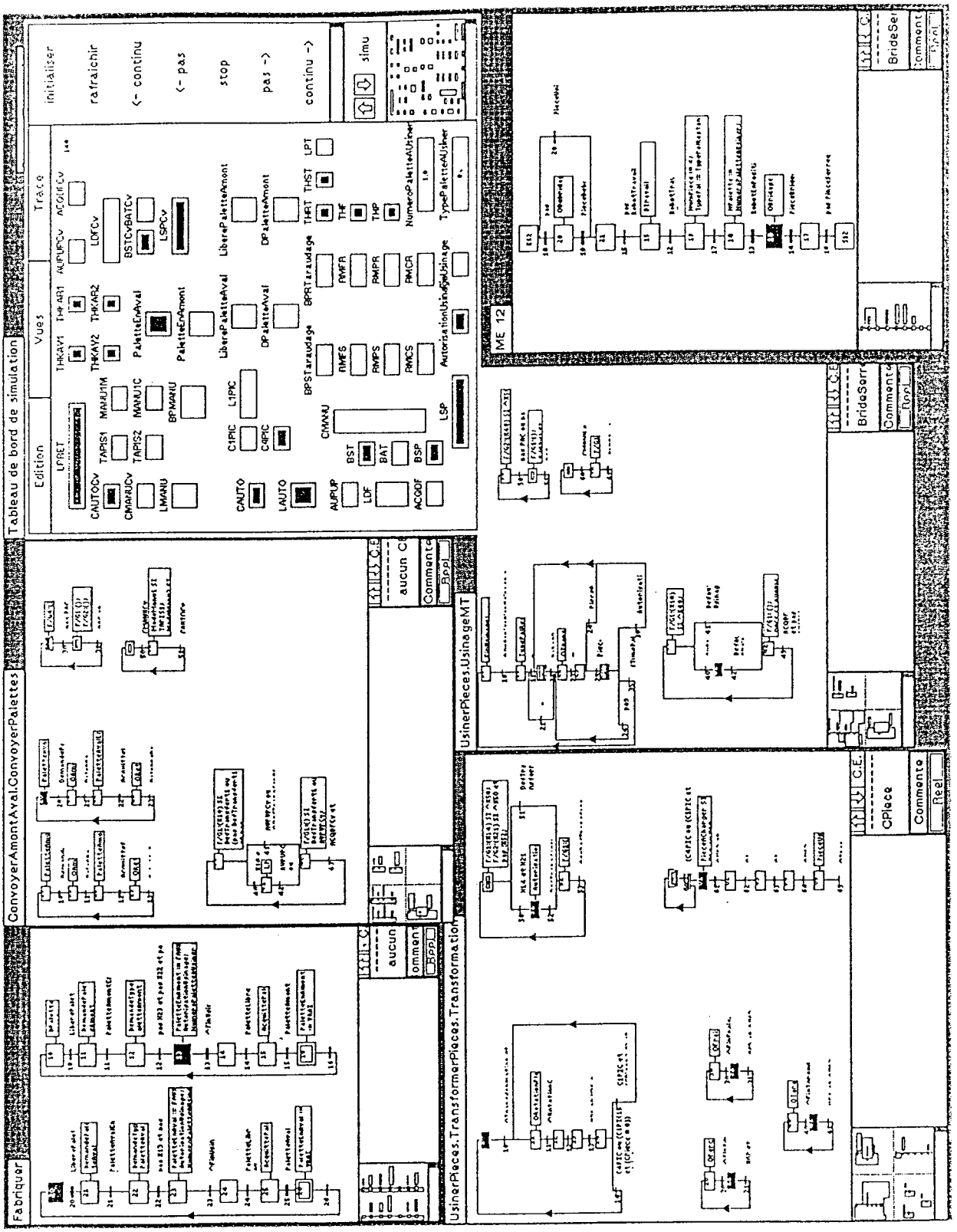


Figure D.2 : Une vue de la simulation

```
/*``includes``*/
#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <memory.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "PosteMT.h"

AexecuterScenario()
{
        switch (compteur) {
                case 1   :
                        booleenAffecter(_C4PIC,1);
                        booleenAffecter(_CAUTO,1);
                        booleenAffecter(_CAUTOCv,1);
                        booleenAffecter(_BSP,1);
                        booleenAffecter(_BST,1);
                        booleenAffecter(_BSTCv,1);
                        break;
                case 44  :
                        booleenAffecter(_LiberePaletteAmont,1);
                        break;
                case 65  :
                        booleenAffecter(_LiberePaletteAmont,0);
                        break;
                case 633:
                        booleenAffecter(_LiberePaletteAval,1);
                        break;
                case 672:
                        booleenAffecter(_LiberePaletteAval,0);
                        break;
                case 879:
                        booleenAffecter(_LiberePaletteAmont,1);
                        break;
                case 897:
                        booleenAffecter(_LiberePaletteAmont,0);
                        break;

        }
}
```

Figure D.3 : Un scénario de Test

Ce scénario alimente l'installation energie au cycle 1. Aux cycles 44 et 879, la machine en amont libère la palette pour permettre son déplacement vers la machine transfert. La palette aval est rendue à la machine au cycle 633.

Figure D.4 : Un extrait de la Trace textuelle résultante

Trace simulation Machine Transfert

xxx.Xn	etat etape n du graphe xxx
xxx.CXn	numero cycle d'activation de l'etape n du graphe xxx

Les noms de graphes sont précédés de la hiérarchie des noms de Diagrammes Fonctionnels les incluant.

Nom variable	Etat Précédent	Nouvel Etat
COMPTEUR CYCLE 1		
C4PIC (Bool)	0	1
CAUTO (Bool)	0	1
CAUTOCv (Bool)	0	1
BSP (Bool)	0	1
BST (Bool)	0	1
BSTCv (Bool)	0	1
UsinerPieces.TransformerPieces.AcqDefUnite (Bool)	0	1
UsinerPieces.TransformerPieces.TarauderMT.AcqDefRotTaraudage (Bool)	0	1
UsinerPieces.TransformerPieces.TarauderMT.FCTR (Bool)	0	1
UsinerPieces.TransformerPieces.Percage.AcqDefRotation (Bool)	0	1
UsinerPieces.TransformerPieces.Percage.AcqDefTranslation (Bool)	0	1
UsinerPieces.TransformerPieces.Percage.FCRentree (Bool)	0	1
UsinerPieces.TransformerPieces.Fraisage.AcqDefRotation (Bool)	0	1
UsinerPieces.TransformerPieces.Fraisage.AcqDefTranslation (Bool)	0	1
UsinerPieces.TransformerPieces.Fraisage.FCRentree (Bool)	0	1
UsinerPieces.RobotDegage (Bool)	0	1
PaletteEnAmont (Bool)	0	1
PaletteEnAval (Bool)	0	1
ConvoyerAmontAval.Transfert1.AcqDef (Bool)	0	1
ConvoyerAmontAval.Transfert1.PousserPalette.FCRentree (Bool)	0	1
ConvoyerAmontAval.Transfert1.BloquerDebloquer.FCSortie (Bool)	0	1
ConvoyerAmontAval.Transfert1.EjecterPalette.FCSortie (Bool)	0	1
ConvoyerAmontAval.Transfert1.PALET (Bool)	0	1
ConvoyerAmontAval.Transfert2.AcqDef (Bool)	0	1
ConvoyerAmontAval.Transfert2.PousserPalette.FCRentree (Bool)	0	1
ConvoyerAmontAval.Transfert2.BloquerDebloquer.FCSortie (Bool)	0	1
ConvoyerAmontAval.Transfert2.EjecterPalette.FCSortie (Bool)	0	1
ConvoyerAmontAval.Transfert2.PALET (Bool)	0	1
UsinerPieces.TransformerPieces.Percage.Moteur.OrdreR (Bool)	0	1
UsinerPieces.TransformerPieces.Fraisage.Moteur.OrdreR (Bool)	0	1
UsinerPieces.EnergieUsinage.EnergiePneumatique.OrdreR (Bool)	0	1
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.OrdreR (Bool)	0	1
ConvoyerAmontAval.Transfert1.BloquerDebloquer.Verin.OrdreS (Bool)	0	1
ConvoyerAmontAval.Transfert1.EjecterPalette.Verin.OrdreS (Bool)	0	1
ConvoyerAmontAval.Transfert2.PousserPalette.Verin.OrdreR (Bool)	0	1

ConvoyerAmontAval.Transfert2.BloquerDebloquer.Verin.OrdreS (Bool)	0	1
ConvoyerAmontAval.Transfert2.EjecterPalette.Verin.OrdreS (Bool)	0	1
ConvoyerAmontAval.EnergieTapis.EnergiePneumatique.OrdreR (Bool)	0	1
UsinerPieces.TransformerPieces.TarauderMT.MoteurVisSansFin.X20	1	0
Fabriquer.X16	1	0
Fabriquer.X26	1	0
ConvoyerAmontAval.Transfert1.DeplacerTapisMT.MoteurTapis.X20	1	0

.
.
.

COMPTEUR CYCLE 44

LiberePaletteAmont (Bool)	0	1
DPaletteAmont (Bool)	1	0
Fabriquer.X10	1	0
Fabriquer.X11	0	1
DemandePaletteAmont (Bool)	0	1
Fabriquer.CX11		44

COMPTEUR CYCLE 45

ConvoyerAmontAval.ConvoyerPalettes.X10	1	0
ConvoyerAmontAval.ConvoyerPalettes.X11	0	1
ConvoyerAmontAval.OAmener1 (Bool)	0	1
ConvoyerAmontAval.ConvoyerPalettes.CX11		45

.
.
.

COMPTEUR CYCLE 65

LiberePaletteAmont (Bool)	1	0
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.OrdreS (Bool)	0	1
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.OrdreR (Bool)	1	0
ConvoyerAmontAval.Transfert1.PousserPalette.FCRentree (Bool)	1	0
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.X10	1	0
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.X11	0	1
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.CX11		65
ConvoyerAmontAval.Transfert1.PousserPalette.Verin.position (Entier)	0	2

COMPTEUR CYCLE 66

ConvoyerAmontAval.Transfert1.PousserPalette.Verin.position (Entier)	2	4
UsinerPieces.TransformerPieces.Percage.Sortie (Bool)	1	0
UsinerPieces.TransformerPieces.Percage.CommandeVerin.X13	1	0
UsinerPieces.TransformerPieces.Fraisage.Sortie (Bool)	1	0
UsinerPieces.TransformerPieces.Fraisage.CommandeVerin.X13	1	0
UsinerPieces.TransformerPieces.Percage.CommandeVerin.X10	0	1
UsinerPieces.TransformerPieces.Fraisage.CommandeVerin.X10	0	1
UsinerPieces.TransformerPieces.Percage.PositionSortie (Bool)	0	1
UsinerPieces.TransformerPieces.Fraisage.PositionSortie (Bool)	0	1

UsinerPieces.TransformerPieces.Percage.CommandeVerin.CX10	241	317
UsinerPieces.TransformerPieces.Fraisage.CommandeVerin.CX10	241	317

.
.
.

COMPTEUR CYCLE 633

LiberePaletteAval (Bool)	0	1
Fabriquer.X20	1	0
Fabriquer.X21	0	1
DemandePaletteAval (Bool)	0	1
Fabriquer.CX21		633

COMPTEUR CYCLE 634

ConvoyerAmontAval.ConvoyerPalettes.X20	1	0
ConvoyerAmontAval.ConvoyerPalettes.X21	0	1
ConvoyerAmontAval.OAmener2 (Bool)	0	1
ConvoyerAmontAval.ConvoyerPalettes.CX21		634

.
.
.

Thèse de l'Université de NANCY I
Option "Production Automatisée"

Auteur : Hervé PANETTO
Etablissement : Université de NANCY I

Titre : "Une contribution au Génie Automatique : le Prototypage des Machines et Systèmes Automatisés de Production

Soutenue publiquement le 28 Janvier 1991 devant la Commision d'Examen composée de :

Président	R. HUSSON	Professeur à l'I.N.P.L. Directeur du C.R.A.N.
Rapporteurs	F. PRUNET	Professeur à l'U.S.T.L. Montpellier Directeur Dpt S.D.C.I.A. / L.A.M.M.
	J.F. AUBRY	Professeur à l'I.N.P.L.
Directeur de Thèse	M. VERON	Professeur à l'Université de Nancy I Directeur du L.A.C.N. / C.R.A.N.
Examinateurs	G. VILLERMAIN LECOLIER	Professeur à l'Université de Reims Directeur du L.A.M.
	F. CORBIER	Responsable Méthodes et Coordination des Moyens - SPIE AUTOMATION
	P. LHOSTE	Maître de Conférences à l'Université de Nancy I - L.A.C.N. / C.R.A.N.

RESUME :

L'émergence du Génie Automatique est la conséquence de besoins en méthodes et outils adaptés aux métiers de l'Automatisation. La mise en oeuvre des principes largement éprouvés par le Génie Logiciel tels que structuration, modularité, réutilisation, prototypage des applications contribue ainsi à l'évolution du Génie Automatique et à l'amélioration de la qualité de la conception. SPEX, outil de conception pour le Génie Automatique, offre ces possibilités mais nécessite, pour assurer une cohérence globale du processus de développement, de pouvoir s'intégrer, ou tout du moins s'interconnecter, avec des outils assurant les phases amont (spécification) et aval (codage et tests hors site) de l'application. A terme, la définition d'un Atelier de Génie Automatique devrait centraliser l'ensemble des informations dans une base de données ou d'objets commune à l'ensemble des outils pour permettre une mémorisation du savoir-faire entreprise.

MOTS CLES : Méthode - Méthodologie - Modèles - SADT - Prototypage - Maquettage - Génie Automatique - Atelier Logiciel - Grafcet - Conception - Outil de Conception

ABSTRACT :

Automation Engineering appearance results from needs of methods and tools for Automation "life cycle". Software Engineering concepts such as structuration, modularity, reutilisability, application prototype conception make a contribution to Automation Engineering evolution and to conception quality amelioration. SPEX tool implements these possibilities but requires, for global consistency of developpement process, to become integrated, or connected, with specification, code production and test tools. At short-dated, the definition of an Automation Engineering CASE would centralize information on common data or object base for know-how memorization.

KEYWORDS : Method, Models, SADT, Prototype conception, Automation Engineering, CASE tool, State charts, Conception, Conception tool